End-to-End Video Captioning

Silvio Olivastri[1], Gurkirt Singh[2] and Fabio Cuzzolin[2]
[1]AI Labs, Bologna, Italy [2]Oxford Brookes University, United Kingdom
silvio@ailabs.it, gurkirt.singh-2105@brookes.ac.uk, fabio.cuzzolin@brookes.ac.uk

Abstract

Building correspondences across different modalities, such as video and language, has recently become critical in many visual recognition applications, such as video captioning. Inspired by machine translation, recent models tackle this task using an encoder-decoder strategy. The (video) encoder is traditionally a Convolutional Neural Network (CNN), while the decoding (for language generation) is done using a Recurrent Neural Network (RNN). Current state-of-the-art methods, however, train encoder and decoder separately. CNNs are pretrained on object and/or action recognition tasks and used to encode video-level features. The decoder is then optimised on such static features to generate the video's description. This disjoint setup is arguably sub-optimal for input (video) to output (description) mapping.

In this work, we propose to optimise both encoder and decoder simultaneously in an end-to-end fashion. In a two-stage training setting, we first initialise our architecture using pre-trained encoders and decoders – then, the entire network is trained end-to-end in a fine-tuning stage to learn the most relevant features for video caption generation. In our experiments, we use GoogLeNet and Inception-ResNet-v2 as encoders and an original Soft-Attention (SA-) LSTM as a decoder. Analogously to gains observed in other computer vision problems, we show that end-to-end training significantly improves over the traditional, disjoint training process. We evaluate our End-to-End (EtENet) Networks on the Microsoft Research Video Description (MSVD) and the MSR Video to Text (MSR-VTT) benchmark datasets, showing how EtENet achieves state-of-the-art performance across the board.

1. Introduction

Video captioning is the problem of generating textual descriptions based on video content, a key functionality to pave the way for, e.g., talking cars, surgical robots or factories. The task is particularly challenging for approaches should capture not only the objects, scenes, and activities present in the input video (i.e. address video tagging, object and action recognition), but also express how these objects, scenes, and activities relate to each other in a spatial and temporal fashion using a natural language construction.

Two major approaches to video captioning exist [31]: template-based language models and sequence learning-based ones. The former class of methods detect words from the visual content (e.g. via object detection) to then generate the desired sentence using grammatical constraints such presence of a subject, verbs, object triplets, and so on. Interesting studies in this sense were conducted in [7, 20, 35]. The latter group of approaches, instead, learn a probability distribution from a set of feature vectors extracted from the video to flexibly generate a sentence without using any specific language template. Examples of this second category of approaches are [28, 36, 30].

Encoder-decoder frameworks. Thanks to the recent developments of deep learning frameworks such as Long Short-Term Memory (LSTM) [9] networks and Gated Recurrent Units (GRU) [4], as well as of machine translation techniques such as [23], the currently dominant approach to video captioning is based on sequence learning in an *encoder-decoder* framework. In this setting, the encoder represents the input video sequence as a fixed-dimension feature vector, which is then fed to the decoder to generate the output sentence one word at a time. At each time step in the decoder, the current input (the word) and the previously generated hidden states of the output sequence are used to predict the next word and the hidden state.

One of the most severe drawbacks of such models, however, is that the underlying video content feature space is static and does not change during the training process. More specifically, an encoder (typically, a Convolutional Neural Network, CNN) is pre-trained on datasets designed for different tasks, to be then used as feature extractor for video-captioning. Although this makes some sense given the multi-task nature of the video captioning process illustrated above, the resulting disjoint training process, in which the decoder is trained on the captioning task with static features as input, is inherently suboptimal. Recent state-of-the-art methods [36, 17, 38] try to address this issue by capturing

dynamic temporal correspondences between feature vectors corresponding to different video frames. However, these works do not address the basic fact that video captioning may well require the system to learn task-specific features that will necessarily differ from those learned for the action classification, video tagging or object detection tasks for which the CNN encoder was trained.

From complex decoders to learning task-specific features. Our view is that *a decoder designed for video captioning should instead be able to learn from task-specific features*. As shown in Figure 1 (A) in all previews architectures the encoder is never trained, (e.g., the gradient is not updated during the learning of the encoder part). In addition, the CNN implementing the encoding is trained using a different loss function aimed at solving a different task.

In order to compensate for this fundamental flaw, new decoding mechanisms must be implemented to learn better video features. As a result decoders (and the associated training procedures) have consistently increased in complexity over the years. To illustrate this point: Venugopalan *et al*. [28] (2015) used a vanilla LSTM; Yu *et al*. [37] (2016) multiple Gate Recurrent Units with two attention mechanisms; Pan *et al*. [15] (2017) implemented a Transfer Unit in combination with two LSTMs; finally, PickNet [3] (2018) achieved excellent results using reinforcement learning to train a deep net able to identify the most relevant frames.

Our approach is in radical contrast with these previous attempts. Rather than designing new, expensive decoding mechanisms to learn better video features, we force the encoder's feature extractor to focus its attention on 'significant' objects in order to generate good textual descriptions, exploiting the ability of convolutional networks to extract relevant features from images or videos.

Our proposal: end-to-end training. We propose to address this problem by bringing forward the end-to-end training of both encoder and decoder as a whole, as shown in Figure 1 (B). Our philosophy is inspired by the success of end-to-end trainable network architectures in image recognition [10], image segmentation [12], object detection [19] and image-captioning [13, 29], but *was never before adopted in a video captioning setting*.

The reason is that, if done naively, training end-to-end the resulting large scale, heterogeneous network has a prohibitive computational cost, especially on machines with a limited number of GPUs. Introducing the technique into video captioning, however, is quite imperative for end-to-end training allows for simple inference [29] and can handle complex, multi-task problems best described by multiple losses [16].

Two-stage learning for efficient end-to-end training. An efficient training procedure is then crucial to unlock the potential of end-to-end training for video captioning.

In this paper we address this issue in two ways.

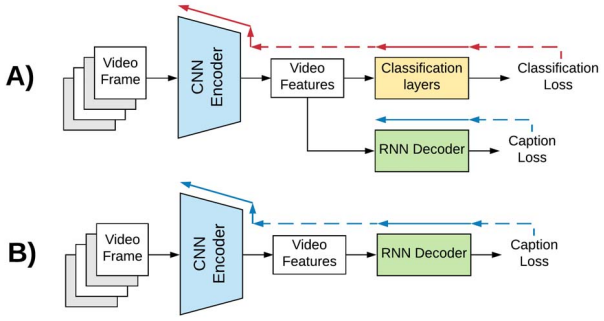

Figure 1. Gradient flow comparison between the disjoint training of the RNN decoder and the CNN encoder (A) and the end-to-end training of both encoder and decoder (B). In the first case (A), the CNN encoder does not update its parameters in dependence of the captioning loss, but as a function of just the classification loss. Only the decoder updates its parameters as a function of the gradient of the captioning loss (blue arrow). In the end-to-end case considered here (B), the parameters of both CNN and RNN are updated according to the evolution of the gradients from the captioning loss. Gradient flow is again depicted by a blue arrow.

Firstly, we propose a *two-stage learning process*, in which encoder and decoder are first trained separately, after initialisation from disjoint models, to leave fine tuning to be conducted in the second stage in a fully end-to-end fashion. In second place, during training we accumulate gradients over multiple steps in order to update parameters only after the required effective batch size is achieved. This leads to an increased number of iterations, which is however mitigated by our two-stage approach.
Additionally, to further improve performance, we propose significant changes to the Soft-Attention decoder by Yao *et al*. [36], while preserving its simplicity of concept. The major improvements we propose are presented in Section 3.2, and encompass structural architectural changes, a different computational graph and an averaged loss applied to the attention coefficients for caption generation, inspired by [34].

Contributions. Summarising, to the best of our knowledge: (1) our work presents the first End-to-End trainable framework (EtENet) for video captioning, designed to learn task-specific features, based on a new two-stage efficient training strategy. Our approach propagates the gradient from the last layer of the RNN decoder to the first layer of the CNN encoder as illustrated in Figure 1 (B). Unlike [38, 30, 15, 36] which all use more than 25 frames per video clip, our model only uses 16 frames, a significant contribution in terms of resource economics.
(2) Performance-wise, training our network architecture in the traditional, disjoint way produces comparable results to the current state-of-the-art, whereas our end-to-end training framework delivers significant performance improvements no matter the choice of the base encoder network. This *sets a new benchmark for the field of video captioning*,

upon which further progress can be made. Notably, this is achieved without using any 3D CNN encoding.
Finally (3), inspired by [36], we present and plug into our end-to-end framework an overhauled version of the Soft-Attention decoder characterised by improved performance compared to the original version.

2. Related work

Inspired by the latest computer vision and machine translation techniques [23], recent efforts in the video captioning field follow a *sequence learning* approach. The commonly adopted architecture, as mentioned, is an encoder-decoder framework [31] that uses either 2D or 3D CNNs to collect video features in a fixed-dimension vector, which is then fed to a Recurrent Neural Network to generate the desired output sentence, one word at a time.

The first notable work in this area was done by Venugopalan *et al*. [28]. They represented an entire video using a mean-pooled vector of all features generated by a CNN at frame level. The resulting fixed-length feature vector was then fed to an LSTM for caption generation. Although state-of-the-art results were achieved at the time, the temporal structure of the video was not well modelled in this framework. Since then, alternative views have been supported on how to improve the visual model in the encoder-decoder pipeline. The same authors [27] have later proposed a different method which exploits two-layer LSTMs as both encoder and decoder. In this setting, compared to the original pipeline [28], each frame is used as input at each time step for the encoder LSTM, which takes care of encoding the temporal structure of the video into a fixed size vector. This model, however, still leaves room for a better spatiotemporal feature representation of videos, as well as calling for improved links between the visual model and the language model.

To address this problem, 3D CNNs and attention models have been since introduced. Inspired by [34], Yao *et al*. [36] employ a temporal Soft Attention model in which each output vector of the CNN encoder is weighted before contributing to each word's prediction. The spatial component is extracted using the intermediate layers of a 3D CNN which is used in combination with the 2D CNN. On their part, [37] have proposed a spatiotemporal attention scheme which includes a paragraph generator and sentence generator. The paragraph generator is designed to pick up the sentence's ordering, whereas the sentence generator focuses on specific visual elements from the encoder.

A distinct line of research has been brought forward by Pam and his team in [15] and subsequently in [17]. The first work tries, in addition to using features from both 2D and 3D CNNs, to introduce a visual semantic embedding space for enforcing the relationship between the semantics of the entire sentence and the corresponding visual content. Multiple Instance Learning models have been used in [5] for detecting attributes to feed to a two-layer LSTM controlled by a transfer unit.

In the last two years, numerous relevant papers have been published. Similarly to [17], Zhang *et al*. [38] use a task-driven dynamic fusion across the LSTM to process the different data types. The model adaptively chooses different fusion patterns according to task status. Xu *et al*. [33] test on the MSR-VTT dataset a Multimodal Attention LSTM Network that fuses audio and video features before feeding the result to an LSTM multi-level attention mechanism. In [6], the authors create a Semantic Compositional Network plugging into standard LSTM decoding the probabilities of tags extracted the frames, in addition to the usual video features, merged in a fixed-dimension vector. Chen *et al*. [3] show that it is possible to get good results using just $\sim$6-8 frames. An LSTM encoder takes a sequence of visual features while a GRU decoder helps generate the sentence. The main idea of this interesting work is to use reinforcement learning, while a CNN is used to discriminate whether a frame must be encoded or not. A recent work [30] improves the performance of [36] and its model architecture by inserting a reconstruction layer on top that aims to replicate the video features, starting from the hidden state of the LSTM cell.

Unlike all previous work, in which the weights of the encoder part do not change, our study focusses on end-to-end training. This strategy, which has been proven successful in various applications, encourages the encoder to capture features which are actually discriminant for caption generation. Our training process is divided into two stages: while in the first stage only the decoder is trained, in the second one the whole model is fine-tuned. In this work, in particular, we test two different 2D CNN encoders: GoogLeNet [25] and Inception-ResNet-v2 [24], thus showing how end-to-end training is beneficial no matter the choice of the base encoder. This also demonstrates that our training strategy is not linked to any particular encoder network. As decoder, we present a modified version of the Soft-Attention model defined by [36]. Nevertheless, the approach is entirely general and can be applied to other decoding architectures.

3. Approach

This section describes in detail our end-to-end trainable encoder-decoder architecture for video-captioning. The overall framework is depicted in Figure 2. Section 3.1 illustrates the chosen encoder, based on the Inception-ResNet-v2 [24] architecture. The decoder, based on our original Soft-Attention LSTM (SA-LSTM) design inspired by [36], is discussed in § 3.2. Difference with [36] and initialisation details are also discussed there. Note that our framework is also tested using GoogLeNet [25] as the encoder.

The training process is explained in § 3.3. Firstly, gradi-

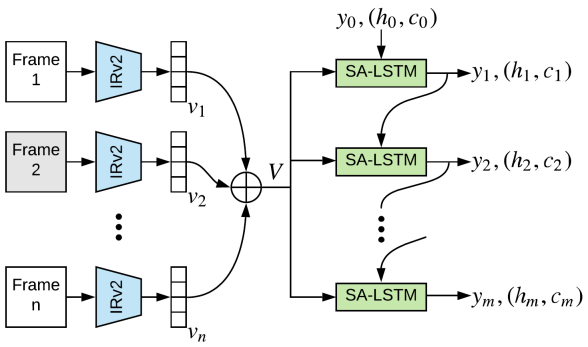

Figure 2. Diagram of our EtENet-IRv2 architecture. Each frame is processed individually using Inception-ResNet-v2 (IRv2). All resulting vectors $v_1, ..., v_n$ are concatenated ($\oplus$) to represent their collection V. At each decoding time step t a single word is predicted by SA-LSTM, based on the vector V, the previous word y_{t-1} and the hidden states (h_{t-1}, c_{t-1}).

ent accumulation is used to achieve the desired high batch size for the training of the decoder. Secondly, the proposed two-stage training process is described. Lastly, the proposed averaged loss functions are defined and justified.

3.1. Encoder

A common strategy [28] is to use a 2D CNN pre-trained on the ImageNet dataset [21] as an encoder. Typically, feature vectors are generated before the first fully-connected layer of the neural network, for each frame of the video. This is done as a preprocessing step, and many versions of 2D CNN were brought forward over the course of the years for video captioning. For instance, Venugopalan *et al*. [28, 27] would use variants of AlexNet [10], 16 layer VGG [22] and GoogLeNet [25]. Yao *et al*. [36] would also use GoogLeNet, in combination with a 3D CNN. Gan *et al*. [6], instead, preferred ResNet-152 [8] whereas Wang *et al*. [30] used Inception-v4 [24].

As noted by Wang *et al*. [30], deeper networks are more likely to capture the high-level semantic information about the video. Thus, in this work we decided to use Inception-ResNet-v2 [24] as the encoder, among all possible convolutional network architectures. The version used in our experiments is pre-trained on ImageNet, and is publicly available[1]. To show the generality of our framework, additional tests were conducted using our own PyTorch version of GoogLeNet [25], also pre-trained on ImageNet. The weights were imported from a well known Caffe version[2].

Formally, given a video $X = \{x_1, ..., x_n\}$ composed by a sequence of n RGB images, our encoder ϕ is a function mapping each frame x_i to a feature vector v_i using the average pooling stage after the `conv2d_7b` layer of Inception-ResNet-v2 or, in the case of GoogLeNet, the `pool5/7x7 s1` layer. We thus denote by

$$V = \{v_1, ..., v_n\} = \phi(\{x_1, ..., x_n\}) \tag{1}$$

the output of the encoder, later fed as input to the decoder.

3.2. Decoder: Soft Attention LSTM

Using Inception-v4 as decoder, the Soft-Attention model developed by Yao *et al*. [36] achieves good performance compared to other, more complex systems (e.g., [37]).
For this reason our work builds on the version of [36] developed in Theano[3]. The resulting original SA-LSTM framework, implemented in PyTorch, exhibits a number of significant improvements over [36], explained in detail below.

Formulation. At each decoding step the SA-LSTM ψ takes as input n vectors generated by the encoder, $V = \{v_1, ..., v_n\}$, together with the previous hidden state h_{t-1}, memory cell c_{t-1} and word y_{t-1}. Its output is composed by: (i) the probability of the next word $P(y_t|y_{<t}, V)$ based on the previously observed words $y_{<t}$ and on the feature vectors V; (ii) the current hidden state h_t, and (iii) the current memory cell state c_t. Namely:

$$\begin{bmatrix} P(y_t|y_{<t}, V) \\ h_t \\ c_t \end{bmatrix} = \psi(V, y_{t-1}, h_{t-1}, c_{t-1}). \tag{2}$$

The algorithm runs sequentially through the output sequence, predicting one word at a time.

More in detail, at any given time t the first step is to create a single vector from V by applying a Soft-Attention mechanism φ_t to the whole encoder output. Firstly, for each vector v_i a normalised score α_i^t is computed:

$$e_i^t = W_a^\top \tanh(W_{eh} h_{t-1} + W_{ev} v_i + b_e) + b_a; \tag{3}$$

$$\alpha_i^t = \frac{\exp\{e_i^t\}}{\sum_j^{|V|} \exp\{e_j^t\}}, \tag{4}$$

where W_a, W_{eh}, W_{ev}, b_e, b_a are all trainable variables and e_i^t is the unnormalised score of vector i at time t.
Secondly, a coefficient $\beta_t \in [0, 1]$ is computed to measure the importance of the final vector $\varphi_t(V)$ as a function of the previous hidden state, with parameters W_β and b_β:

$$\beta_t = \sigma(W_\beta h_{t-1} + b_\beta), \tag{5}$$

where σ is a sigmoid activation function. The final output of the Soft-Attention function is computed as follows:

$$\varphi_t(V) = \beta_t \sum_i^{|V|} \alpha_i^t v_i. \tag{6}$$

[1] `https://github.com/Cadene/pre-trained-models.pytorch`

[2] `https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet`

[3] `https://github.com/yaoli/arctic-capgen-vid`

The vector (6) is then concatenated with an embedding E of the previous word y_{t-1}

$$z_t = [\varphi_t(V), E[y_{t-1}]], \tag{7}$$

and fed to a standard LSTM with state vector h_t:

$$(h_t, c_t) = LSTM(z_t, h_{t-1}, c_{t-1}). \tag{8}$$

The word prediction $p_t = P(y_t|y_{<t}, V)$ is a function of the concatenation of $\varphi_t(V)$ and h_t, and of the embedding $E[y_{t-1}]$ of y_{t-1}:

$$u_t = W_u[\varphi_t(V), h_t] + E[y_{t-1}] + b_u; \tag{9}$$

$$p_t = softmax(W_p \tanh(u_t) + b_p), \tag{10}$$

where $[\cdot, \cdot]$ denotes vector concatenation and all weight matrices W and bias vectors b are trainable parameters.

Decoder innovations. Our decoder architecture differs from [36]'s in a number of ways. (i) In Equation (9), $E[y_{t-1}]$ is not mapped by a matrix parameter. The effect of this is that the visual information adapts to the word embedding, rather than the opposite, as in residual connection frameworks [8]. (ii) In Equation (5), a β_t term is added to reflect the fact that for some connective words in the sentence (e.g. 'the') the attention term should weigh less. (iii) An averaged Doubly Stochastic Attention is used as attention loss (as discussed in Sec. 3.3), for this has shown to improve performance. Implementation-wise, (iv) the LSTM machinery (Equations (7) to (8)) is derived from `torch.nn.LSTMCell`, rather than having been implemented from scratch.

Another substantial difference between the two models lies in their computational graphs at training time. In [36] Soft-Attention is a *sentence-by-sentence* model, meaning that the dimensionality of the decoder input is (batch_size, sequence_length, embedding_dim+encoder_output_dim). In our decoder we use a *word-by-word approach* with as input shape: (batch_size, embedding_dim+encoder_output_dim). The latter first computes the entire sequence of the LSTM hidden states $\{h_t\}$ to then produce a sentence. The former generates one hidden state vector at a time, and the word is directly predicted at every step. This difference affects the gradient upgrading procedure during back-propagation.

Initialization Details. Our framework takes blocks of 16 RGB frames as input. Each frame is processed by our version of Inception-ResNet-v2 up to the average pooling stage after the `conv2d_7b` layer. Thus, the encoder output V is composed by 16 vectors of 1536 elements each. In the GoogLeNet case we use the output of the `pool5/7x7 s1` layer, so that the vector has 1024 elements.
As explained, at each step the decoder takes as input V, the previously observed word y_{t-1} and the hidden h_{t-1} and c_{t-1} states. The first word of every predicted sentence is the token `<SOS>`, while h_0 and c_0 are initialised as follows:

$$h_0 = \tanh(W_h\overline{V} + b_h); \tag{11}$$

$$c_0 = \tanh(W_c\overline{V} + b_c), \tag{12}$$

where W_h, W_c, b_h, b_c, are trainable variables and $\overline{V}$ is the mean of all the vectors in V.
The desired video caption is predicted word by word until `<EOS>` is produced or after a maximum caption length is reached (set to 30 for MSVD and to 20 for MSR-VTT). The input V is the same throughout each iteration. We use 512 as the dimension of the LSTM hidden layer, 486 as embedding dimension for $E[y_{t-1}]$, while the cardinality of the word probability vector p_t obviously depends on the size of the vocabulary being considered ($\sim$12,000 for MSVD, $\sim$200,000 for MSR-VTT).

3.3. Training Process

Accumulate to Optimize. Recurrent networks require a large (e.g. 64 in [36]) batch size to converge to good local minima. This is true for our SA-LSTM as well, since it is based on an LSTM recurrent architecture.
In our initial tests, when using a disjoint training setup similar to [36]'s, we noticed that increasing the batch size would indeed boost performance. Unfortunately, Inception-ResNet-v2 (as other CNNs) is very expensive in term of memory requirements, hence large batch sizes are difficult to implement. A single batch, for instance, would use 5 GB (GigaByte) of GPU memory. The machine our tests were conducted on comes with 4 Nvidia P100 GPUs with 16 GB of memory each, allowing a maximum batch size of 12.

To overcome this problem, our training strategy is centred on accumulating gradients until the neural network has processed 512 examples. After that, the accumulated gradients are used to update the parameters of both encoder and decoder. The pseudocode for this process is provided in Algorithm 1. The standard training process is modified into one that accumulates gradients for $accumulate\_step$ size. As a result, the approach achieves an effective batch size equal to $accumulate\_step \times mini\_batch\_size$.

Two-Stage Training. Stochastic optimisers require many parameter update iterations to identify a good local minimum. Hence, if naively implemented, our gradient accumulation strategy would be quite slow, as opposed to disjoint training in which GPU memory requirements are much lower. To strike a balance between a closer to optimal but slower end-to-end training setup and a faster but less optimal disjoint training setting we adopt a *two-stage training* process, which is also crucial to allow end-to-end training of an heterogenous network from a computational standpoint.

In the first stage, we freeze the weights of the pre-trained encoder to train the decoder. As the encoder's weights are

Algorithm 1 Training with accumulated gradient.

Require: $accumulate\_step$

$i \leftarrow 0$
Reset gradient to zero
for batch size of Examples in Training set **do**
  Model forward step using Examples
  Compute loss
  Normalise loss using $accumulate\_step$
  Backward step and accumulate gradients
  if $(i \mod accumulate\_step)$ is 0 **then**
    Update model with accumulated gradient
    Reset gradient to zero
  end if
  $i \leftarrow i + 1$
end for

kept constant, this is equivalent to training a decoder on precomputed features from the encoder. As a result, memory requirements are low, and the process is fast. Once the decoder reaches a reasonable performance on the validation set, the second stage of the training process starts.
In the second stage, the whole network is trained end-to-end while freezing the batch normalisation layer (if the encoder contemplates it).

In both stages, at each time step SA-LSTM uses the real target output (i.e., the target word) as input, rather than its own previous prediction.
Given the heterogeneity of the architecture, we use Adam [9] as an optimisation algorithm and different parameter values for encoder and decoder. For the former, inspired by [25, 24], since the batch size is 512 and each example has 16 frames, we set the learning rate to $1e-09$ and the weight decay to $4 * 1e-05$ in the experiments with Inception-ResNet-v2. The version with GoogLeNet uses a learning rate of $2e-04$ and a weight decay of $2 * 1e-04$. The decoder is instead updated using $1e-04$ as learning rate and $1e-04$ as weight decay. To avoid the vanishing and exploding gradient problems typical of RNNs, we force the gradient to belong to the range $[-10, 10]$. At test time a beam search [14] with size 5 is used for final caption generation as in [30]. Larger sizes do not improve performance.

Loss Function. Similarly to [36, 34], we adopted as overall loss of the network the following sum:

$$\mathcal{L}_{tot}(\theta) = \mathcal{L}(\theta)_{NLL} + \lambda \mathcal{L}(\theta)_{aDSA}. \tag{13}$$

The Negative Log-Likelihood loss is given by: $\mathcal{L}(\theta)_{NLL} = -\sum_i^N \sum_t^C \log p(y_t^i | y_{<t}^i, x_i, \theta)$, where C is the caption length, N the number of examples. The second component is an original *averaged Doubly Stochastic Attention* loss:

$$\mathcal{L}(\theta)_{aDSA} = \frac{1}{L} \sum_k^L \left(1 - \sum_t^C \alpha_k^t\right)^2, \tag{14}$$

which we found improves performance over the standard DSA one, with L the size of the feature vector v_i. The (average) DSA component of the loss can be seen as encouraging the model to pay equal attention to every frame over the course of the caption's generation process.

Similarly to [36], we set λ to 0.70602.

4. Evaluation

Before discussing our tests, we first describe the metrics (§ 4.1) and datasets (§ 4.2) used to evaluate our model, and the pre-processing steps applied to the input data (§ 4.3).

4.1. Metrics

To guarantee a fair quantitative comparison with the state of the art we used the most common and well known metrics: BLEU [18] (4-gram version), METEOR [1], ROUGE-L [11] and CIDEr [26]. While BLEU and METEOR were created for machine translation tasks, ROUGE-L's aim is to compare a machine-generated summary with the human-generated sentence. CIDEr is notable as the only metric created for evaluating image descriptions that use human consensus.

4.2. Datasets

We evaluated our model and compared it with our competitors on two standard video captioning benchmarks. MSVD is one of the first such datasets to include multi-category videos. MSR-VTT, on its side, is based on 20 categories and is of a much larger scale than MSVD.

MSVD. The most popular dataset for video captioning systems evaluation is, arguably, the Microsoft Video Description Corpus (MSVD), also known as YoutubeClips [2]. The dataset contains 1970 videos, each video depicting a single activity lasting about 6 to 25 seconds. Each video is associated with multiple descriptions in the English language, collected via Amazon Mechanical Turk, for a total of 70,028 natural language captions. We adopted the evaluation setup of [28], and splitted the dataset into three parts: 1,200 videos for training, 100 videos for validation, and the remaining 670 videos for testing.

MSR-VTT. The MSR Video to Text [32] dataset is a recent large-scale benchmark for video captioning. 10K video clips from 20 categories were collected from a commercial video search engine (e.g., music, sports, and TV shows). Each of these videos was annotated with 20 sentences produced by 1327 Amazon Mechanical Turk workers, for a total number of captions of around 200K.
As prescribed in the original paper, we split videos by index number: 6,513 for training, 497 for validation and 2,990 for the test. The number of unique words present in the captions is close to 30K.

4.3. Preprocessing

Following the usual pre-processing of Inception-ResNet-v2, height and width of each frame of the video were resized to 314, to then use the central crop patch of 299x299 pixels of each frame. Pixel normalisation using a mean and standard deviation of 0.5 was applied. Training, validation and test examples were all subject to the same frame preprocessing steps. For the GoogLeNet encoder, we used the commonly accepted preprocessing steps for that network: each frame of the video was resized to 224x224 pixels, and then normalised by mean subtraction.
Using all frames of a video is very time inefficient – as [3] shows, it is possible to create an efficient model using fewer frames. On the other hand, we did not apply any additional filtering to the frames, as we preferred to leave this task for the attention mechanism to handle. In agreement with [30] and with our findings, we decided to represent each video by 16 equally-spaced features.

As for the captions, we tokenised them by converting all words to lowercase and applying the `TreebankWordTokenizer` class from the Natural Language Toolkit[4] to split sentences into tokens. The tokeniser uses regular expressions as in the Penn Treebank[5], thus adhering to English grammar while maintaining punctuation in the token.

5. Experiments

We conclude by reporting and discussing the experimental validation of our end-to-end trainable framework (EtENet) on the datasets described in (§ 4.2).

5.1. SA-LSTM vs Soft Attention

As a first step, we compared the performance of our SA-LSTM$_{base}$ decoder (in PyTorch) with that of Soft Attention [36] (Theano), with the same parameter values: $learning\_rate = 0.01$, $batch\_size = 64$, Adadelta as optimizer, and using the original DSA loss [34] rather than our bespoke, averaged version. The version of our decoder we term SA-LSTM$_{best}$, instead, improves on SA-LSTM$_{base}$ by using different hyperparameter values, optimisation algorithm and loss. Namely, the best results are achieved using $learning\_rate = 0.0001$, $batch\_size = 512$, Adam as optimizer and our averaged DSA loss (14). Note that the original approach requires a lot of memory, so that the higher batch size possible on our machines was 64. Using our model, instead, we could achieve a value of 512. Note also that we only used one GPU for training for sake of fair comparison, as Theano does not support multi-GPU training. The results are shown in Table 1.

[4] `https://www.nltk.org`

[5] `http://www.cis.upenn.edu/~treebank/tokenizer.sed`

For this comparison we used the features extracted from GoogLeNet stored in the original SA repository [36]. Thus, the results are not comparable with those of Table 2.

Model	B@4	M	C	R-L
SA	46.6	32.0	67.0	68.0
SA-LSTM$_{base}$	46.9	32.1	70.9	69.2
SA-LSTM$_{best}$	48.3	32.2	76.4	69.1

Table 1. Comparison between the original Soft-Attention (SA) decoder and ours (SA-LSTM), on the test set of the MSVD dataset.

Both versions of our decoder outperform [36] – by a very significant amount in the optimised (best) version, especially in the CIDEr metric.

5.2. State-of-the-art Comparison

Tables 2 and 3 clearly show how our approach, both when using only step 1 of the training, and when applying both steps, matches or outperforms all the work done previously using Inception-ResNet-v2 as the encoder (EtENet-IRv2), except when measured using the BLEU metric. In fact, as explained by Banerjee *et al*. [1], BLEU is a metric that has many weaknesses, e.g., the lack of explicit word-matching between translation and reference.
In opposition, according to [26], CIDEr was specifically designed to evaluate automatic caption generation from visual sources, and is thus arguably more relevant. Indeed, our proposed EtENet-IRv2 outperforms all the existing state-of-the-art method across both datasets (see Tables 2, 3) when performance is measured by CIDEr.

5.3. Discussion

The substantial difference between our model and the others assessed confirms that EtENet-IRv2 succeeds in achieving excellent results without requiring an overly complex structure, e.g., the addition of new layers as in RecNet (row 11, Table 2), or the adoption of new learning mechanisms such as reinforcement learning as in PickNet (row 3, Table 3). Moreover, this shows that it is possible to obtain excellent results even when using roughly half the frames used in other competing approaches [36, 33, 38, 30]. *Our framework sets a new standard in terms of top performances in video captioning* and, we believe, can much contribute to further progress in the field. Additionally, this is done without resorting to fancy 3D CNN architectures, thus leaving huge scope for further improvements. Moreover, unlike [38, 30, 15, 36] which all use more than 25 frames per video clip, our model only uses 16 frames, a significant contribution in terms of memory and computational cost.

The performance of EtENet-GLN, which uses GoogLeNet as encoder, is comparable to that of all mechanism using similar or older versions of the decoder

Model	B@4	M	C	R-L
LSTM-YT [28]	33.3	29.1	-	-
S2VT [27]	-	29.8	-	-
SA [36]	41.9	29.6	51.7	-
LSTM-E [15]	45.3	31.6	-	-
h-RNN [37]	49.9	32.6	65.8	-
LSTM-TSA [17]	52.8	33.5	74.0	-
SCN-LSTM [6]	51.1	33.5	77.7	-
MA-LSTM [33]	52.3	33.6	70.4	-
TDDF [38]	45.8	33.3	73.0	69.7
PickNet [3]	**52.3**	33.3	76.5	69.6
RecNet [30]	**52.3**	34.1	80.3	69.8
EtENet-GLN$_{step1}$	48.2	32.0	75.1	68.9
EtENet-GLN$_{step2}$	48.9	32.4	78.0	69.4
EtENet-IRv2$_{step1}$	49.1	33.6	83.5	69.5
EtENet-IRv2$_{step2}$	50.0	**34.3**	**86.6**	**70.2**

Table 2. Comparison between our architecture (EtENet), using both GoogLeNet (GLN) and IRv2 as the encoder, and the state-of-the-art on the MSVD dataset.

(e.g., VGG) [28, 27, 36, 15, 37, 17, 33]. Notably, though, it does achieve its best results in the CIDEr metric.

5.4. Impact of End-to-End Training

Importantly, for both incarnations of our architecture (EtENet-GLN e EtENet-IRv2) end-to-end training (step 2) positively impact the performance across the board (i.e., across datasets and metrics), thanks to the additional fine tuning. Our network is able to match or outperform the other state-of-the-art models, while very significantly outperforming the Soft-Attention (SA) approach [36] (third row of Table 2).

Model	B@4	M	C	R-L
MA-LSTM [33]	36.5	26.5	41.0	59.8
TDDF [38]	37.3	**27.8**	43.8	59.2
PickNet [3]	**41.3**	27.7	44.1	59.8
RecNet [30]	39.1	26.6	42.7	59.3
EtENet-IRv2$_{step1}$	40.3	27.5	46.8	60.4
EtENet-IRv2$_{step2}$	40.5	27.7	**47.6**	**60.6**

Table 3. Comparison between our EtENet-IRv2 architecture and the state-of-the-art on the MSR-VTT benchmark, using the following metrics: BLEU-4, METEOR, CIDEr and ROUGE-L.

5.5. Qualitative results

From a qualitative point of view, Figure 3 reports both some positive and some negative examples. Generally, we can notice that the increase in accuracy achieved by the two-step training setting leads, in some cases, to a visible improvement of the generated sentences.

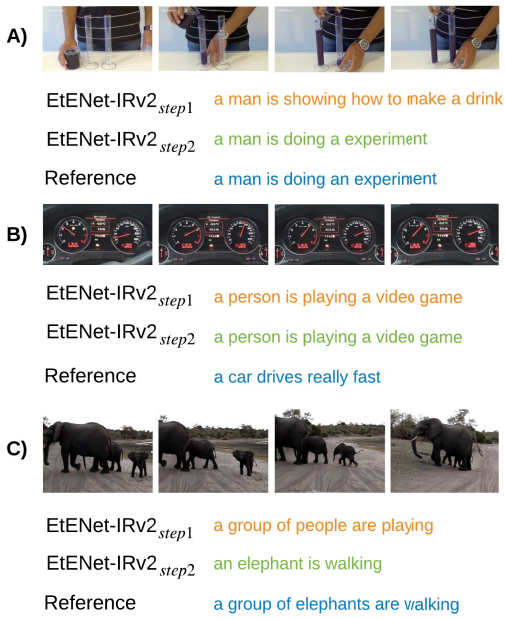

Figure 3. Qualitative results produced by EtENet-IRv2. In (A) and (C), which show a video from MSR-VTT and one from MSVD, respectively, it is possible to observe how end-to-end training can dramatically improve the quality of the resulting caption. In (B) a negative example from the MSR-VTT dataset is shown, for which our network cannot successfully identify the ground truth.

Much more extensive quantitative results are reported in the Supplementary Material.

6. Conclusions

In this paper, we proposed a simple end-to-end framework for video-captioning. To address the problem with the large amount of memory required to process video data for each batch, a gradient accumulation strategy was conceived. We proposed a training procedure articulated into two steps to speed up the training process, and allow efficient end-to-end training. Our evaluation on standard benchmark datasets showed how our approach outperforms the state of the art using all the most commonly accepted metrics.
We believe we managed to set a new baseline for future work thanks to our principled end-to-end architecture, providing an opportunity to take research in the field forward starting from a more efficient training framework.

Our model is not exempt from drawbacks. Training a very deep a neural network end-to-end requires significant computational resources. Our proposed two-stage training process is a step towards an efficient training procedure suited to the task. Further research directions include the integration of a more formal treatment of language semantics in the model, and spatio-temporal attention mechanisms based on state of the art action and object tube detectors.

References

[1] S. Banerjee and A. Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgements. *ACLworkshop*, pages 358–373, 2005.

[2] D. L. Chen and W. B. Dolan. Collecting highly parallel data for paraphrase evaluation. *ACL HLT*, pages 190–200, 2011.

[3] Y. Chen, S. Wang, W. Zhang, and Q. Huang. Less is more: Picking informative frames for video captioning. *ECCV*, pages 358–373, 2018.

[4] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014.

[5] H. Fang, S. Gupta, F. Iandola, R. Srivastava, L. Deng, P.Dollar, J. Gao, X. He, M. Mitchell, J. C. Platt, C. L. Zitnick, and G. Zweig. From captions to visual concepts and back. *CVPR*, 2015.

[6] Z. Gan, C. Gan, X. He, Y. Pu, K. Tran, J. Gao, L. Carin, and L. Deng. Semantic compositional networks for visual captioning. *CVPR*, 2017.

[7] 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. *ICCV*, pages 2712–2719, 2013.

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

[9] S. Hochreiter and J. J. J. Schmidhuber. Long short-term memory. *Neural Computation*, pages 1–15, 1997.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[11] C. Lin. Rouge: A package for automatic evaluation of summaries. *ACLworkshop*, 2004.

[12] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[13] J. Lu, C. Xiong, D. Parikh, and R. Socher. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. *CVPR*, pages 3242–3250, 2017.

[14] P. Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*, chapter 6, pages 195–200. Morgan Kaufmann, 1992.

[15] Y. Pan, T. Mei, T. Yao, H. Li, , and Y. Rui. Jointly modeling embedding and translation to bridge video and language. *CVPR*, pages 4594–4602, 2016.

[16] Y. Pan, Z. Qiu, T. Yao, H. Li, and T. Mei. To create what you tell: Generating videos from captions. *ACM*, 2017.

[17] Y. Pan, T. Yao, H. Li, and T. Mei. Video captioning with transferred semantic attributes. *CVPR*, 2017.

[18] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a 521 method for automatic evaluation of machine translation. *ACL*, pages 311–318, 2002.

[19] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *NIPS*, 2015.

[20] A. Rohrbach, M. Rohrbach, W. Qiu, A. Friedrich, M. Pinkal, and B. Schiele. Coherent multi-sentence video description with variable level of detail. *GCPR*, pages 184–195, 2014.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Karpathy, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2014.

[23] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *NIPS*, pages 3104–3104, 2014.

[24] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI*, 2017.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CVPR*, 2015.

[26] R. Vedantam, C. Lawrence, and D. Parikh. Cider: consensus-535 based image description evaluation. *CVPR*, 2015.

[27] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko. Sequence to sequencevideo to text. *ICCV*, pages 4534–4542, 2015.

[28] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural network. *NAACL-HLT*, pages 1494–1504, 2015.

[29] O. Vinyals, A. Tosheva, S. Bengio, and D. Erhan. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *TPAMI*, 2016.

[30] B. Wang, L. Ma, W. Zhang, and W. Liu. Reconstruction network for video captioning. *CVPR*, pages 7622–7631, 2018.

[31] Z. Wu, T. Yao, Y. Fu, and Y. Jiang. Deep learning for video classification and captioning. In S. Chang, editor, *Frontiers of Multimedia Research*, pages 3–29. ACM Books, 2018.

[32] J. Xu, T. Mei, T. Yao, and Y. Rui. Msr-vtt: A large video description dataset for bridging video and language. *CVPR*, pages 5288–5296, 2016.

[33] J. Xu, T. Yao, Y. Zhang, and T. Mei. Learning multimodal attention lstm networks for video captioning. *ACM*, 2017.

[34] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *ICML*, 2015.

[35] R. Xu, C. Xiong, W. Chen, and J. J. Corso. Jointly modeling deep video and compositional text to bridge vision and language in a unified framework. *AAAI*, pages 2346–2352, 2015.

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

[37] H. Yu, J. Wang, Z. Huang, Y. Yang, and W. Xu. Video paragraph captioning using hierarchical recurrent neural networks. *CVPR*, pages 4584–4593, 2016.

[38] X. Zhang, K. Gao, Y. Zhang, D. Zhang, J. Li, and Q. Tian. Task-driven dynamic fusion: Reducing ambiguity in video description. *CVPR*, pages 3713–3721, 2017.