Neural Diary: Forming Compressed Visual Stories in Real-Time

Edward H. Lee
Stanford University
Stanford, CA
edhlee@stanford.edu

Abstract

In many applications, video data contains temporally-redundant information about context that relates to who, what, and where. In many applications it is not only sufficient to sub-sample frames but necessary on a mobile device. We design a video captioning system on the embedded TX1 that takes an ultra-long video stream (15 minutes to 2 hours) that samples salient frames in time, generates captions on the objects of these frames, and generates spatial attention maps that attend to these captions.

We have three main contributions. First, since the main computational bottleneck is in the LRNN, we design a temporal sampler using CNN features to sample key frames and to minimize the number of LRNN forward passes. This allows us to decrease the number of sampled frames by $> 100\times$*, speed-up runtime by* $7.4\times$*, and increase energy efficiency by* $8\times$*. Second, in each temporally-salient frame, we caption the frame and compute an attention map that attends to the region of interest from the captions. Third, we provide low-level optimization using matrix factorization in the LRNN with retraining allowing* $1.2\times$ *speedup on the TX1 using only* $9W$ *of power and explore FPGA acceleration on the CNN.*

1. Introduction

We as humans are actively bombarded with visual information in a given day. And yet we are still able to retain and highlight key personal experiences in the context of who, what, where, and when of salient events that are added to our long-term memory. Inspired by nature, we ambitously aim to build a computationally efficient system that can automatically highlight certain salient events from long-term video data in a compact representation of just a few salient images and generated sentences. These samples then form a compressed diary that represents the video seqeuence. We deploy this on the real-time embedded TX1 [1].

As a long term vision, our eventual goal is to allow people to wear the camera module for extended lengths of time to store salient events and to tell a story about these events in the form of image-caption pairs. This has numerous applications such as automating journalism and assisting the blind. To demonstrate this, we need to prioritize both network performance, such as the ability to caption images, and hardware performance, such as computational and energy efficiencies.

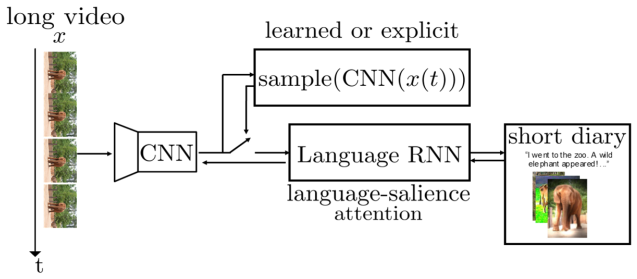


Figure 1: Neural Diary pipeline in the embedded TX1 that takes a video sequence x and returns a compressed representation of the video in a form of a few salient images, attention maps, and descriptions.

The main system objective is to take a video sequence $x \in \mathbb{R}^{H\times W\times C\times T}$ with height, width, and color channels H, W, C and total frames T of length > 15 minutes and generate a short diary containing $\ll T$ salient images and captions using less than 10W of power at > 1 frame-per-second inference on the embedded TX1.

Motivated by simpler design and computational constraints, we constrain the network to operate on individual frames as shown in Fig. 1. Each frame $x^t \in \mathbb{R}^{H\times W\times C}$ is passed through the Convolutional Neural Network (CNN). The controller determines which frames x^t for $t \in \{1,\ldots,T\}$ are temporally salient. If salient, the CNN(x^t) is passed to the Language Recurrent Neural Network (LRNN). This controller also highlights key regions in x^t used in the final diary using only one backward pass through the CNN-LRNN network. This method of finding which parts in x^t have the highest influence in the generated captions efficiently reuses the original network. Therefore the controller serves two roles: 1) highlights parts of the video salient in time and 2) highlights small key regions of each sampled x^t salient in the space. Finally, we also provide low-level optimization using 1) matrix factorization with retraining on the LRNN and 2) field-programmable gate array (FPGA) implementation for the CNN forward pass.

This paper highlights the tensions between optimizing for computational efficiency and designing for better network performance. For example, one of the questions we answer is how do we best integrate time-domain information. Should many adjacent frames in the video sequence be used to caption a salient scene although it would entail more complex and often times more computations? Should we use an explicit or a learned neural controller to sample salient frames in time where the explicit controller is an efficient vector-vector operation and the neural controller is an Long Short-Term Memory (LSTM)?

1.1. Contributions

We have three main contributions:

1. Because the main computational bottleneck is in the LRNN, we design a temporal sampler using CNN features to sample only key frames in the video sequence and to minimize the number of language RNN forward passes. We enjoy over 1) $> 100\times$ decrease in number of saved frames, 2) $7.4\times$ improvement in speed-up, and 3) $8\times$ increase in energy efficiency.

2. at each temporally-salient frames, we caption the frame and perform one backward pass to compute an attention map that attends to the region of interest described by the generated captions.

3. we provide low-level optimization using matrix factorization on the LSTM matrices in the LRNN with retraining. This allows a $1.2\times$ speedup. In order to improve CNN efficiency, we explore FPGA implementation on the CNN.

2. Related Work

Recent state-of-the-art image caption generation contains CNN-RNN architectures [8]. And there have been major advances to extend to video captioning [5]. However as demonstrated in [9], video frames contain highly redundant temporal information which leads to modest improvements in video classification over single-frame baselines.

A fundamental challenge is how and when to sample frames. Sampling too fast yields little new information and sampling too slow may lose too much. Furthermore, this sampling ought to be efficient on an embedded system. Many paper discuss methods to find key frames in a video sequence using clustering techniques for frames and using motion [12, 19]. In similar spirit, our method can be interpreted as online clustering and is performed using the network without external hand-crafted motion energy models. Similarly, to extract key spatial regions of interest in the frame, we choose to reuse the same network for captioning. This is commonly performed using saliency maps in conventional CNN architectures [17]. However, we examine the end-to-end interaction between x^t and the caption generation outputs of the RNN, which is inspired by [21] but which our method requires only 1 backward pass.

Low-level computational optimization such as matrix factorization (tensorification) on FC layers (usually the final layer) have been recently proposed [14, 16] to reduce model size and runtime. We incorporate this into the LSTM matrices of the LRNN with retraining. Implementation of this system on the TX1 is inspired by the work of [4] that deployed Neural Talk [7] on the Nvidia TK1 embedded system. There are limited works on deep learning on FPGAs. However, both works [22, 6] create dedicated convolutional cores designed with optimal compute bandwidth versus memory bandwith in mind.

3. Methods

Throughout this work and shown in Fig. 1, we use VGG-16 [18] for the CNN with the last fully-connected layer (FC) removed and a one-hidden layer LSTM with a sequence length of 16 for the LRNN. We build upon the framework of NeuralTalk 2 [7] and use Torch [2] for all experiments and Xilinx Vivado for FPGA development of the CNN. For computational efficiency, we use an explicit controller to determine which frames are salient-in-time but also explore an LSTM alternative. We primarily focus on the optimization of the LRNN; this is because without any optimization the average speed of the LRNN forward pass is $44\times$ that of the CNN forward pass. All computation times in this paper are computed using the lua os.clock call. As shown in Table 1, the LRNN forward pass is the computational bottleneck. Therefore our goal is to minimize the number of LRNN forward passes on frames that are temporally highly redundant in context information using the CNN features.

	CNN	LRNN	All
Forward-Pass Time (ms)	0.7	310	390

Table 1: Average execution time (milliseconds) in CNN-LRNN forward passes during frame-to-caption inference. The input to the CNN is a single center-cropped 224×224 frame.

We evaluate our methods on the MPII Movie Description Dataset [15] to train the CNN-LRNN network for image caption generation. This dataset contains short clips from movies and captions. Since our network operates on individual frames, we save only the frame that appears in the center of each clip and label it with the corresponding caption. During training and factorization with retraining, $\approx$ 1700 frames and captions from Harry Potter 6 and 7 movies are used with $\approx$ 100 frames set aside for validation. We use the last Harry Potter movie for test with $\approx$ 1000 frames. Both the original and modified captions were used. To evaluate our methods on real-life images, we use the pretrained model from NeuralTalk2 with finetuning (retraining) using MSCOCO2014 [11].

3.1. Temporal sampling and spatial attention

In order to reduce the number of saved frames in a real-time video acquistion, we present a computationally cheap method in Algorithm 1 that senses for changes in context and samples the frames at these key frames. Fig. 2 illustrates the CNN features for a 2500-frame video clip with redundant contextual information. We use these CNN features to determine how to sample.

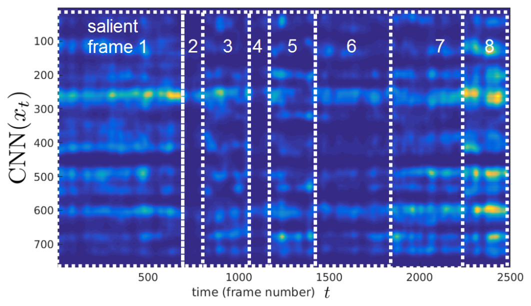


Figure 2: CNN features over time in a video sequence with highly redundant temporal information.

Algorithm 1 Temporal and spatial sampling

```
1:  procedure SAMPLE($x$)
2:  given $\gamma$, the detection threshold, and $K$, the LSTM maximum time-span.
3:      repeat
4:        1. Let $y^t \leftarrow$CNN($x^t$).
5:        2. break if $t = T$.
6:        3. if $||y_{\text{cluster}} - y^t||_v \geq \gamma$ or LSTM($y$) $\geq \gamma$
7:            $y_{\text{cluster}} \leftarrow$ mean($y^t, y_{\text{cluster}}$).
8:            $z^t \leftarrow$ LRNN($y^t$).
9:            $\Omega =$ argmax $z^t$ (across word indices).
10:           $x_s^t \leftarrow x^t \odot$ backCNN(backLRNN($\mathbb{1}(\Omega)$))
11: return $x_s^t$.
```

The choice of norm v can range from 2 to ∞. This hyperparameter is chosen based on how much the user emphasizes a few large changes or many small changes in CNN features over time. In our demonstration, we set $v = 5$. We also compare the performance of an explicit controller to that of a learned LSTM controller.

The single-layer LSTM controller is trained to minimize the regression MSE loss $||z^t_{\text{salient}} - \text{LSTM}(\text{CNN}(x^t))||_2$. The labels z^t_{salient} contains 1's at salient time-stamps and 0's everywhere else, and a gaussian filter is applied to smoothen sharp $1-0$ transitions.

3.2. Computational and hardware optimizations

We propose two low-level optimizations: 1) factoring the LSTM weight matrices $W_{ih} \in \mathbb{R}^{m\times p}, W_{hh} \in \mathbb{R}^{m\times p}, W_{out} \in \mathbb{R}^{n\times p}$ with retraining in the 1-layer LRNN language model, and 2) accelerating the CNN in an FPGA.

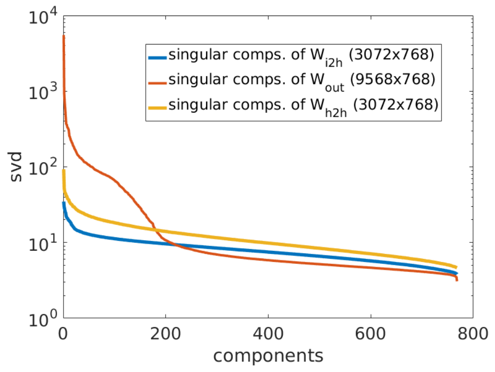


Figure 3: The distribution of singular values in log-scale of W_{ih}, W_{hh}, W_{out} on the pretrained LRNN model.

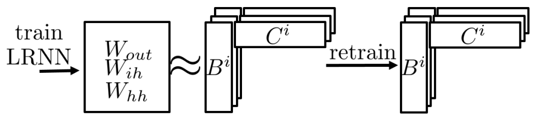


Figure 4: Matrix factorization is used on the LSTM matrices to reduce the number of multiply-and-accumulate operations.

We motivate factorization on the LSTM matrices in order to accelerate the speed-up of the LRNN forward pass. In Fig. 3, the singular values of W_{ih}, W_{hh}, W_{out} are heavily distributed around 0. This provides an opportunity for low-rank matrix factorization to reduce the number of multiply-and-add operations in the forward pass as shown in Fig. 4. The algorithm for factorization and retraining is illustrated in Algorithm 2.

Without factorization, one forward pass through the LRNN requires $2np + mp = 12.1 \times 10^6$ multiply-and-add operations to compute one element (word index) of the output vector z_i^t, which holds the unnormalized log probabilities of the words in the dictionary and the end token. Although this number is comparable to the forward pass through the CNN (dominated by the 4096×4096 FC layers), the LRNN performs these multiply-and-add operations a maximum of 16 times.

With factorization, one LRNN forward pass requires only $2(nr + rp) + mr + rp$ multiply-and-add operations, where r is the rank of all the matrices. If $r \approx 150$, we need only 2.7×10^6, which would entail a $4.5\times$ reduction over the unfactored case. To ensure that factorization does not affect LRNN performance, we retrain over the entire network with the factored matrices included in the new architecture and initial model.

Algorithm 2 Training with factored LSTM matrices

```
1: procedure TRAIN-CNN-LRNN(factor with rank $r$)
2:     $B^{(1)}, C^{(1)} = \underset{B^{(1)} \in \mathbb{R}^{n \times r}, C^{(1)}}{\text{argmin}} ||W_{out} - B^{(1)}C^{(1)}||_F$
3:     $B^{(2)}, C^{(2)} = \underset{B^{(2)} \in \mathbb{R}^{m \times r}, C^{(2)}}{\text{argmin}} ||W_{ih} - B^{(2)}C^{(2)}||_F$
4:     $B^{(3)}, C^{(3)} = \underset{B^{(3)} \in \mathbb{R}^{m \times r}, C^{(3)}}{\text{argmin}} ||W_{hh} - B^{(3)}C^{(3)}||_F$
5:     initialize CNN with VGG-16 conv. weights.
6:     initialize LRNN with $B^{(i)}, C^{(i)}$ for $i = 1, 2, 3$.
7:     train CNN-LRNN jointly using Adam solver.
```

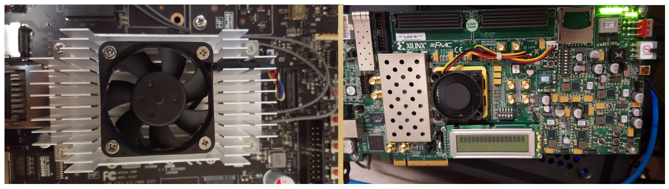


Figure 5: Embedded Jetson TX1 (left) and VC707 FPGA (right) development boards.

We evaluate computational and hardware performance using the Jetson TX1 and VC707 FPGA development kits as pictured in Fig. 5 using Vivado [3]. Because of limited time and constraints on the maximum configurable logic block (CLB) utilization of the FPGA, we only deploy the CNN on the VC707.

4. Experiments and Analysis

In order to evaluate our methods as described in Section 3, we first train our CNN-LRNN model on MPII using Adam solver [10] with model and validation checkpoints every 3000 iterations with training and validation mini-batch size of 50 image-sentence pairs. All captions for the training set are converted to lower-case and all words that occur less than 2 times are removed. We also clip gradients elementwise for the LRNN training and apply dropout in the CNN layers. The softmax output of the LRNN computes the unnormalized log-probabilities of words across the dictionary, and the log-loss criterion over the words is the objective function. Convergence in validation loss is achieved after a couple thousand iterations for the unfactored LRNN.

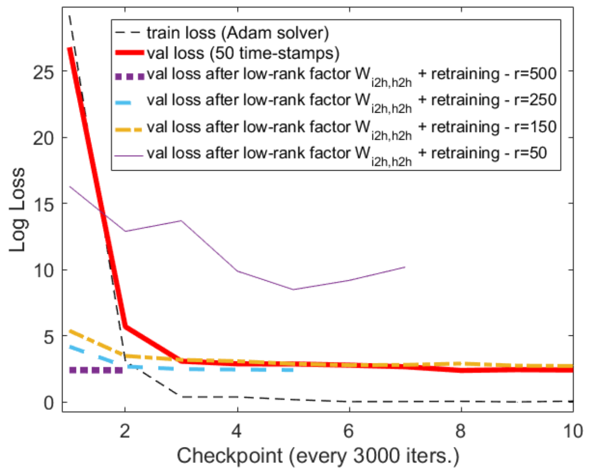


Figure 6: Training and validation loss curves on MPII. Two sets of curves are shown: 1) unfactored LRNN and 2) factored LRNN.

The training and validation log-loss is shown in Fig. 6 without and with matrix factorization on all three LRNN LSTM matrices (W_{ih}, W_{hh}, W_{out}) for varying ranks r. The final test loss is 2.9 for unfactored training and 2.9 with factorization and retraining.

4.1. Temporal sampling and spatial attention from generated captions

We first compare the sampling method as presented in Algorithm 1 with the baseline method that uses the raw pixels on a short sequence of frames labelled event 1 and event 2 in Fig. 7. Frames 1 to 165 are marked Event 1 and contains little motion and no changes in context while frame 165 is the time-stamp for when a change in context has occurred. In this scenario, a new object enters the camera's field of view.

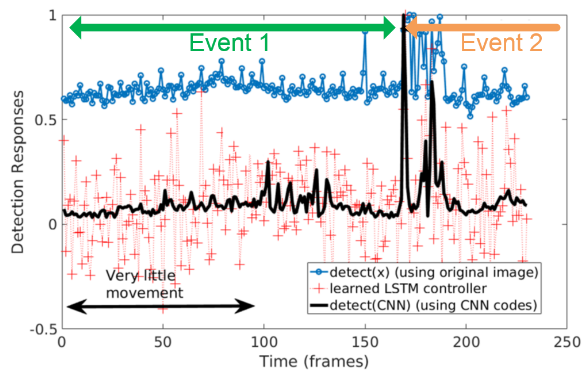


Figure 7: Detection detect$(y^t) = ||y_{\text{cluster}} - y^t||_5$ as used in Algorithm 1 for a short clip. The signal to noise ratio using the features is ≈ 8 dB higher than using the raw pixels.

We plot the normalized detect$(y^t) = ||y_{\text{cluster}} - y^t||_p$ where y^t is the CNN feature vector of a frame at time t. We compare this to the baseline where y^t is simply the raw pixels of the frame at time t. The detection responses indicate a much higher signal to noise ratio using the features a s opposed to the raw pixels. We also plot the result of using an LSTM controller; while at frame 165 the learned function produces a high detect signal, it is still noisy at other redundant frames such as in event 1 (high false positive).

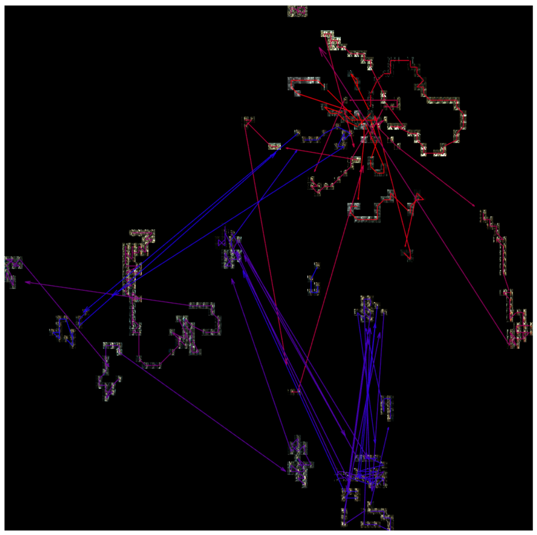


Figure 8: t-SNE embedding visualization of a 30-second clip in the Harry Potter test set with time flow. The colored arrows represent the time flow from red to blue. Note the clusters of many redundant frames.

We apply our proposed Algorithm 1 on both the Harry Potter test set (2-hr. length movie) and a 15-minute video demonstration across Stanford Campus. We compare our

Method	False Pos.	False Neg.
Uniform subsample[1]	0.51	0.34 ± 0.1
Using raw pixels[1]	0.32	0.28 ± 0.1
Proposed explicit[1]	**0.03**	$\mathbf{0.09 \pm 0.1}$
Proposed learned[1]	0.03	0.4 ± 0.1
Uniform subsample[2]	0.63	0.25
Using raw pixels[2]	0.58	0.21
Proposed explicit[2]	**0.00**	**0.11**
Proposed learned[2]	0.15	0.31

Table 2: Sampling using Algorithm 1 on Harry Potter test set[1] of length 2-hours and user-video[2] of length 15-minutes. The total number of sampled frames is set approximately the same for each video sequence.

method (proposed explicit) with naive uniform subsampling and Algorithm 1 using raw pixels instead of the features. Proposed explicit uses detect$(y^t) = ||y_{\text{cluster}} - y^t||_5$ and raw pixels uses detect$(x^t) = ||x_{\text{cluster}} - x^t||_2$ It is empirically observed that l_2 norm produces the lowest false positive and negative rates for raw pixels.

The false positive and false negative rates are shown in Table 2. Each frame in the two sequences are human-labelled. A false positive event is an event where the algorithm incorrectly samples a frame that contains both the same location and objects in the frame of view. A false negative event is an event where the algorithm misses an otherwise temporally-salient frame. To account for some ambiguously salient events in the 2-hour movie, we add 0.1 margins to the false negative for the 2-hour movie.

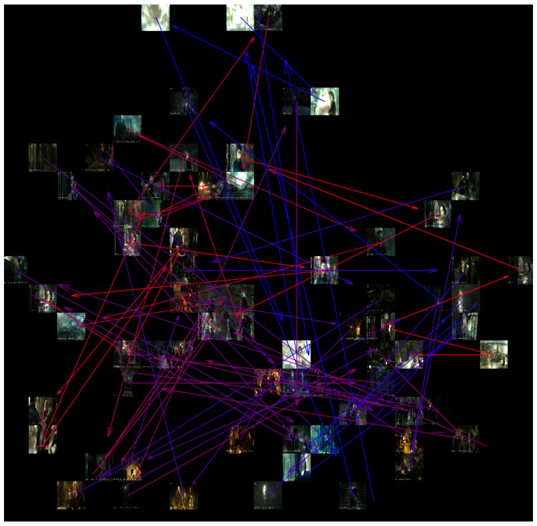


Figure 9: Proposed method's t-SNE embedding of the entire 2-hour movie in the test set.

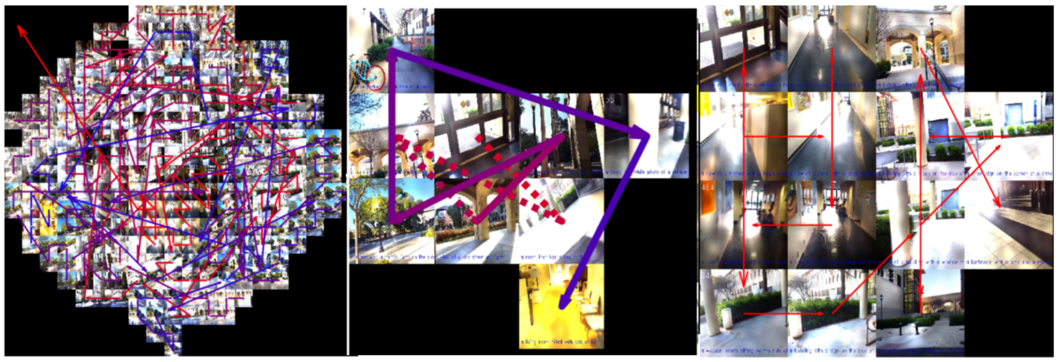


Figure 10: t-SNE embeddings of subsampled ($4\times$), proposed, naive sampling using raw pixels. The generated diary: A view of a window in a house. A street scene with a bike on the side of the street. A building with a clock on the side of it. A view of a building with a door open. A person standing in front of a building. A street with a traffic light on the side of it. A bike parked on the side of a road. A bike parked in front of a building.

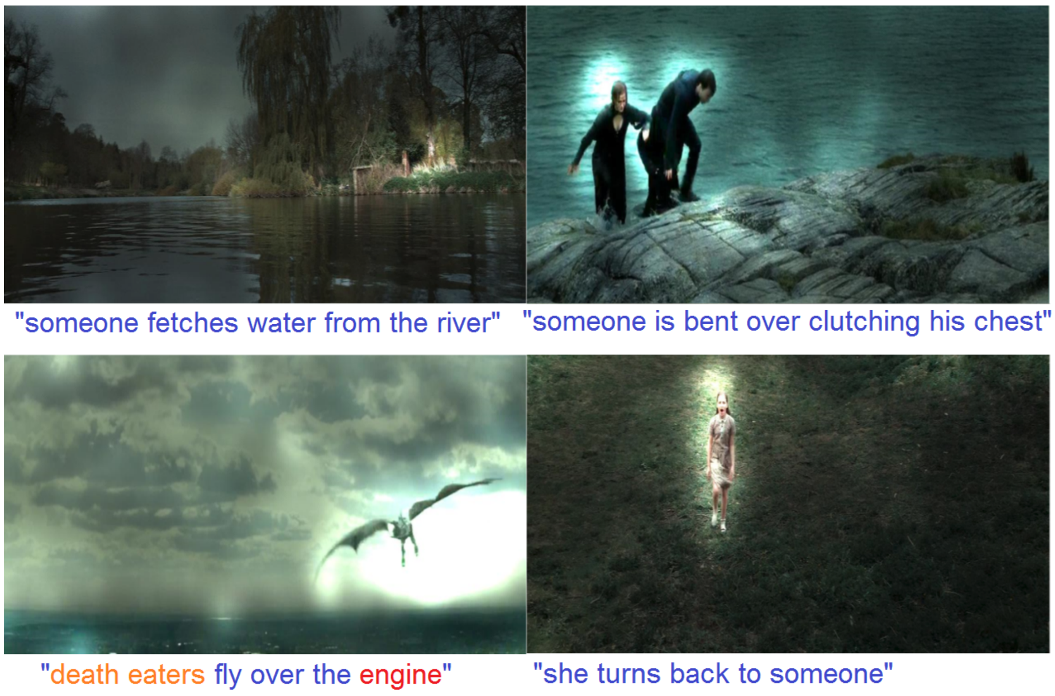

"someone fetches water from the river"
"someone is bent over clutching his chest"
"death eaters fly over the engine"
"she turns back to someone"

Figure 11: Sampled frames that the LRNN captions (on test set) and generates the spatial-attention maps based on its generated captions.

Using proposed explicit, we see a more than $10\times$ and $3\times$ decrease in false positive and negative rates respectively compared to using raw pixels. Unfortunately, the learned controller does not work as well in false negative rates. This is most likely due to the higher occurrence of false positives during the training procedure. However, even if we penalize false negatives higher than false positives in the optimization, the performance does not improve.

We present qualitative t-SNE embeddings [20] that illustrates the effect of subsampling. Fig. 8 illustrates the t-SNE embedding using CNN features on a 30-second clip of the test set. Using our proposed explicit method, we are able to compactify all 100 salient frames from the 2-hour length movie in one t-SNE visualization as shown in Fig. 9. To qualitatively compare the sampling methods, we show t-SNE on the user-video, which is a smaller video sequence.

Method	Samples	Avg. Time (ms)	Power (W)	Energy (J/frame)
Conventional	$\sim$ 1000	390	9.7 to10.5	3.8
Proposed on TX1	9	52.6	9.0 to 10.5	0.47
Proposed factor-TX1	9	43.8 (r =150)	9.0 to 10.5	0.47
Projected FPGA (CNN)	9	33.3	< 1	< 0.03

Table 3: Performance results in hardware. The FPGA implementation is still an on-going development.

In Fig. 9, we visualize the t-SNE with subsampled $4\times$ in order to fit to screen, our sampling, and sampling using raw pixels. We can clearly see that using raw pixels yields frames that are redundant (high false positive) and missing novel frames (high false negative).

Following the Algorithm 1, for each of the sampled frames we feed the CNN features into the LRNN for caption generation and backpropagate the derivatives from the generated words. This allows us to understand the end-to-end interaction between the spatial regions of x^t and the caption generation outputs. Fig. 11 illustrates 4 representable sampled frames that are captioned and element-wise multiplied with the attention map. Before multiplication with x^t, we apply a small gaussian filter on the map to smoothen minor high-frequency artifacts.

These attention maps highlight the regions of interest corresponding to the captions but can also be used to hard-segment the objects. Results of a simple hard thresholding on the attention maps are shown in the Appendix (section 6). Performance of these saliency maps are not included in this paper; these attention maps are used only for qualitative purposes for the final generated diary.

4.2. Computational and hardware optimization

Since the LRNN is the computational bottleneck as shown in Table 1, we perform matrix factorization on the LSTM matrices as detailed in Algorithm 3.2. After factorization for various ranks r, we retrain the entire CNN-LRNN network. The hyperparameter r determines the sizes of the factored matrices $B^{(i)}, C^{(i)}$; the larger the r the better the approximation of the unfactored matrix in the Frobenius sense but the greater the number of matrix operations. As we show in Fig. 6, retraining with an $50 < r$ allows convergence of validation loss to near unfactored levels. However, for $r = 50$, convergence is dramatically impeded.

Using the best model checkpoints in the factorization with retraining process, we measure the LRNN forward pass runtime in Fig. 12. The test loss (log-loss) is also shown; the test loss for this experiment is a direct measure of the dissimilarity of the generated words and the ground truth captions. With even a large $r \approx 500$, we can boost the runtime by about 5% without any loss in caption generation performance. This can be pushed to $r \approx 250$ where the best test loss exceeds the best test loss of the unfactored case with a runtime close to $1.2\times$ the unfactored case.

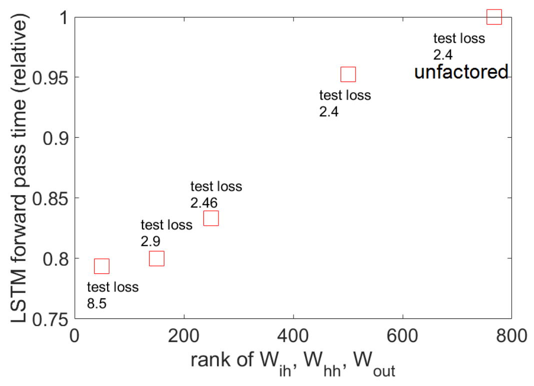


Figure 12: Test loss after retraining for varying matrix factorization rank r.

We summarize the performance of our embedded system on the TX1 using Algorithm 1 and 2 in Table 3. Here, we clearly see the $100\times$ reduction in the number of sampled frames, $7.4\times$ speed-up in runtime ($8.9\times$ with factorization and retraining), and increase in energy efficiency by $8\times$.

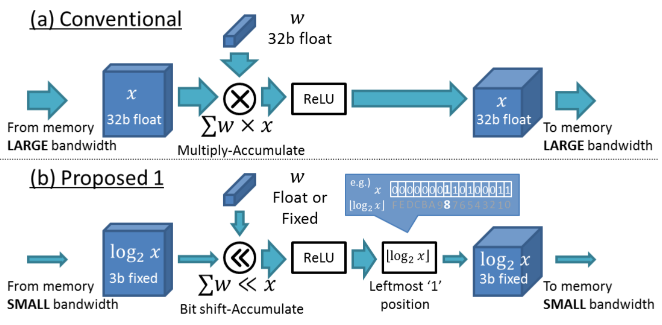
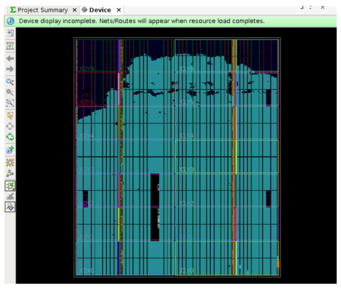
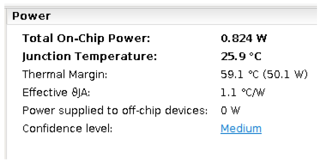


Figure 13: Proposed multiplier-less algorithm to evaluate the dot products in the CNN forward-pass as shown in [13].



Device display incomplete. Nets/Routes will appear when resource load completes.



Power	
Total On-Chip Power:	**0.824 W**
Junction Temperature:	**25.9 °C**
Thermal Margin:	59.1 °C (50.1 W)
Effective ϑJA:	1.1 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Figure 14: Architectural floorplan on FPGA and the power estimated using Xilinx Vivado environment.

By subsampling the forward pass through the LRNN, the computational bottleneck is shifted to the CNN forward pass. In order to increase energy efficiency and runtime of the CNN pass, we explore FPGA acceleration, which contains dedicated convolutional logic with distributed memory for the weights and activations.

We modify the network that we published recently [13] and deploy it on the FPGA. The network is a VGG-like network without any FC layers. The algorithm is shown in Fig. 13; this algorithm contains no dedicated multipliers. The floorplan illustrating the CLB utilization and power are shown in Fig. 14. Functionality of the FPGA-CNN has been verified with the LRNN in simulation. However, end-to-end functionality of the FPGA-CNN and LRNN is still an on-going work. Using an FPGA for the CNN allows us to improve frame rates and potentially reduce power by an order of magnitude over the embedded TX1.

5. Conclusion

In this paper, we design a real-time video captioning embedded system that takes an ultra-long video stream (15 minutes to 2 hours) and samples salient frames in time, generates captions on the objects of these frames, and generates attention maps that attend to these captions. We have three contributions. First, we design a temporal sampler using CNN features to sample key frames and to minimize the number of LRNN forward passes. This yields a $100\times$ reduction in the number of sampled frames, $7.4\times$ speed-up in runtime ($8.9\times$ with factorization and retraining), and increase in energy efficiency by $8\times$ on the embedded TX1. Second, for each temporally-salient frame, we caption the frame and compute an attention map that attends to the region of interest from the captions. Third, we apply matrix factorization on the LRNN matrices with retraining allowing $1.2\times$ speedup and an increase in energy efficiency. We finally explore FPGA acceleration for the CNN.

Immediate future work consists of moving the system that is currently deployed in the TX1 development kit to just the TX1 module. This allows for ultra-portability. We also consider moving both the LRNN and CNN to the FPGA. Finally, results using the LSTM-based temporal sampler have shown some promise and could be used for end-to-end training. Finally, we hope that this project will be part of a larger robotics system very soon.

6. Appendix - Hard Attention

We do not explore segmentation or localization in this paper. However, we show preliminary qualitative hard attention maps generated from captions in Fig. 15.

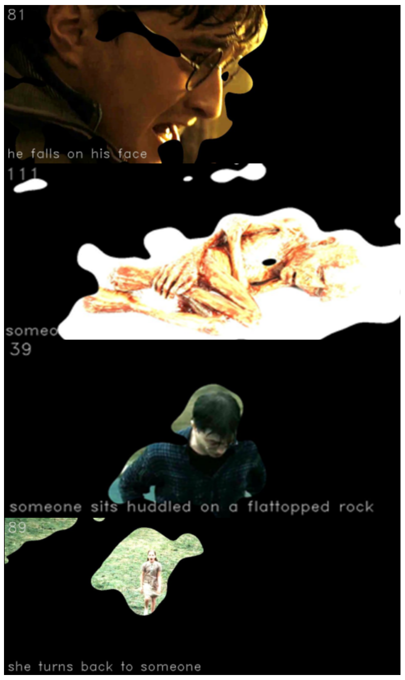

81
he falls on his face
111
someo
39
someone sits huddled on a flattopped rock
89
she turns back to someone

Figure 15: Thresholded spatial attention maps where the spatial regions are set to 0 if backCNN(backLRNN($\mathbb{1}(\Omega)$)) is smaller than a predetermined threshold. These maps and captions are generated from the unthresholded image on the test set.

7. Acknowledgements

We thank the entire teaching staff for their great suggestions and teaching! We thank Daisuke Miyashita!

References

[1] Nvidia Jetson TX1. `http://www.nvidia.com/object/jetson-tx1-module.html`, 2015.

[2] Torch. `https://github.com/torch/torch`, 2016.

[3] Vivado design suite. `http://www.xilinx.com/products/design-tools/vivado.html`, 2016.

[4] S. Das and S. Han. Neuraltalk on embedded system and gpu-accelerated rnn. *CS224D Project*, 2015.

[5] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *arXiv preprint arXiv:1411.4389*, 2014.

[6] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 257–260. IEEE, 2010.

[7] A. Karpathy. Neuraltalk 2. `https://github.com/karpathy/neuraltalk2`, 2015.

[8] A. Karpathy and F. Li. Deep visual-semantic alignments for generating image descriptions. *arXiv preprint arXiv:1412.2306*, 2014.

[9] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *arXiv preprint arXiv:1405.0312*, 2014.

[12] T. Liu, H.-J. Zhang, and F. Qi. A novel video keyframe-extraction algorithm based on perceived motion energy model. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(10):1006–1013, Oct 2003.

[13] D. Miyashita, E. H. Lee, and B. Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[14] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov. Tensorizing neural networks. *CoRR*, abs/1509.06569, 2015.

[15] A. Rohrbach, M. Rohrbach, N. Tandon, and B. Schiele. A dataset for movie description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[16] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659, May 2013.

[17] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] S. Uchihashi and J. Foote. Summarizing video using a shot importance measure and a frame-packing algorithm. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 6, pages 3041–3044 vol.6, Mar 1999.

[20] L. van der Maaten. Barnes-hut-sne. *CoRR*, abs/1301.3342, 2013.

[21] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[22] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 161–170, New York, NY, USA, 2015. ACM.