Image Captioning with Recurrent Neural Networks

Kaiyuan Wang
k5wang@ucsd.edu

Merve Kilic
mkilic@ucsd.edu

Utkarsh Jain
utjain@ucsd.edu

Marialena Sfyraki
msfyraki@ucsd.edu

Abstract

Image Captioning is a prominent problem in computer vision and natural language processing. The main goal of this problem is to generate a caption that closely describes an image. In this regard, we explore an encoder-decoder architecture that encodes an image using a Convolutional Neural Network (CNN) and uses it to generate captions with a Recurrent Neural Network (RNN). In this project, we choose the pre-trained ResNet50 model as our encoder and explore the power of Long Short-Term Memory (LSTM) networks to generate captions. We experiment with Vanilla RNNs and also try passing the encoded image at every time step in our LSTM decoder to observe their effect on the quality of the generated captions. We use the COCO dataset and after hyperparameter tuning achieve a performance[1] of **(1.67, 66.66, 7.76)** for the baseline model, **(1.68, 64.53, 6.50)** for the Vanilla RNN model, and **(1.60, 67.37, 7.58)** for the model with the image encoding given at each timestep. From our experiments, we find that the model with image encoding passed at every time step gives the best performance, followed by the LSTM model and then the Vanilla RNN model. Furthermore, we notice that stochastic sampling is superior to deterministic sampling while generating captions. Lastly, we notice how different hyperparameters effect the overall performance and we discuss our observations later.

1 Introduction

Image Captioning task is an important problem in artificial intelligence which employs knowledge from computer vision and natural language processing to generate captions that closely describe an image. Obtaining such automatically generated descriptions for images has a myriad of applications such as helping visually impaired people to understand the world and their surroundings, early childhood education, image retrieval, etc.

In this project, we employ a neural network model that takes in an image as input and generates a caption for it. The whole model contains an encoder that encodes an image as a feature representation and a decoder that uses the feature representation to generate captions. The encoder uses a pre-trained ResNet50 model [3], for generating a feature representation for a given image. The decoder learns a dense feature embedding for the words in the vocabulary and uses it, along with the encoded image, to generate captions. The pre-trained encoder is frozen during training. To train the decoder, we use teacher-forcing that uses the teaching signal from the training dataset at the current time step, $target(t)$, as input in the next time step $x(t+1) = target(t)$, rather than the output $y(t)$ generated by the network. This is found to make the learning faster and lead to quicker convergence.

In our experiments, we use a subset of the Common Objects in Context (COCO) dataset [5] which is a large-scale object detection, segmentation, key-point detection, and captioning dataset with over

[1] performance reported in form of (test loss, bleu1, bleu4)

328K images. However, we only use 9,000 training images and around 2,000 validation images respectively for training and validating the model. Figure 1 shows that our model learns to generate meaningful captions that closely describe the images. Our model can be further improved by using more powerful image encoders and decoders motivated by recent advancements.

Figure 1: Examples of images, actual captions, and generated captions

2 Related Work

The COCO dataset was introduced in [5] with the goal of placing the question of object recognition in the context of the broader question of scene understanding. Among other annotations, every image is associated with 5 captions. The COCO dataset now serves as a benchmark dataset in the deep learning research community. Other similar datasets include Flickr8K, Flickr30K, and IAPR TC-12.

More recently [8] used the COCO dataset as the benchmark for their novel and effective approach of using attention mechanism over image features to generate better captions. In their work, the authors used a similar encoder-decoder approach as ours, however, instead of encoding an image as a one-dimensional feature vector, they extract multiple features from a lower convolutional layer, each of which is a representation corresponding to a part of the image. Then, using an attention module, they generate a context vector, which is a dynamic representation of the relevant part of the image, at every timestep. Using the context vector, along with the LSTM state and previous word, they compute the output word probability for the next word. Based on their experiments, they achieved state-of-the-art performance on several metrics on various benchmarking datasets, including COCO dataset. Based on a similar idea of attention mechanism [1] proposed a novel combined bottom-up and top-down visual attention mechanism that enables attention to be calculated more naturally at the level of objects and other salient regions. The bottom-up mechanism in their model works by proposing several "regions of interest" in the image, each associated with a feature vector. The top-down mechanism then weighs each feature during caption generation, using the existing partial output sequence as context, and generates a combined feature that is used as input to the decoder.

ResNet was proposed in [3] where the authors presented a residual learning framework to ease the training of networks that are substantially deep. They experiment with residual networks of various depths and achieved state-of-the-art results on the ImageNet dataset and COCO object detection dataset. In our project, we use a pre-trained ResNet50 model (depth of 50 layers) to encode the images.

Vanilla RNN (or Elman RNN) was introduced in [2] where the author brought forth a novel neural network architecture that incorporated the concept of "time" in neural networks. This was achieved by using recurrent connections between hidden units and memory units, and trainable parameters from the memory units to the hidden units. LSTM network was proposed in [4] to address the issue of vanishing gradients in Vanilla RNNs. It achieves this by using several gates which control the gradient flow and effectively deal with the vanishing gradient problem. We will be using Vanilla RNNs and LSTMs in our project as a decoder to generate captions.

3 Methods

In this section, we explain the architecture of the encoder and decoder module of our model (please see Figure 2), the decoding strategies we use during evaluation, how our model learns the word embedding, and the training procedure.

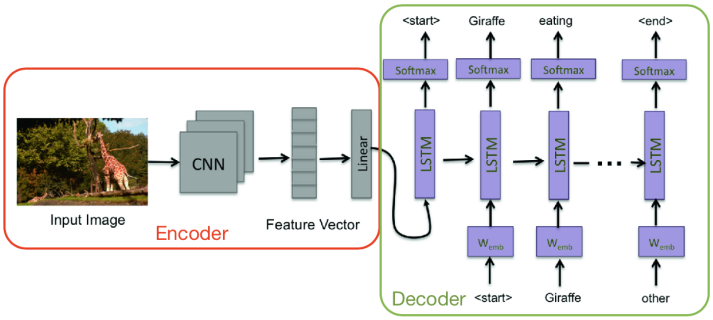


Figure 2: Encoder-Decoder Architecture for Image Captioning. [7]

3.1 Encoder

For our experiments, we use a pre-trained ResNet50 network [3]. However, we remove its last fully-connected layer and add a trainable linear layer to map from the penultimate hidden layer dimension to a desired image embedding dimension. This encoded representation is then passed to the decoder where it is used to generate the captions.

The dimension of the image embedding is set to the word embedding dimension in the LSTM and Vanilla RNN decoder because we pass the encoded image as an input in the first timestep. However, this constraint is relaxed in Architecture 2 because the image is concatenated with the words and then passed to the LSTM at each time step.

3.2 Vanilla RNN Decoder

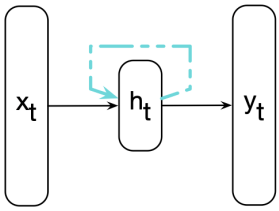


Figure 3: Vanilla Recurrent Neural Network

In this section, we consider the most basic class of RNNs called Vanilla Networks (or, Elman RNNs). Fig 3 illustrates the working behind a Vanilla RNN. The input vector at time t, x_t, is multiplied to a weight matrix and passed through a non-linear function to generate h_t, which is also called the hidden state at time t. This hidden state is then used to compute the corresponding output, y_t. As you can see in the figure, the recurrent link (represented with a dashed line) combines the hidden state from the preceding point in time, i.e h_{t-1} and input x_t to generate h_t. Using h_{t-1} this way serves as a memory, or context, that encodes previous computations and guides future decisions. Given x_t and h_{t-1}, h_t is computed as follows:

$$h_t = tanh(Uh_{t-1} + Wx_t) \tag{1}$$

where, $U \in R^{d_h \times d_h}$, $W \in R^{d_h, \times d_{in}}$, $h_t, h_{t-1} \in R^{d_h}$, $x_t \in R^{d_{in}}$, and $h_0 = \{0\}^{d_h}$ (d_{in} and d_h are the input and hidden sizes respectively).

In our decoder, we use a two-layered Vanilla RNN to form a stacked RNN, with the second RNN taking in the outputs of the first RNN and generating the final hidden states. These hidden states are then passed through a linear layer with a softmax activation function to generate a probability distribution over the set of words in the vocabulary. A word is then sampled from this distribution using the strategy discussed in section 3.5. The sampled word is embedded into a lower dimensional space and passed as input to the stacked RNN in the next time step. The encoded image from the encoder is used as the input in the first time step and the hidden states are initialized to zeros.

3.3 LSTM Decoder

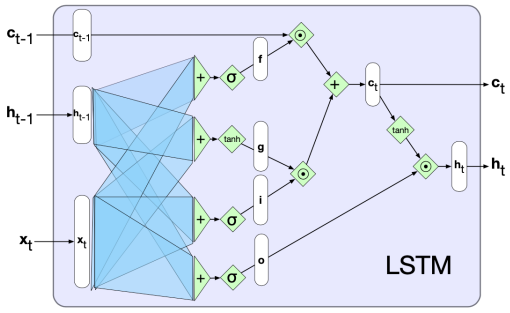


Figure 4: A single Long Short-Term Memory Network Unit

An LSTM processes information in the same way a Vanilla RNN does. However, to maintain relevant information for a longer period of time, it employs the concept of a 'memory cell'. It also uses different gates, such as forget gate, update gate, and output gate, that allow more control over the gradient flow and effectively deals with the vanishing gradient problem. Figure 4 gives an accurate visualization of how all the computations occur.

In our decoder, we use a two-layered LSTM to form a stacked RNN, with the second LSTM taking in the outputs of the first LSTM and generating the final hidden states. These hidden states are then passed through a linear layer with a softmax activation function to generate a probability distribution over the set of words in the vocabulary. A word is then sampled from this distribution using the strategy discussed in section 3.5. The sampled word is embedded into a lower dimensional space and passed as input to the stacked LSTM in the next time step. The encoded image from the encoder is used as the input in the first time step. The hidden states and cell states are initialized to zeros.

3.4 Architecture 2

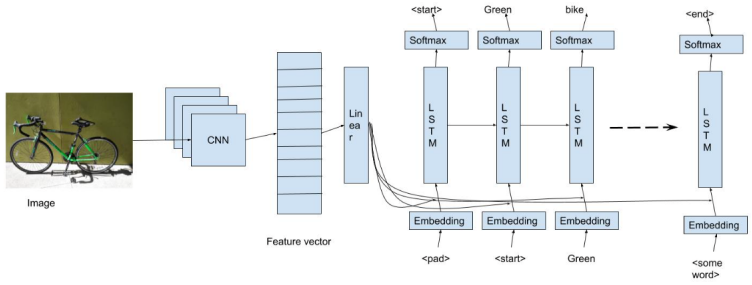


Figure 5: Architecture 2 for Image Captioning

In the previous models, we were using the encoded image as the input for the first time step. In architecture 2, however, the encoded image is concatenated with the word embedding for "$< pad >$" and passed to the stacked LSTM as input in the first timestep. The output is used to generate a word using the strategy discussed in section 3.5. The sampled word is embedded into a lower dimensional space, concatenated with the encoded image, and passed as input to the stacked LSTM in the next

time step. Since we are no longer using the encoded image as a sole input to the LSTM, and rather using it along other words, we no longer have to make sure that the image encoding size matches the word embedding size. As usual, the hidden states and cell states are initialized to zeros. Please refer to Figure 5 for easy visualization of this process.

3.5 Generating Strategy

The output from every time step from our decoder basically returns a probability distribution over the entire set of vocabulary, and we use this distribution to select a word for the next time step. There are two strategies for choosing the next word:

- **Deterministic Sampling:** In this strategy, we choose the most probable word and use it in the next time step. Although this sounds intuitive, in practice it has certain downsides. First, for a particular image, it will always generate the same caption which is not optimal. Second, it is seen this strategy is prone to grammatical errors and doesn't perform very well when generating long sequences.
- **Stochastic Sampling:** In this strategy we use the probability distribution over the vocabulary to sample a word. This results in a more novel and realistic caption generation. However, in order to gain more control over the randomness of the generated caption, we augment the softmax function with temperature scaling. The scaling is done by dividing the logit vector (the raw scores generated by the final linear layer) by a temperature term T, followed by the softmax function.

$$y_j = \frac{exp(o_j/T)}{\sum_n exp(o_n/T)} \tag{2}$$

where o_j is the output from last layer, n is the size of the vocabulary, and T is the temperature.

The temperature controls how stochastic the sampling is: as the temperature approaches 0, the distribution becomes nearly deterministic and collapses to a point mass, and as it goes to infinity, the distribution becomes completely uniform, thus increasing the uncertainty.

3.6 Training Procedure

To train our model we use the concept of teacher-forcing which uses the teaching signal from the training dataset at the current time step, $target(t)$, as input in the next time step $x(t+1) = target(t)$, rather than the output $y(t)$ generated by the network. In simpler terms, while training, instead of using the words predicted by the decoder in the next timestep, we use the ground-truth label as the input. However, during inference, since we do not have the ground-truth labels, we let the network run on its own by using the network's prediction to obtain the next input word. We also limit the size of the generated caption to 20 during inference.

We keep the pre-trained ResNet50 encoder frozen during training and only learn the parameters associated with the linear layer after the encoder, the embedding layer in the decoder, the RNN, and the final linear layer when backpropagating over the error. We use CrossEntropyLoss over the raw scores produced by the final linear layer to compute the error and gradients, and employ the Adam optimizer for gradient descent. We then conduct experiments to adjust the hyperparameters of our models.

Experiments

We run four kinds of experiments in this project. In the first one, we try tuning the learning rate and batch size of the baseline model while keeping the rest of the hyperparameters at default configuration (word embedding dimension: 300, two hidden layers of 512 LSTM units). We try the values 1e-3, 5e-4, and 1e-4 for the learning rate, and 32, 64, and 128 for the batch size. This corresponds to section 3.1 in the assignment handout.

Second, for the Vanilla RNN model, we keep the same set of hyperparameters as the baseline model, but tune over the learning rate and batch size. This corresponds to section 3.2.1 in the assignment

handout. For the learning rate, we try the values 1e-3, 5e-4, and 1e-4, while the values we use for the batch size are 32, 64, and 128.

For the third experiment, we try tuning the embedding and hidden sizes of the baseline model (using the optimal learning rate and batch size found from the first experiment) to improve its performance. This corresponds to section 3.2.2 in the assignment handout. We try values of 200, 300, and 400 for the embedding size and values of 256, 512, and 1024 for the hidden dimension.

For the fourth experiment, we tune Architecture 2 over the image embedding dimension, word embedding dimension, and learning rate. This corresponds to section 3.3 in the assignment handout. We try values of 200, 300, and 400 for the image embedding dimension, 300, 400, and 600 for the word embedding dimension, and 1e-4, and 5e-4 for the learning rate.

Please check section 4 for the optimal setting of hyperparameters and its evaluation on the test dataset for each experiment.

3.7 Learning Word Embeddings

We use PyTorch's nn.Embedding layer to map every word in the vocabulary to a low-dimensional embedding space that is initialized randomly drawn from a uniform distribution. During training, the gradients from the loss function backpropagate to this layer, and the associated parameters are adjusted to optimize the model's performance.

4 Results

Hyperparameter Settings and Evaluation

The best setting of hyperparameters for each model and the corresponding BLEU-1 scores, BLEU-4 scores, and cross entropy loss value computed on the test set are reported in Table 1. Please check appendix A for hyperparameter tuning experiments.

Model	Image Dim	Word Dim	Hidden Dim	lr	bs	Bleu1	Bleu4	Test Loss
Baseline	300		512	1e-4	32	66.65182	7.54300	1.69074
Vanilla RNN	300		512	1e-4	64	64.53590	6.50821	1.68191
Tuned Baseline	400		512	1e-4	32	66.66819	7.76160	1.67754
Architecture 2	200	400	512	1e-4	64	67.37103	7.57806	1.60253

Table 1: Best hyperparameters for each model and test evaluation

Training and Validation Loss Curves

The plots of training and validation loss vs number of epochs for each model are illustrated in Figure 6.

Examples of Generated Captions

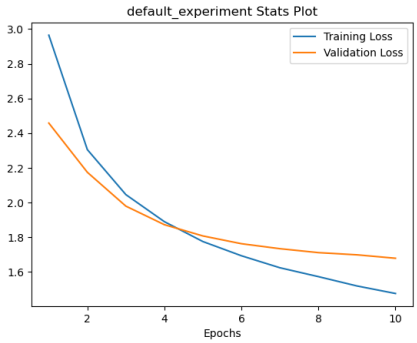
Please check appendix B for examples of captions generated by the tuned baseline model for different temperature settings, appendix C for the tuned RNN model, and appendix D for tuned Architecture 2.

5 Discussion

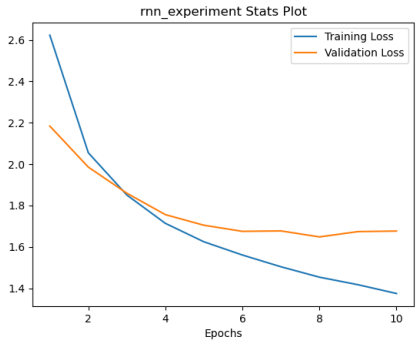
Here we discuss the results and explain some of the observations we made while running the experiments.

Model Comparison

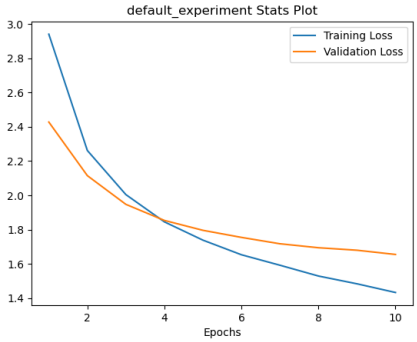
Based on our experiments and performance on test dataset, we conclude that Architecture 2 works the best as compared to the other two in all evaluation metrics. We believe this is because by



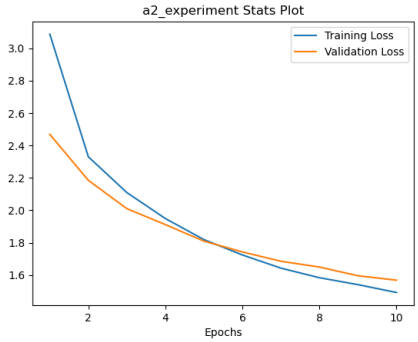
(a) Baseline Model



(b) Vanilla RNN Model



(c) Tuned Baseline Model



(d) Architecture 2 Model

Figure 6: Train/Validation curve using the best hyperparameters for each model

feeding the image at every time step to the model, we are essentially letting the model focus less on "remembering" the image and focus more on learning representations that lead to better caption quality. On the other hand, since we are feeding the image only at the first time step, we make it harder for the other two models to retrieve the image embedding information at later time steps. This is because after every time step, the other two models lose some of the encoded image information. This is equivalent to the "information bottleneck" problem in the Seq2Seq Machine Translation networks [6]. From Figure 6, we also notice that the validation loss curve closely follows the training loss curve for Architecture 2 and shows no sign of overfitting until the 10th epoch. Because of the limited computational power, we could not train it for longer, but we expect Architecture 2 to perform even better with more training.

We also notice that the baseline model with the LSTM decoder performs better than the Vanilla RNN decoder model. This might be true because of the superiority of LSTM networks in dealing with the vanishing gradients problem and the overall flow of information and gradient in the network. Vanilla RNNs suffer from the vanishing gradients problem which prevents them from learning effective representations as compared to LSTMs. This can also be seen by the difference in the test loss and the BLEU scores of the two models. While experimenting with the two models, we also notice that LSTMs do a better job of generating longer captions than the Vanilla RNN model. This might be true because of the design of LSTMs that helps them remember meaningful information for longer periods of time which in turn helps them generate more "continuous" captions. Furthermore, based on the training/validation loss curve in Figure 6, we can see that the Vanilla RNN model starts overfitting after the 8th epoch, whereas the baseline model continues learning more. We notice that the baseline takes longer to overfit and produces better results.

Furthermore, while tuning the baseline model, we basically run two kinds of experiments. In the first kind of experiment, we tune over learning rate and batch size, and in the second one, we tune over word embedding dimension and hidden dimension. Based on our experiments, and Table 1,

we see that tuning word embedding dimension and hidden dimension produced better results and we notice a good drop in the test loss. Not just that, but we also noticed an improved BLEU1 score and a sharp increase in the BLEU4 score. Also, we observe that increasing the word embedding dimension generated better results, which might be because it increases the overall representational capacity of the model. Based on the training/validation loss curve in Figure 6, we also notice that the tuned baseline model achieves an overall lower training and validation loss by the 10th epoch and still shows no sign of overfitting.

Effect of Hyperparameters

While tuning the word embedding dimension and hidden dimension, we found that increasing the hidden dimension did not help as much as increasing the word embedding dimension. This is possible because to generate better captions, the model basically should learn better word representations. So, in a way the word embedding dimension kind of acts like a bottleneck here. Making it too small would prevent the model to learn effective word representations. However, making the word embedding dimension too high also resulted in lower performance. The same trend can be seen with the image embedding size as well.

We also observe that using a learning rate of 1e-4 produced much better results than a learning rate of 5e-4. This felt strange at first because both values are pretty low and in the previous assignments we usually found 1e-3 to produce optimal results. This might be because RNNs work very differently than the feed-forward networks we have seen in past assignments. Basically, in an RNN, since we use the same weights over all the time steps, the gradients keep on accumulating on the same sets of weights again and again. Using a lower learning rate helps the model take smaller steps over those "higher" gradient values.

Caption Generation Strategy

Through our experiments, we found that stochastic sampling is superior to deterministic sampling. Stochastic generation offers better diversity, i.e. the ability to choose from a wider range of potential words at each time step. Stochastic generation also offers better robustness against inferior outputs because there's always a variety of word choices at any given time step. Whereas, with the deterministic strategy we always generate the same caption which is not optimal. Also, we observed that the best captions generated by the deterministic approach had lower BLEU scores as compared to those generated by the stochastic sampling approach.

While experimenting with different values of temperature in stochastic sampling, we found that lower values of temperature usually resulted in better BLEU1 and BLEU4 performance and also produced meaningful captions. However, reducing it to very low values resulted in almost deterministic caption generation. Meanwhile, increasing it to high values usually resulted in lower BLUE1 and BLEU4 performance. This makes sense if we see how stochastic sampling works. Using a really low-temperature value just makes the probability distribution concentrate on one word, which then leads to deterministic caption generation. Whereas, a high-temperature value makes the probability distribution more and more uniform and the model ends up picking words that might not describe the image or make sense in the sentence in general.

Scope for Improvement

We believe that using pre-trained word embeddings, like GloVe vectors, and fine-tuning them on our task would result in better performance. This is because GloVe vectors and other commonly available word embeddings are trained on huge datasets and do a better job of learning word representations.

Furthermore, during our experimentation, we saw that although our models did a good job of generating captions, they still sometimes missed out on a few details in the image. If you see appendix B you would see a lot of such examples. For instance, the caption generated for the left most image in Figure 7 completely misses out the bottle of wine in the image. We believe that using an encoder that can pay attention to the details and generate a more effective encoding could help our decoder to generate better captions.

6 Team Contribution

- **Kaiyuan Wang**: Worked on implementing the train() and val() functions in the experiment.py file. Also wrote Visualization.ipynb notebook which helps visualize generated captions and the associated images. Tuned the baseline model on word embedding dimension and hidden dimension. Wrote the first three sections of the report and collaborated on other sections.
- **Utkarsh Jain**: Worked on implementing and integrating early stopping mechanism during training and loading the best model during test time. Worked on implementing the resnet encoder and the baseline decoder in the model_factory.py file. Ran experiments on baseline model. Wrote the "Methods" section of the report and worked with others on finishing the report.
- **Merve Kilic**: Worked on implementing the test() function in the experiment.py file. Implemented the Vanilla RNN decoder architecture and integrated it with the Decoder() class in model_factory.py file. Ran experiments on the Vanilla RNN model. Worked on the "Methods" section and "Results" section of the report.
- **Marialena Sfyraki**: Worked on understanding how BLEU scores work and implemented the compute_bleu_score() helper function in the experment.py file. Implemented Architecture 2 and integrated it with the Decoder() class in model_factory.py file, and wrote the generate_captions() function in the decoder. Ran experiments on Architecture 2. Worked on the "Methods" and "Discussion" sections of the report.

References

[1] Peter Anderson et al. "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[2] Jeffrey L. Elman. "Finding structure in time". In: *Cognitive Science* 14.2 (1990), pp. 179–211. ISSN: 0364-0213. DOI: https://doi.org/10.1016/0364-0213(90)90002-E. URL: https://www.sciencedirect.com/science/article/pii/036402139090002E.

[3] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[4] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[5] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215. URL: http://arxiv.org/abs/1409.3215.

[7] Anurag Tripathi, Siddharth Srivastava, and Ravi Kothari. "Deep Neural Network Based Image Captioning". In: *Big Data Analytics*. Ed. by Anirban Mondal et al. Cham: Springer International Publishing, 2018, pp. 335–347. ISBN: 978-3-030-04780-1.

[8] Kelvin Xu et al. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. 2015. DOI: 10.48550/ARXIV.1502.03044. URL: https://arxiv.org/abs/1502.03044.

Appendix A: Hyperparameter Tuning

Model	Image Dim	Word Dim	Hidden Dim	lr	bs	Bleu1	Bleu4
Baseline	300		512	1e-3	64	61.17416	6.32826
				5e-4	64	62.53496	6.99643
				1e-4	64	65.65333	6.85300
				1e-4	32	66.65182	7.54300
				1e-4	128	62.51799	5.25601
Vanilla RNN	300		512	1e-3	64	53.11193	2.71601
				5e-4	64	62.24884	6.39600
				1e-4	64	64.53590	6.50821
				5e-4	32	61.51282	5.90524
				1e-4	32	63.55750	6.90025
				5e-4	128	61.95502	6.20733
				1e-4	128	62.22749	5.68200
LSTM	200		512	1e-4	64	65.07194	6.66888
	200		1024	5e-4	64	63.65530	7.35463
	300		256	1e-4	32	65.12327	6.32260
	300		1024	1e-4	32	66.64962	8.03710
	400		256	5e-4	64	64.68249	7.49722
	400		512	1e-4	32	66.66819	7.76160
	400		1024	1e-4	32	66.38546	8.11547
LSTM arch2	200	300	512	1e-4	64	66.43983	7.21069
	200	400		1e-4	64	67.37103	7.57806
	400	400		1e-4	64	66.42556	7.39015
	400	400		5e-5	64	65.00900	5.91048
	600	600		1e-4	64	54.98837	1.57368

Table 2: Hyperparameter Tuning

Appendix B: Captions with Baseline Decoder

Captions generated with hyperparameter setting of: word embedding dimension = 400; hidden dimension = 512; learning rate = 1e-4; batch size = 32; temperature = 0.1 unless specified differently.

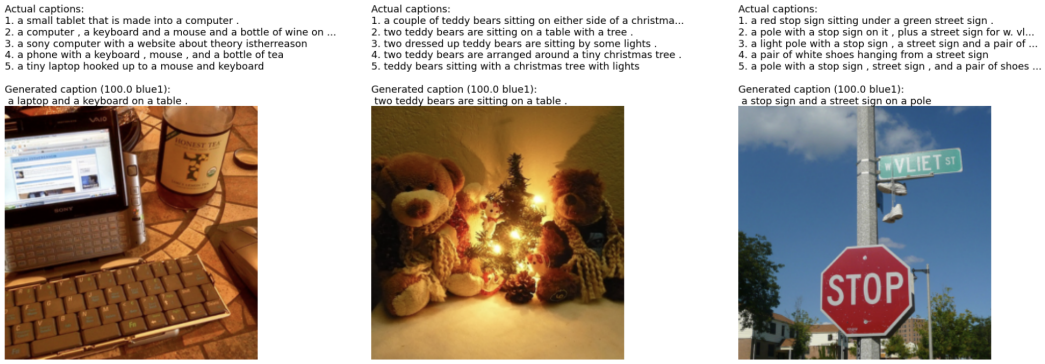


Figure 7: Good captions produced by baseline model

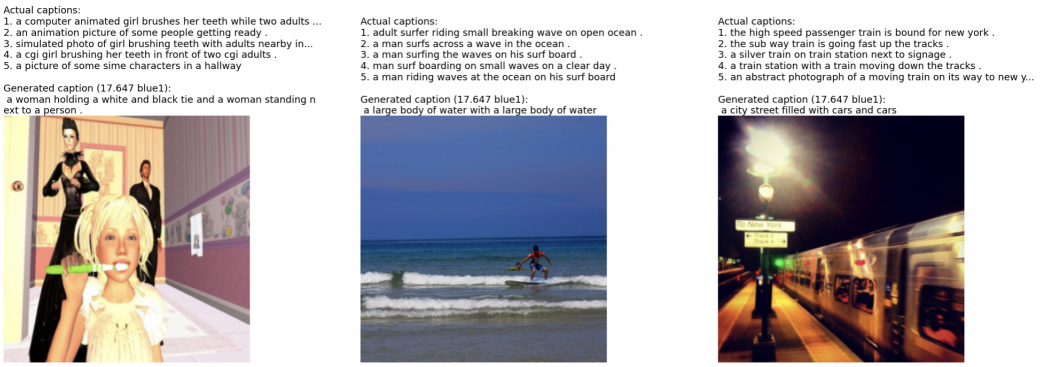


Figure 8: Bad captions produced by baseline model

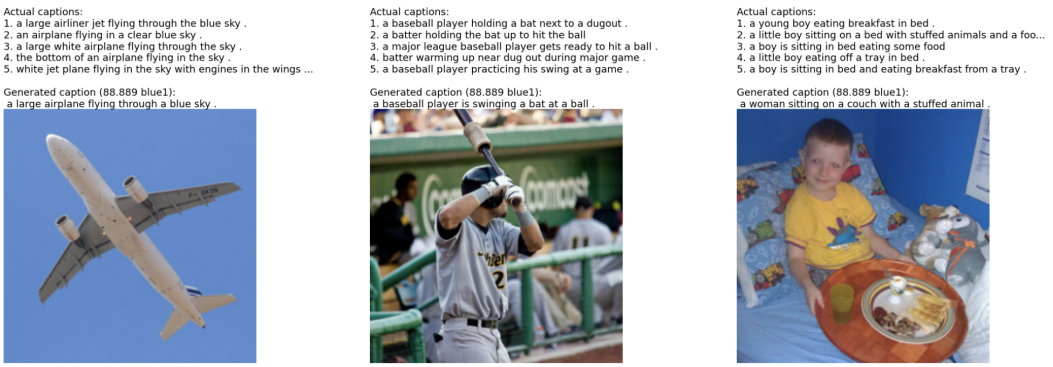


Figure 9: Deterministic captions produced by baseline model

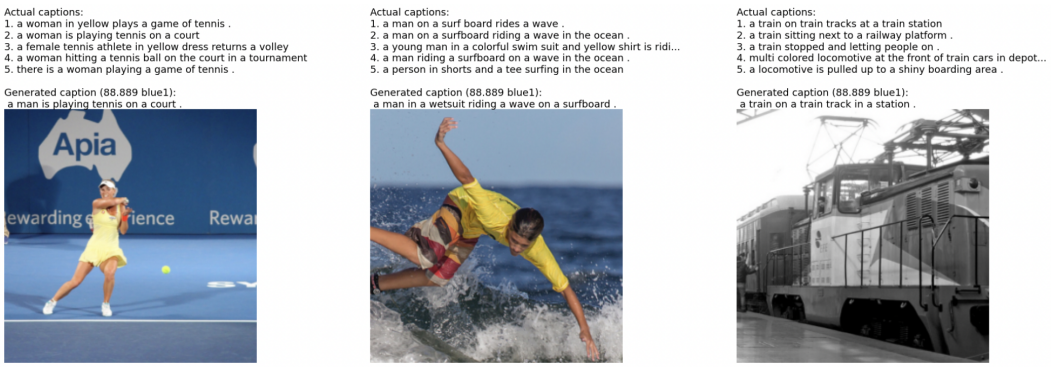


Figure 10: Low-temperature captions (temperature = 0.001) produced by baseline model

Figure 11: High-temperature captions (temperature = 5) produced by baseline model

Appendix C: Captions with Vanilla RNN Decoder

Captions generated with hyperparameter setting of: word embedding dimension = 300; hidden dimension = 512; learning rate = 1e-4; batch size = 64; temperature = 0.1 unless specified differently.

Figure 12: Good captions produced by Vanilla RNN model

Figure 13: Bad captions produced by Vanilla RNN model

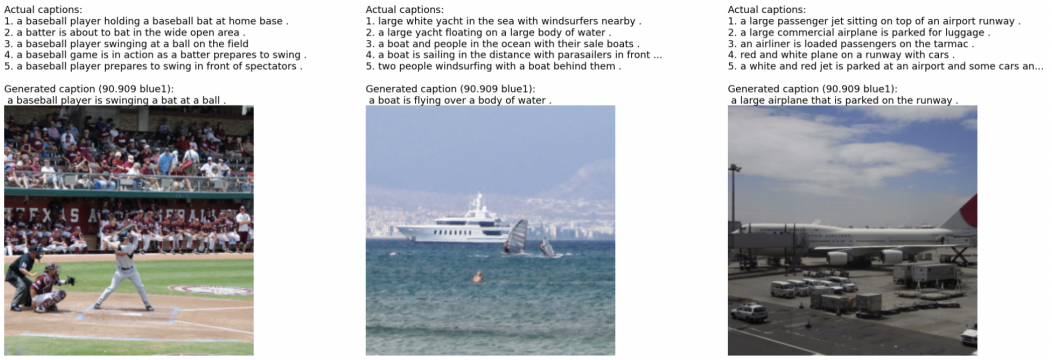


Figure 14: Deterministic captions produced by Vanilla RNN model

Figure 15: Low-temperature captions (temperature = 0.001) produced by Vanilla RNN model

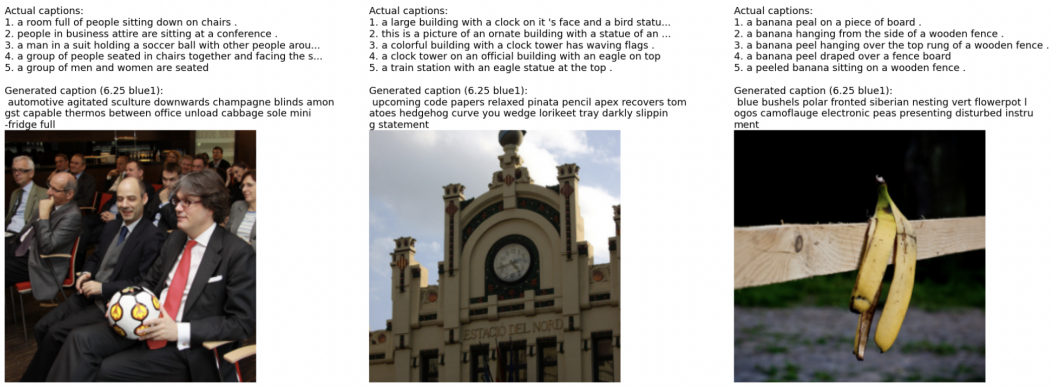


Figure 16: High-temperature captions (temperature = 5) produced by Vanilla RNN model

Appendix D: Captions with Architecture 2

Captions generated with hyperparameter setting of: image embedding size = 200, word embedding dimension = 300; hidden dimension = 512; learning rate = 1e-4; batch size = 64; temperature = 0.1 unless specified differently.

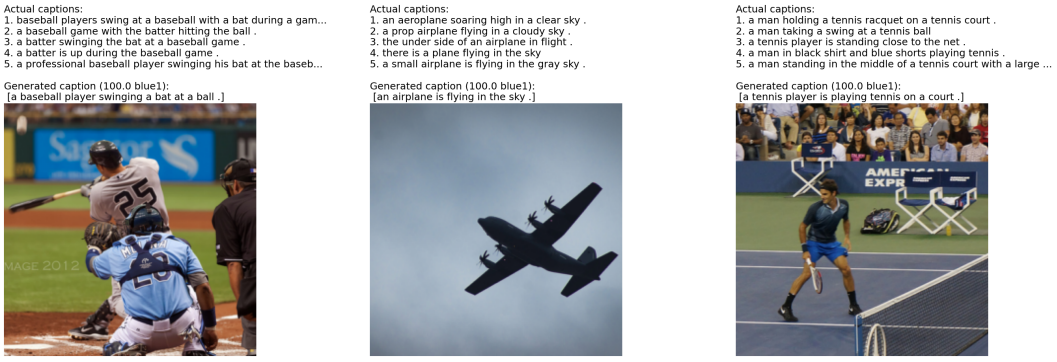


Figure 17: Good captions produced by Architecture 2

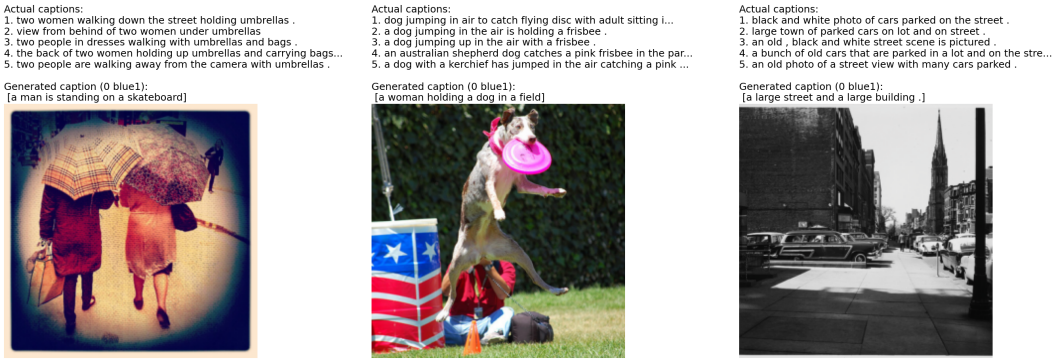


Figure 18: Bad captions produced by Architecture 2

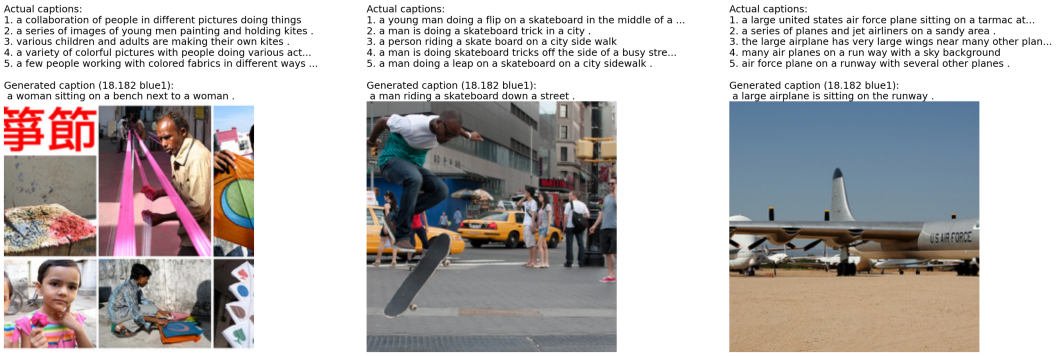


Figure 19: Deterministic captions produced by Architecture 2

Actual captions:
1. two people standing next to each other on top of the snow .
2. two people standing in the snow with skies on the ground and...
3. two people standing in the snow with ski equipment .
4. two people with hats standing next to ski gear
5. skiers with equipment looking at snow covered rise in distan...

Generated caption (11.111 blue1):
 [a man is skiing down a snowy slope .]

Actual captions:
1. people at the baggage claim area of an airport .
2. three people standing before airport counters below airport ...
3. people standing at counters of booths being served
4. the baggage delivery section of an air port .
5. patrons are going to the shops of an airport .

Generated caption (11.111 blue1):
 [a group of people sitting on a table with a large clock]

Actual captions:
1. a collaboration of people in different pictures doing things
2. a series of images of young men painting and holding kites .
3. various children and adults are making their own kites .
4. a variety of colorful pictures with people doing various act...
5. a few people working with colored fabrics in different ways ...

Generated caption (11.111 blue1):
 [a woman sitting on a bench next to a woman .]

Figure 20: Low-temperature captions (temperature = 0.001) produced by Architecture 2

Actual captions:
1. a photographer setting up his equipment for a photo shoot .
2. a person holding up a purple suit case .
3. a man wearing jeans holds up a purple suitcase .
4. a man holding a suit case above his head in front of a poste...
5. a man standing in a room holding a piece of purple luggage o...

Generated caption (0 blue1):
 blooming nearest enjoying parasails hood unfolded \n \n \n \
n \n pancake together roast 2009 \n \n \n \n \n recliner
 bareback shower point marking vanity privacy linked condit
ions

Actual captions:
1. a city bus is shown with a bike in front .
2. a street view , left , shows a curb with a yellow fire hydra...
3. a bicycle is attached to the front of this bus .
4. a city bus drives down a street
5. a bus that has a bike on the front of it .

Generated caption (0 blue1):
 cents breads container slaw duke lookin \n \n \n \n lande
d features chillin flowers \n white highway edible

Actual captions:
1. one person holds the back of a surfboard while another perso...
2. one man in the water holds a surfboard while another man bal...
3. a girl standing on one leg on a surfboard while a man holds ...
4. a person stands on one leg on one end of a surfboard while a...
5. a man holding a surfboard with a woman jumping off of it .

Generated caption (0 blue1):
 clap sucking sorted avocado mingle

Figure 21: High-temperature captions (temperature = 5) produced by Architecture 2