INM706: Deep Learning for Sequence Analysis

Image Captioning

Coursework
2021-22

Anthony Hoang
Priyanka Velagala

Table of Contents

1.0 Introduction (Anthony Hoang)

1.1 Problem Domain

Image captioning is a complex task that lies in the intersection of the field of computer vision and natural language processing. It involves generating syntactically and semantically meaningful text descriptions based on the content of an image. This task has impactful applications from the ability to automate image indexing to adding enhanced capabilities to virtual assistants used by visually impaired persons.

The current state of the art proposes several models all heavily focused on deep learning methods combined with deep reinforcement and adversarial training [1] . Some of the more recent developments in the field include the UpDown architecture , which combines the Faster-R-CNN with an attention LSTM where the bounding boxes from the CNN allow attention to be focused on the more salient regions of the image. Another is meta-learning, where models that use reinforcement learning have the dual learning task of optimizing both the reward and the maximum likelihood estimate (MLE). However, currently, the best-performing model by far combines an Image-Captioning GAN with an ensemble of CNNs and RNNs (see Appendix A).

For the task of image captioning, a survey conducted by Chohan et al. [2] in 2019 showed the best performing architectures use an encode-decode mechanism where CNN-LSTMs significantly outperform CNN-RNNs. As such, with consideration of architectures in scope for this course, we have chosen to implement the former for our coursework.

1.2 Dataset

We chose Flickr30k, a benchmark dataset to train and test our model. In using a benchmark dataset, this facilitates easy comparison of our model performance with industry standards. The dataset itself contains 31,000 images, each with 5 descriptive captions. Due to time and compute restrictions, we've chosen to work with a subset of 10,000 images using all 5 reference captions for training. As there are no standard data splits for this dataset, a 7:2:1 train-validation-test split was used resulting in a total of 35,000 training, 10,000 validation and 5,000 testing instances.

1.3 Code References

While the source code was written entirely from scratch, a full list of references that inspired our implementation and guided us in following best practices can be found in Appendix B of this document. Instructions for dataset retrieval and running test cases on our final model can also be found in the same section.

2.0 Data Preprocessing

For the encoder, as we use a pretrained Inception v3, the model expects that the input images are at least 299x299 and normalized by a mean = [0.485, 0.456, 0.406] and standard deviation = [0.229, 0.224, 0.225] such that the input is scaled to a value between 0 and 1[3]. Hence, this transform was applied to all images in the dataset. Additionally for training, we applied further transforms such that the model generalizes better on previously unseen instances.

For captions, as we didn't intend to use pre-trained word embeddings, we have a special class that builds a dictionary from the corpus and indexes each unique word. Each caption is then numericalized by using the corresponding word to index mapping which can be interpreted by the decoder. To ensure there are adequate instances of a word in our dataset to meaningfully correlate the word to a feature from the image, we discard any word that appears fewer than 5 times in the dataset in an effort to produce high quality captions.

3.0 Model Components

The proposed architecture was inspired by advances made in machine translation and was first introduced in the Neural Image Caption (NIC) model by Vinyals et al (2014) [4]. In this section, we describe the functionality of the main components in our implementation.

3.1 Encoder

The purpose of the encoder is to extract meaningful features from the input image. For this, we use a convolutional neural network (CNN) pre-trained on an image classification task. For our implementation, we chose to work with the inception v3 model that was trained on 1.3 million images from the ImageNet dataset achieving a 78.1% accuracy [5].

By training on a dataset that large, we assume the model has a good representation of objects in the world and can extract defining features with a high degree of accuracy. This allows us to save on the computationally intensive task of training the CNN from scratch and apply transfer learning where the features learned from the ImageNet dataset are used to extract key features from the images in our chosen dataset.

Our choice of CNN was motivated in working with a computationally limited platform. Inception models are able to achieve SOTA performance however tend to have lower computational cost in comparison to deeper networks such as ResNets [6]. This was of key importance in our baseline model where in addition to feature extraction, the model required fine-tuning of the last fully-connected layer based on the embedding size parameter.

3.2 Decoder

For our decoder, as specified in the original paper, we use a recurrent neural network (RNN) for sequence modeling. RNNs perform particularly well in tasks such as image captioning due to their ability to correlate information extracted from prior inputs to influence both inputs and outputs at a given time step[7]. These networks have a "memory" component which allows them to remember and account for the previous stream and position of inputs giving them strong predictive capabilities. Figure 3.2 below shows the feedback mechanism (of hidden states) in the form of an unrolled RNN.

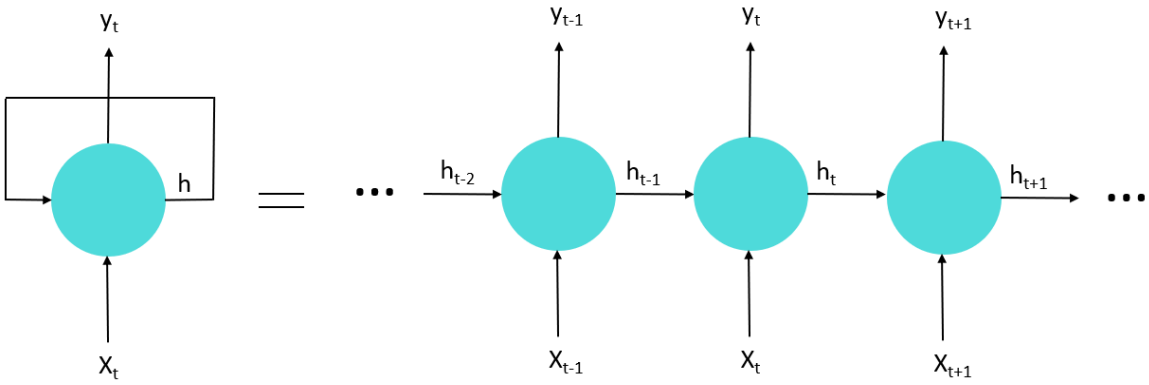


Figure 3.2 : Representation of an unrolled RNN

A key characteristic of this network is that they share the same weight parameters within each layer of the network. Therefore, it uses a slight variation of the back propagation algorithm traditionally used in feedforward networks where instead of back propagation after each forward pass, the error calculated is summed at each time step. As a consequence, on update, this can cause either exploding or vanishing gradients resulting in an unstable model or a model with no learning, respectively[7].

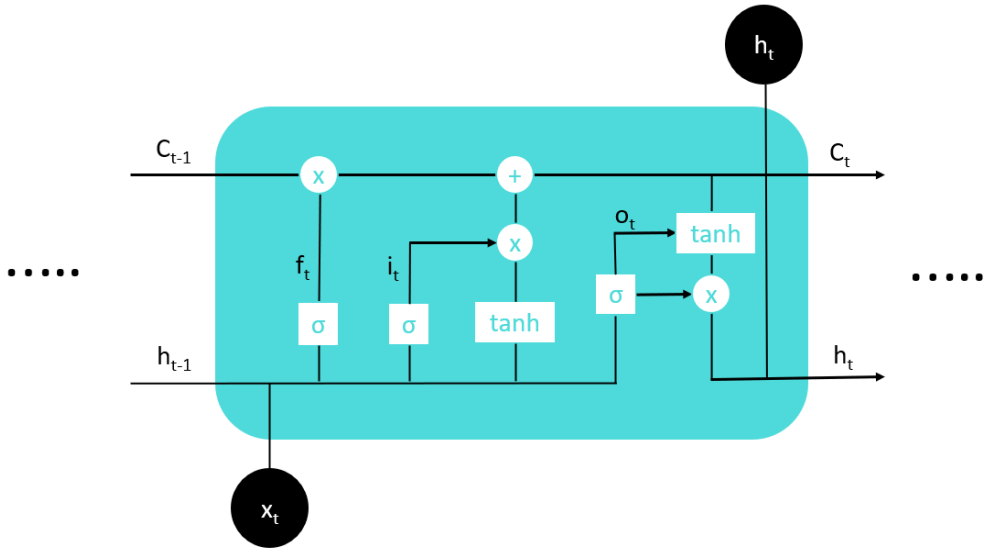


Figure 3.3: Long Short Term Memory (LSTM) Cell

We motivate our choice of using LSTM RNNs as a result of this problem, as well as traditional RNNs' inability to handle long term dependencies[7]. The LSTM introduces a memory cell that is used to store information for longer periods of time. There are three key components that regulate what is stored in this cell (see Figure 3.3):

1. Forget gate (f_t) - decides which information to omit from the previous time step
2. Input gate (i_t) - decides which information passes through based on relevance in the current timestep
3. Output gate (o_t) - decides how much of the cell state impacts the output

These three gates let the memory cell decide when to remember or ignore inputs in the hidden states.

3.3 Attention mechanism

An enhancement that was made to our baseline model to improve the quality of captions generated was through the addition of an attention mechanism. The use of attention in image captioning was introduced by Xu et al. (2015) and proposed two types of attention - soft attention and hard attention. Fundamentally, both mechanisms require focusing on certain regions of the image when generating the next word in the sequence. How they differ is where hard attention requires only a single patch of the image be attended to, soft attention allows attending to more than one region at a given time step[8]. We chose to implement soft attention as it is easier to train and allows us to use standard back-propagation methods such as gradient descent[9].

Soft Attention considers all regions of interest and context as its input, and outputs the regions most relevant at that time step. The weights of the attention module are learnable parameters which are adjusted based on features maps from the encoder and the context thus far. To ensure the regions output by the attention module are only those most relevant to the context, summing the two allows the context most similar to the region to be amplified. Passing these intermediary values through a softmax layer produces a probability distribution of the attention weights. The dot product of these weights and the regions highlight the context which is then passed as input to the LSTM. Figure 3.3 below shows a high-level overview of each step.

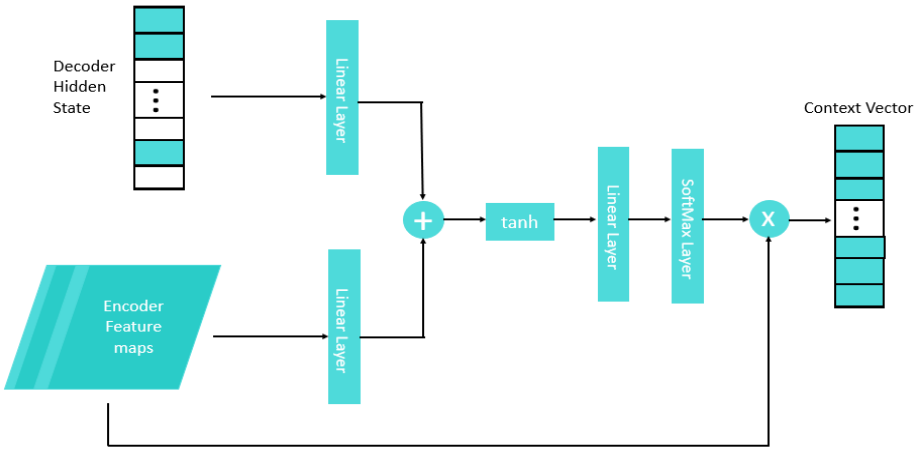


Figure 3.4: Soft Attention Module

4.0 Model Architecture

In this section, we provide a high-level overview of our model architecture(s). We will also justify our choice of hyperparameters, loss functions and optimizers used and cover other pertinent aspects of model training.

4.1 Models

4.1.1 Baseline Model

The setup for our baseline model is as seen below in Figure 4.1 . This closely follows the NIC model specified in the paper by Vinyals et al (2014). For our encoder, we use a pre-trained inception v3 for feature extraction. As the model was originally designed for an image classification problem, and weights pretrained on the ImageNet dataset, the output layer has a fixed size of 1000 (i.e. number of classes in ImageNet). For our model, in order to be able to customize the number of features extracted, the size of the output layer needed to be changed. This meant the last layer needed to be fine tuned such that the weights of the final layer are recalculated for the specified output size.

As for the decoder, we pass in both the features extracted by the encoder as well as the numericalized captions as inputs. The first component of the decoder is an embedding layer that takes in just the captions as input and with training, learns to identify syntactic and semantic meaning of the words. The original paper states there was marginal improvement in quality of the generated caption in using a pre-trained word embedding such as Word2vec or GloVe when compared to training the embeddings from scratch. Given that, we followed their recommendation and trained our own embeddings. The second component is the LSTM cell which takes as input the embeddings and features to generate the caption. We also have a dropout layer at the final stage of the decoder which regularizes the model such that it doesn't overfit our training data.

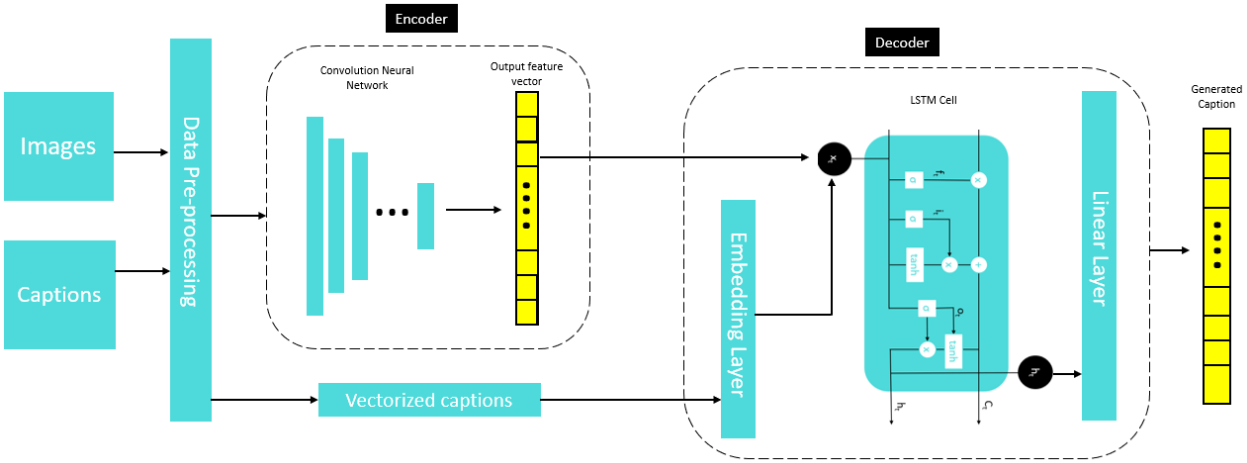


Figure 4.1: Baseline Model Architecture

4.1.2 Baseline+ Model

The setup for our baseline+ model follows closely with the baseline model however two key differences exist (see Figure 4.2 for overview of the architecture):

1. The addition of an attention module in the decoder
2. Omission of the final activation layer of the encoder such that the feature maps from the CNN can be passed in directly to the decoder

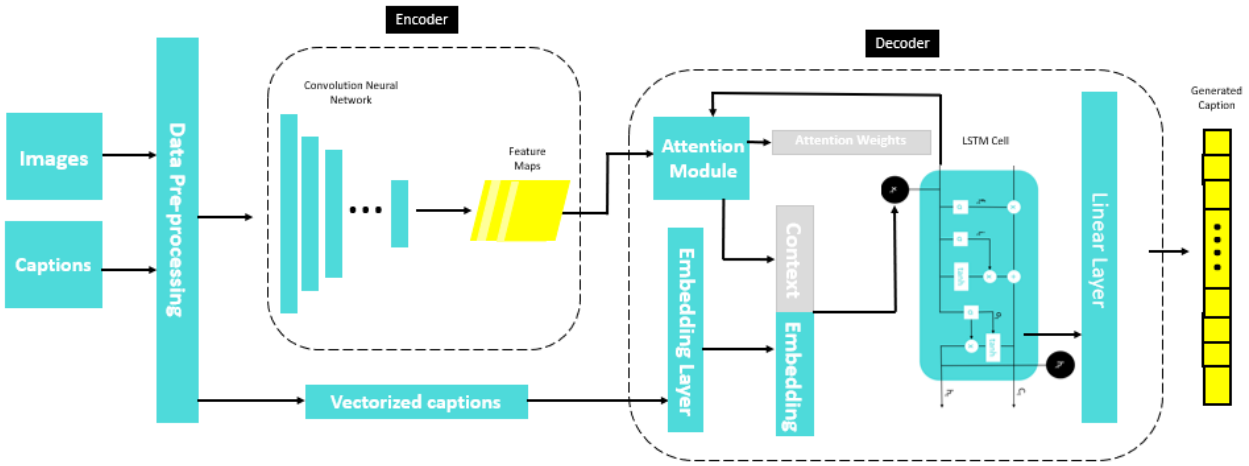


Figure 4.2: Baseline+ Model Architecture

4.2 Hyperparameters

The problem of computationally intensive training combined with long training times restricted our options in employing methods such as grid search for finding the optimal hyperparameter values. Instead, our choice of values were guided by those indicated by SOTA models and by referring to other implementations that tackle problems in the same problem domain as ours. Table 4.1 below shows the hyperparameter values used for the baseline and baseline+ models.

Model	Encoder Output Size	Embeddings Size	Decoder Hidden Size	Decoder Layers	Attention Dimension	Learning Rate
Baseline	256	256	256	1	N/A	0.0003
Baseline+	2048(x5x5)	256	512	1	256	0.0003

Table 4.1: Hyperparameter values

4.3 Loss and Optimizers

To calculate loss for the model, we used Cross Entropy Loss which is often used for classification problems (see below for formula) . During training, the model outputs the likelihood of each word at a given position in the caption and compares it to the expected value. Where differences lie, the error is used during backpropagation to update model parameters. Note that as we sampled captions of different lengths in the same batch during training, we set an additional parameter in our loss function to ignore any padding that was added to ensure the dimensions match.

$$CE = -\sum_{i}^{C} y \log(\hat{y}) \qquad \begin{array}{l} C = size\ of\ vocab \\ y = expected\ word \\ \hat{y} = predicted\ word \end{array}$$

For model optimization, we used Adam, as Xu et al. (2015) found this worked best with the Flickr30k dataset. This optimizer combines the best properties of AdaGrad and RMSProp and tends to be insensitive to hyperparameters compared to the rest. We used the default parameter configuration with exception to the learning rate which was set to 0.0003 for which we observed good performance.

4.4 Model Inference

During testing, as the objective is to generate a caption, we use the output from the linear layer of the decoder and pass the tokens through a softmax layer to get the probability distribution of each token at a given timestep. We employ a greedy search and choose the token with the highest probability for that time step and repeat the process for each subsequent word until either a “<EOS>” token is encountered or the maximum number of words are reached (set to 20 in our case).

5. Model Evaluation

In this section we will go over the metrics used for model evaluation and discuss how our models performed respective to those metrics.

5.1 Evaluation Metrics

For language models, a commonly reported metric is the BiLingual Evaluation Understudy (BLEU) score. Although other metrics for model evaluation exist (e.g. METEOR, CIDER), we use BLEU score as it allows us to compare our model performance with metrics reported by other SOTA models on the same dataset. We use the implementation from the nltk library.

BLEU score works by comparing a set number of sequential words between the generated and reference caption, the intuition being that greater the overlap between the two, the higher the quality of the caption generated. This is typically calculated for the n=1,2,3,4 grams where the score for each n is

equally weighted. We report this equally weighted average as the Cumulative BLEU-4 score. However, we are also interested in how the model performs for each n-gram overlap individually thus also report BLEU 1,2,3,4-gram scores. We use an out of the box smoothing function from the nltk library that adds a small epsilon value to handle cases when there are no n-gram overlaps between the reference and target captions[14].

5.2 Results

5.2.1 Training

For training, we used mini-batches (with batch size = 128) such that model parameters are updated more frequently. We monitored the following metrics during training:

1. Train loss - to ensure the model is learning, we expect this metric to steadily decline with training
2. Validation loss - to ensure the model generalizes well to unseen data, we set our early stopping condition to when the validation loss for an epoch increases over at least 3 epochs

Table 5.1 below reports the results obtained from these metrics during training for the baseline and baseline+ model, whereas Figure 5.2.1 and 5.2.2 show the training and validation loss for each of the models respectively.

	Epochs	Time (min)	Last Training Epoch Loss	Last Validation Epoch Loss
Baseline	41	365.85	2.617	3.004
Baseline+	27	265.22	2.603	3.174

Table 5.1 : Training Results

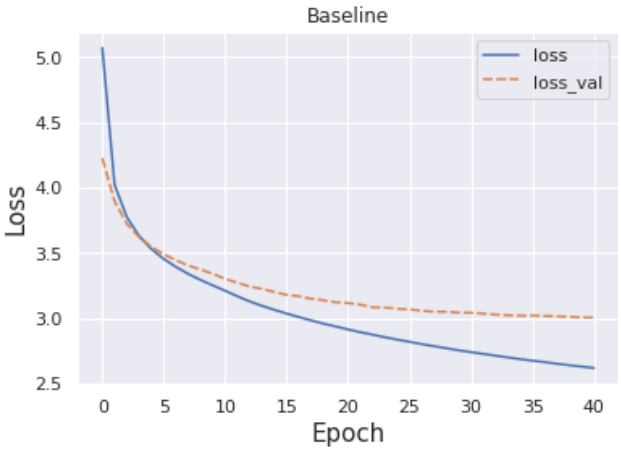


Figure 5.2.1: Baseline Training Loss

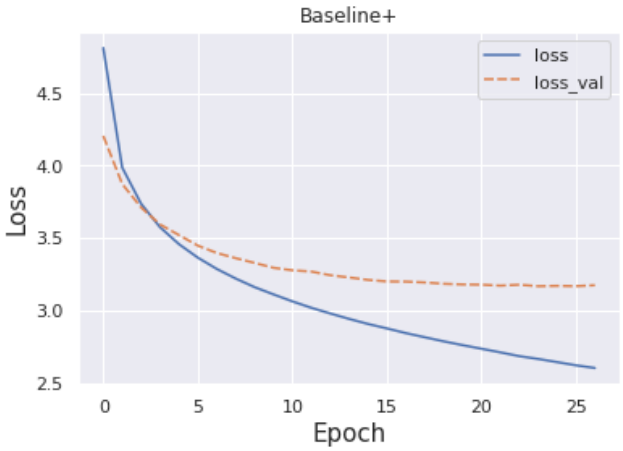


Figure 5.2.2: Baseline+ Training Loss

By inspecting the loss diagrams we notice that the Baseline+ model was able decrease its loss at a much faster rate than the Baseline model. It finished training 14 epochs earlier while reaching nearly identical loss values on the training set. Once the early stopping was triggered for both trainings, the Baseline model actually had a slightly lower loss value than the Baseline+ for the validation set. The difference between the two loss values is relatively minor so further analysis will be done using the BLEU score.

Figure 5.3 below shows the model performance during training for each of the tracked metrics. The general trend observed is in line with expectations, where the Baseline+ model reported higher scores all across compared to the baseline model. We also note that the BLEU scores for the train set are slightly higher than the validation set as the model parameters are adjusted based on errors calculated on the train set. However, the scores for the validation set are still comparable which indicates the model should exhibit similar performance on the test set.

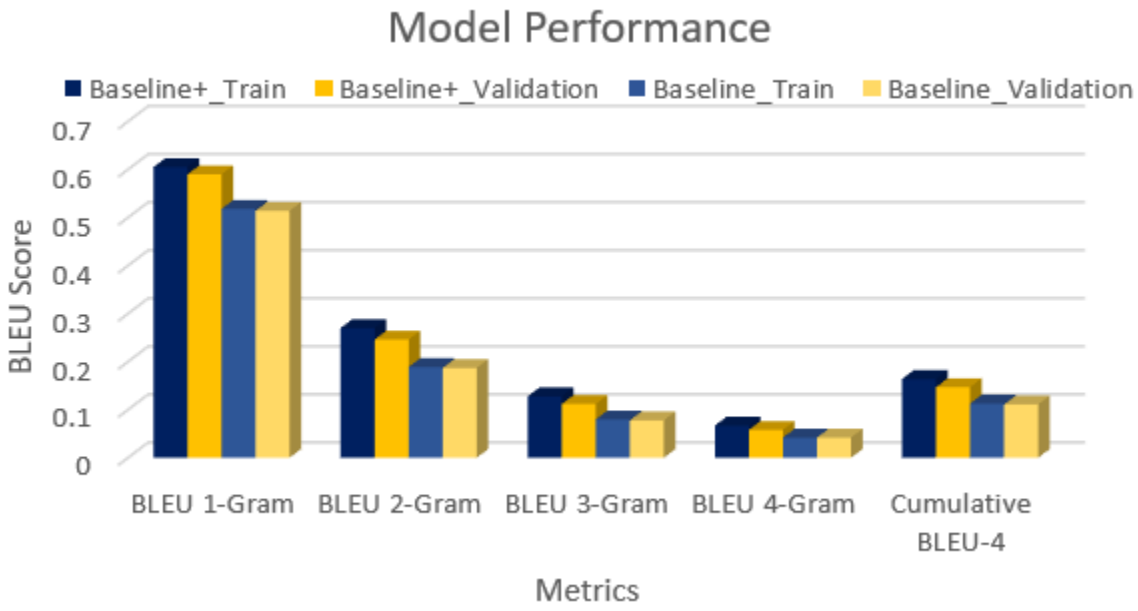


Figure 5.3: Model Performance on Train and Validation Sets

5.2.2 Test

Table 5.2 below shows our model accuracy on our 1000 image test set. Each image was fed through the two models to generate a caption and was then compared against the five reference captions. Even though at the end of training, validation loss for the baseline model was slightly lower, the Baseline+ model had higher BLEU scores for all 5 metrics. As expected, the scores decrease for both models when we evaluate longer n-grams.

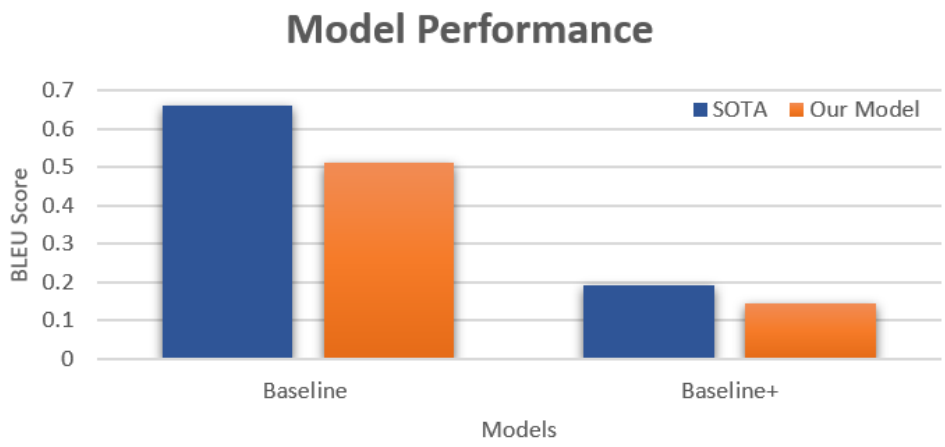
	BLEU 1-Gram	BLEU 2-Gram	BLEU 3-Gram	BLEU 4-Gram	Cumulative BLEU-4
Baseline	0.518	0.185	0.077	0.041	0.110
Baseline+	0.593	0.249	0.111	0.057	0.146

Table 5.2: Testing Results

The aforementioned BLEU-4 score is often used by translators to evaluate the quality of machine translations when compared to actual translations. With an average score of 0.110 and 0.146 respectively, neither model can be reliably used to generate a caption for an image. They would need a score at least 0.2 to make some sense but with many grammatical errors[12]. A caption generated by our Baseline+ model can be seen in Figure 5.4.

Figure 5.4: Sample Baseline+ Test

Our model results fall short of those reported by NIC. We compare the BLEU-1 scores reported by Vinyals et al (2014) with our baseline model. For the Flickr30k and Flickr8k dataset, NIC achieves a score of 0.66 and 0.63 respectively where our baseline closely follows at 0.51. For our Baseline+ model which incorporates a soft attention mechanism we compare with Xu et al (2015). We compare the BLEU-4 score reported; where our model achieved 0.146, SOTA reports 0.191. Figure 5.5 below shows a graph comparing our results to those from the original paper (See Appendix C for full breakdown of SOTA BLEU scores).



**Note that for the scores reported for Baseline are BLEU-1 and Baseline+ are BLEU-4*

Figure 5.5: Comparison of Model Performances

With the addition of the attention mechanism in the Baseline+ model, we also were able to visualize the regions of the image the model is considering during the word generation. Figure 5.6 shows the highlighted parts of the image are where the attention is when the word is generated.

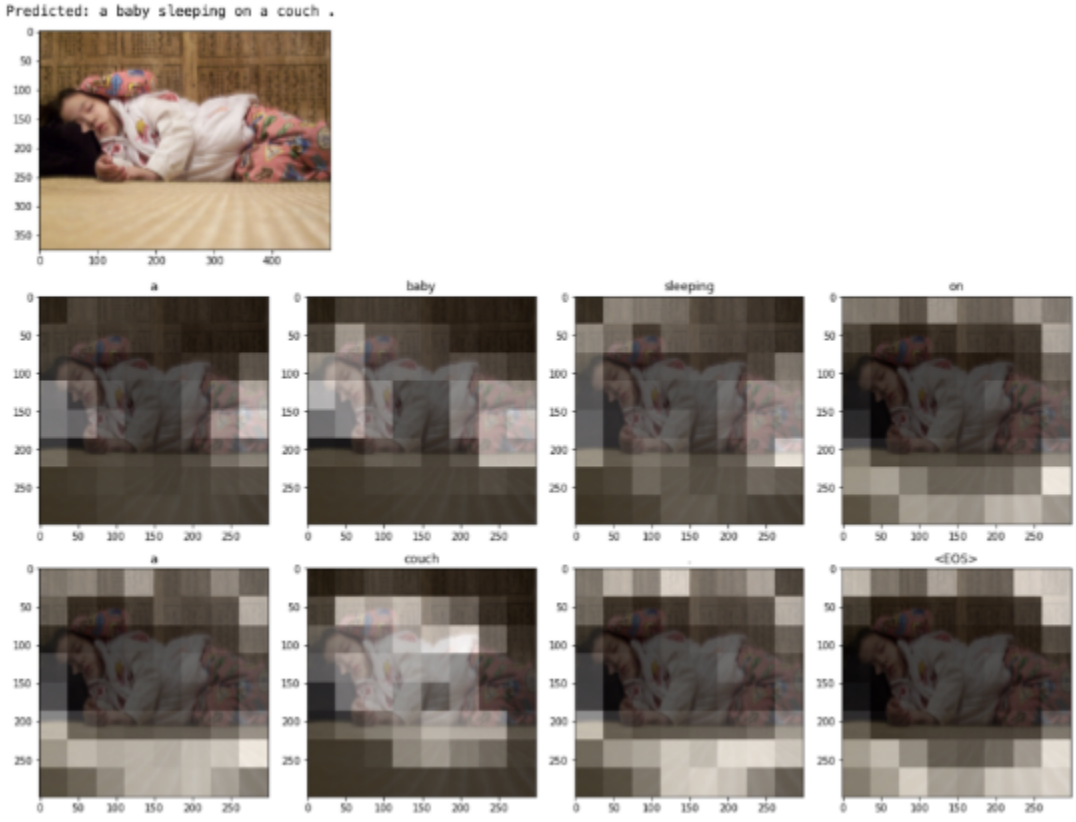


Figure 5.6: Attention Visualization

6.0 Conclusion (Priyanka Velagala)

In our implementation, although we used best practices and mimicked the architecture set out by the NIC model in Vinayals et al (2014) and Xu et al (2015), unfortunately our model boded quite poorly in comparison. The BLEU scores attained indicate that the model has not sufficiently learned to form a meaningful text description based on contents of an image. Between the two implementations, the Baseline+ model, which incorporates soft attention on the Baseline model, expectedly performed better given that the model has a mechanism to more strategically focus on certain regions of the image when generating the caption.

One reason as to why our model was unable to perform as well as the architectures we followed in our implementation could be due to the use of only a subset of the full data which means the model was exposed to fewer examples during training. We also deviated from the implementation set out in the papers we followed in other regards. One such being that although we used the same word frequency threshold (5) as indicated in the paper, with the subset of the data we used, this yielded a vocabulary of approximately 8k words, whereas Vinayals et al (2014) specified a vocabulary of 10k which implies our model had to learn nearly just as many representations with nearly one third of the training examples. These discrepancies from the original architecture could have all contributed to the overall worse performance.

An aspect that we can be critical of in our model evaluation technique is the use of BLEU score as an accuracy measure. BLEU score is generally used to evaluate translations since there are only so many ways a sentence can be expressed. However, there are some obvious drawbacks to using this metric. For example, one such being that the model penalizes all errors equally, although omission or misplacement of certain words can change the meaning of the sentence entirely. Another drawback is that the quality of the generated caption is evaluated strictly in relation to the reference sentences. A low BLEU score doesn't necessarily imply the quality of the generated caption is poor, only that it doesn't strictly match the reference sentence. However in the context of images, since a drastically great amount of descriptive captions can be made, it is possible that certain variations of the reference caption allow the generated caption to have a higher or lower BLEU score based on the aspect of the image being described.

An enhancement to the model we would have liked to explore is to evaluate differences in model performance between greedy and beam search for caption generation. Typically, beam search tends to yield higher quality captions as instead of choosing the next token in the sequence with the highest probability, based on the hyperparameter beam size (k), the algorithm continues to build the caption by choosing k next tokens with the highest probabilities until maximum caption length is reached or the <EOS> token is encountered. However, an obvious disadvantage to this approach is the higher time complexity of this algorithm compared to greedy. Also, the algorithm doesn't guarantee finding the output sequence with the highest score [13].

7.0 Reflections

As we were working with limited computational and memory resources, we decided it was best to work with a subset of our full dataset. Initially we planned on working with 10,000 images with 1 caption each. A challenge that we encountered in doing so is that in early training stages,the same caption would be generated with objects the model encountered frequently (e.g. man, blue shirt, jeans, bench, building etc.) despite the objects not being present in the image. We also noticed poor sentence structure and repetition of the same word in the generated caption. We suspected the dataset was not sufficiently large for the model to meaningfully learn how to syntactically form a sentence. Once we expanded our dataset to use all 5 captions, we noticed improved results where the model learned to use the <EOS> token, improved quality of caption generated based on features in image etc.

In using only a subset of the data, we arbitrarily chose to use the first 10,000 images without consideration of the distribution of objects in the dataset. For example, if the images were somehow grouped sequentially by objects present, the model learns certain representations better than others. In our case, we noticed that the model almost always accurately identifies when there is a man in the picture and often tends to misclassify pictures of women with men. This becomes even more problematic when each of the data splits have different distributions.

In retrospect, there should have been more consideration to mitigate effects of different data distributions. One solution is to have dedicated more time in the preliminary stages to analyzing the captions to detect which objects were present in the image. However this was not feasible given the time constraints of the project. Alternatively, a different dataset could have been used such as MSCOCO which labels images by broader categories of objects present which would have allowed us to sample each category such that they're well represented in each split.

Ideally, we would also like to have explored using different CNNs for the encoder, different hyperparameter values and investigate their impact on model accuracy and performance. However, due to the complexity of the task, the model requires a lot of training prior to producing high quality captions. Therefore we felt restricted in terms of being able to carry out this type of analysis and settled by using the inception v3 and hyperparameter values that yielded reasonable performance.

8.0 References

[1] Elhagry, A. and Kadaoui, K., 2022. *A Thorough Review on Recent Deep Learning Methodologies for Image Captioning*. [online] arXiv.org. Available at: <https://arxiv.org/abs/2107.13114> [Accessed 30 April 2022].

[2] Chohan, M., Khan, A., Saleem, M.A., Hassan, S., Ghafoor, A., & Khan, M. (2020). Image Captioning using Deep Learning: A Systematic Literature Review. *International Journal of Advanced Computer Science and Applications, 11*.

[3] Pytorch.org. 2022. *PyTorch*. [online] Available at: <https://pytorch.org/hub/pytorch_vision_inception_v3/> [Accessed 29 April 2022].

[4] Vinyals, O., Toshev, A., Bengio, S. and Erhan, D., 2022. *Show and Tell: A Neural Image Caption Generator*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1411.4555> [Accessed 31 April 2022].

[5] Google Cloud. 2022. *Advanced Guide to Inception v3 | Cloud TPU | Google Cloud*. [online] Available at: <https://cloud.google.com/tpu/docs/inception-v3-advanced> [Accessed 2 May 2022].

[6] Medium. 2019. *The differences between Inception, ResNet, and MobileNet*. [online] Available at: <https://medium.com/@fransiska26/the-differences-between-inception-resnet-and-mobilenet-e97736a709b0> [Accessed 4 May 2022].

[7] Education, I., 2020. *What are Recurrent Neural Networks?*. [online] Ibm.com. Available at: <https://www.ibm.com/cloud/learn/recurrent-neural-networks> [Accessed 5 May 2022].

[8] Weng, L., 2018. *Attention? Attention!*. [online] Lilianweng.github.io. Available at: <https://lilianweng.github.io/posts/2018-06-24-attention/> [Accessed 5 May 2022].

[9] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R. and Bengio, Y., 2016. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1502.03044> [Accessed 6 May 2022].

[10] 2018. *Attention in Neural Networks*. [video] Available at: <https://youtu.be/W2rWgXJBZhU> [Accessed 6 May 2022].

[11] Makarov, A., 2020. *Image Captioning with Attention: Part 1*. [online] Medium. Available at: <https://medium.com/analytics-vidhya/image-captioning-with-attention-part-1-e8a5f783f6d3> [Accessed 6 May 2022].

[12]Google Cloud. 2022. *Evaluating models | AutoML Translation Documentation | Google Cloud*. [online] Available at: <https://cloud.google.com/translate/automl/docs/evaluate> [Accessed 7 May 2022].

[13]Meister, C., Vieira, T. and Cotterell, R., 2021. *Best-First Beam Search*. [online] arXiv.org. Available at: <https://arxiv.org/abs/2007.03909> [Accessed 7 May 2022].

[14]Lee, C., Li, H., Hou, R. and Lim, C., 2022. *NLTK :: nltk.translate.bleu_score*. [online] Nltk.org. Available at: <https://www.nltk.org/_modules/nltk/translate/bleu_score.html> [Accessed 7 May 2022].

9.0 Appendix

Appendix A - SOTA Model Performances

Method	CIDEr	SPICE
Resnet Baseline	111.1	20.2
UpDown	120.1	21.4
MLE Maximization	110.2	20.3
*RL Maximization	120.4	21.3
*MLE + RL Maximization	119.3	21.2
*Meta Learning	121.0	21.7
IC-GAN (Updown/CNN-GAN)	123.2	22.1
IC-GAN (Updown/RNN-GAN)	122.2	22.0
IC-GAN (Updown/ensemble)	**125.9**	**22.3**

Table 9.1 - Overall performance on MS COCO Karpathy test split[1]

Appendix B - Code References and Running Test Cases

[1] Persson, A (2021) Image Captioning [Source Code].
https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/more_advanced/image_captioning

[2] Bashyal, A (2020). Image Captioning with Attention [Source Code]
https://www.kaggle.com/code/mdteach/image-captioning-with-attention-pytorch

[3] Udacity (2021). CVND---Image Captioning-Project [Source Code]
https://github.com/udacity/CVND---Image-Captioning-Project

[4] Makarov, A (2020). Image-Captioning-With-Attention [Source Code]
https://github.com/MakarovArtyom/Image-Captioning-with-Attention

Data Source: https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset
We performed our analysis on Google Colab. Please see the README.md file in the zip folder for instructions on how to run the performance analysis notebook.

Appendix C - Model Performance of NIC

Metric	BLEU-4	METEOR	CIDER
NIC	**27.7**	**23.7**	**85.5**
Random	4.6	9.0	5.1
Nearest Neighbor	9.9	15.7	36.5
Human	21.7	25.2	85.4

Table 1. Scores on the MSCOCO development set.

Approach	PASCAL (xfer)	Flickr 30k	Flickr 8k	SBU
Im2Text [24]				11
TreeTalk [18]				19
BabyTalk [16]	25			
Tri5Sem [11]			48	
m-RNN [21]		55	58	
MNLM [14][5]		56	51	
SOTA	25	56	58	19
NIC	**59**	**66**	**63**	**28**
Human	69	68	70	

Table 2. BLEU-1 scores. We only report previous work results when available. SOTA stands for the current state-of-the-art.

Table 9.3 - BLEU-1 Scores of NIC [1]

		BLEU				
Dataset	Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR
Flickr8k	Google NIC(Vinyals et al., 2014)†Σ	63	41	27	—	—
	Log Bilinear (Kiros et al., 2014a)°	65.6	42.4	27.7	17.7	17.31
	Soft-Attention	**67**	44.8	29.9	19.5	18.93
	Hard-Attention	**67**	**45.7**	**31.4**	**21.3**	**20.30**
Flickr30k	Google NIC†∘Σ	66.3	42.3	27.7	18.3	—
	Log Bilinear	60.0	38	25.4	17.1	16.88
	Soft-Attention	66.7	43.4	28.8	19.1	**18.49**
	Hard-Attention	**66.9**	**43.9**	**29.6**	**19.9**	18.46
COCO	CMU/MS Research (Chen & Zitnick, 2014)a	—	—	—	—	20.41
	MS Research (Fang et al., 2014)†a	—	—	—	—	20.71
	BRNN (Karpathy & Li, 2014)°	64.2	45.1	30.4	20.3	—
	Google NIC†∘Σ	66.6	46.1	32.9	24.6	—
	Log Bilinear°	70.8	48.9	34.4	24.3	20.03
	Soft-Attention	70.7	49.2	34.4	24.3	**23.90**
	Hard-Attention	**71.8**	**50.4**	**35.7**	**25.0**	23.04

Table 9.4 - BLEU Score of NIC with Attention [4]