Image Caption

– Coco2014: Describe Image Contexts in Words

Abstract

The task is to generate natural language descriptions of given images. In deep learning, this task can be called “image caption”. By inputting a picture, the model will describe the picture contents in English. Tasks like this usually implement an Encoder-Decoder structure, with an encoder following CNN architecture to encode an image to extracted features, and a RNN decoder. The decoder uses the information extracted by the encoder and generates the description word by word. The model performance is evaluated by BLEU score, to measure the consistency between prediction sentences and groundtruth(label) sentences. In this assignment, we implemented two decoder structures: RNN with Teacher Forcing, RNN with Attention, and compared their performances on the given task.

1 Introduction

Image Caption is the task of predicting a caption/description of a given image. Use applications are aiding visually impaired people that can help them navigate through different situations. In other words, it helps to improve the content accessibility for people by describing images for them[1].

The dataset used in this task is coco2014 dataset[2]. It contains 82,783 training images, 40,504 verification images, and 40,775 test images. The dataset split method and annotations are provided by Andrej Karpathy[3]. A code framework for image and data pre-processing are provided. All images will be resized and normalized. Values on RGB channels will in addition be regularized by mean and deviation. Since the captions need to be the ground truth labels and also the input of the decoder, we need to expand them to a uniform length by <*start*> and <*end*> as placeholders. This operation efficiently reduces the time cost on padding.

Encoder in this case is an implementation of ResNet101[4]. By replacing the last Pooling layer and full connect layer with adaptive pooling, we could generate encoding results with a fixed size. Encoder we used is pre-trained on ImageNet, and fine-tune[5] can be invoked for better training results.

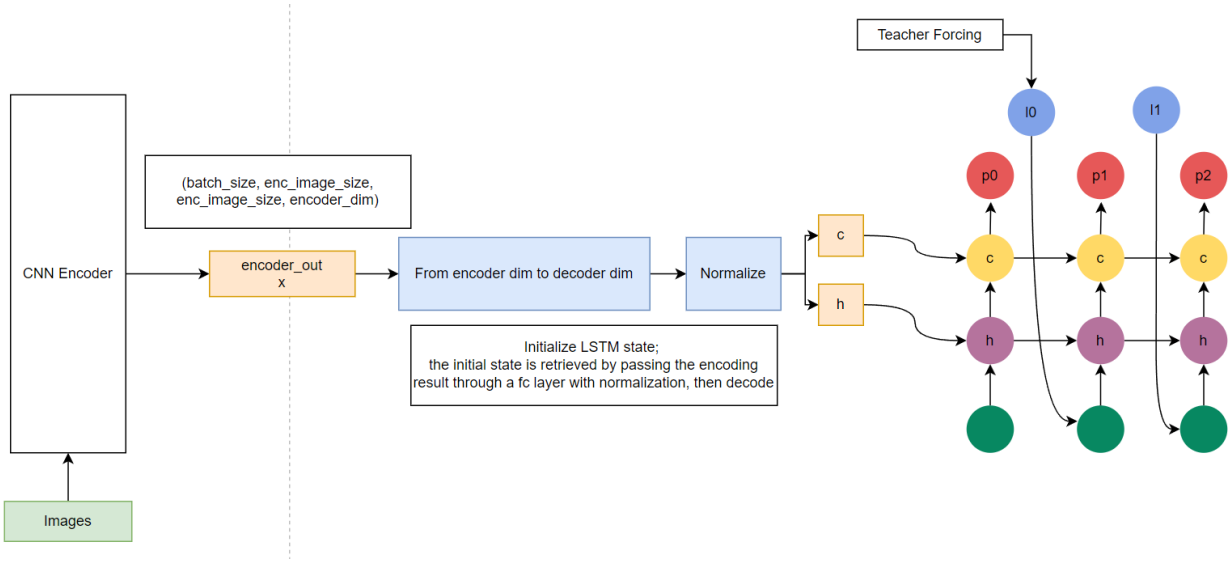
Inspired by “Show and Tell: A Neural Image Caption Generator”[6] and “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”[7], we reproduced two different decoder architectures, RNN with teacher forcing mechanism and RNN with attention. More details will be provided in later sections.

2 Decoder Architectures

Two decoder architectures are implemented in this experiment. We compared and contrasted on how they behave on the same dataset under the same task. In this section, we will shortly introduce each structure and the associated code. The code implemented is inspired by: https://gitcode.com/sgrvinod/a-pytorch-tutorial-to-image-captioning/overview .

2.1 RNN with Teacher Forcing

Teacher forcing is a training technique used in machine learning and natural language processing to improve the performance of recurrent neural networks (RNNs). It involves using the correct output sequence as the input to the RNN at each time step during training, rather than using the predicted output of the RNN as the input[8].



Above is a structure diagram drawn in the learning process. Detailed formulas for this model are stated in [6]. LSTM is used for decoding information.

Once the images have finished embedding, the result will go through a full connect layer with normalization and a decode to initialize the LSTM state, that is, hidden state and cell(memory) state. The predicted probability is calculated by a FC layer and softmax. To move from timestep t to t+1, we use the correct label word rather than the prediction from the last step as the input. This mechanism is known as “teacher forcing” mentioned above. By using this technique, the model can generally converge faster and reduce error propagation, which refers to mistakes made early in the sequence that can compound and affect the quality of later predictions[9].

```
    # Initialize LSTM state
    # the initial state is retrieved by passing the encoding result through a fc layer with normalization
    # then decode the normalized results
    init_input = self.bn(self.init(encoder_out))
    h, c = self.decode_step(init_input)  # (batch_size_t, decoder_dim)

    # for each time step
    for t in range(max(decode_lengths)):
        # retrieve the index for current timestep
        # remember that each image has different decode length so need to do add up
        i = sum([l > t for l in decode_lengths])
        preds, h, c = self.one_step(
            embeddings[:i, t, :], h[:i], c[:i])
        predictions[:i, t, :] = preds
    return predictions, encoded_captions, decode_lengths, sort_ind

def one_step(self, embeddings, h, c):
    h, c = self.decode_step(embeddings, (h, c))
    # Compute the scores over the vocabulary
    preds = self.fc(self.dropout_layer(h))
    return preds, h, c
```

Here are code snippets for implementing this decoder.

One thing needs to be noticed is that due to the teacher forcing mechanism, and since each image’s caption is of different length, when we try to retrieve the “correct” word at time step t, we need to sum up the length of the captions to get to the right index of the word.

From the code snippets, we could in addition see a dropout layer before feeding the hidden state into the FC layer for generating probabilities. This dropout layer can be removed, that is, we could choose whether to embed any dropout mechanism. Since we already have “dropout” in model configuration, we will use this mechanism in the model.

2.2 Attention

This ability of self-selection is called attention. It is a simulation of a complex cognitive ability that human beings possess. The attention mechanism allows the neural network to have the ability to focus on its subset of inputs to select specific features[10]. Below is the representation of how attention weights are calculated. Code snippets also show the breakdown of this formula.

$$\alpha = softmax\left(fc\left(relu\left(fc(encoder\_output) + fc(h)\right)\right)\right)$$

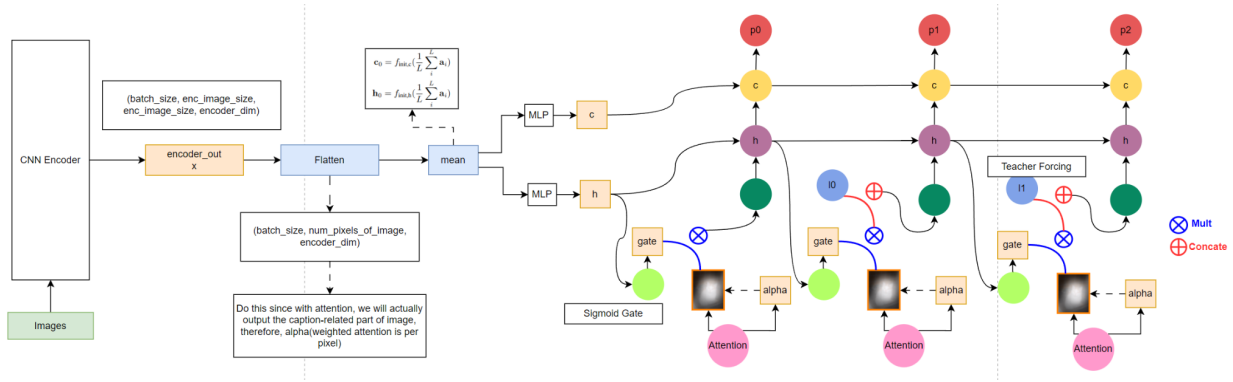
```
# "e = f_att(encoder_out, decoder_hidden)"
# "alpha = softmax(e)"
# "z = alpha * encoder_out"
encode_attention = self.encoder_attention(encoder_out)  # (batch_size, num_pixels, attention_dim)
decode_attention = self.decoder_attention(decoder_hidden)  # (batch_size, attention_dim)
# (batch_size, num_pixels, attention_dim) -> (batch_size, num_pixels)
e = self.att(self.relu(encode_attention + decode_attention.unsqueeze(1))).squeeze(2)
alpha = self.softmax(e)  # (batch_size, num_pixels)
z = (encoder_out * alpha.unsqueeze(2)).sum(dim=1)  # (batch_size, encoder_dim)
return z, alpha
```

The *‘encoder_attention’* and *‘decoder_attention’* are the two FC layers in the formula above. With these two layers, one could transform the tensor from the encoder output and the decoder output to the dimension of attention. Except alpha, the attention weight, the calculation also generates a “z”, a dynamic representation of relevant part of the image input at timestep t.

2.3 RNN with Attention and Teacher Forcing

Teacher forcing is a training technique used in machine learning and natural language processing to improve the performance of recurrent neural networks (RNNs). It involves using the correct output sequence as the input to the RNN at each time step during training, rather than using the predicted output of the RNN as the input[8].

Below is a structure diagram drawn in the learning process. Detailed formulas for this model are stated in [7]. LSTM is used for decoding information.



The decoding process is similar to the RNN decoder without attention, but just needs to add the attention weighted embedding part of images back into the decoding calculation.

```
    # Teacher forcing is used.
    # At each time-step, decode by attention-weighting the encoder's output based
    # on the decoder's previous hidden state output
    # then generate a new word in the decoder with the previous word and the attention weighted encoding
    # Your Code Here!
    for t in range(max(decode_lengths)):
        i = sum([l > t for l in decode_lengths])
        attention_weighted_encoding, alpha = self.attention(encoder_out[:i], h[:i])
        gate = self.beta(h[:i])
        attention_weighted_encoding = gate * attention_weighted_encoding
        decoder_teacher_forcing = torch.cat([embeddings[:i, t, :], attention_weighted_encoding], dim=1)
        h, c = self.decode_step(decoder_teacher_forcing, (h[:i], c[:i]))
        preds = self.fc(self.dropout_layer(h))
        predictions[:i, t, :] = preds
    ##########################################################################
    return predictions, encoded_captions, decode_lengths, alphas, sort_ind

def one_step(self, embeddings, encoder_out, h, c):
    ##########################################################################
    # To Do: Implement the one time decode step for forward pass
    # this function can be used for test decode with beam search
    # return predicted scores over vocabs: preds
    # return attention weight: alpha
    # return hidden state and cell state: h, c
    # Your Code Here!
    attention_weighted_encoding, alpha = self.attention(encoder_out, h)
    gate = self.beta(h)
    attention_weighted_encoding = gate * attention_weighted_encoding
    decoder_input = torch.cat([embeddings, attention_weighted_encoding], dim=1)
    h, c = self.decode_step(decoder_input, (h, c))
    preds = self.fc(self.dropout_layer(h))
    ##########################################################################
    return preds, alpha, h, c
```

The code implementation is inspired by: “a-PyTorch-Tutorial-to-Image-Captioning”[11].

3 Parameter Tuning

There are in total 82,783 images in the train set, which corresponds to 413,915 data points if we set the per image caption length to 5. Therefore, in order to iterate fast to find a proper combination of parameters and model structures.

```
    # get a smaller subset
    indices = self.get_random_subsets(subset_size)
    imgs = self.imgs[indices]
    self.imgs = imgs
    caplens_subset = []
    captions_subset = []
    for i in indices:
        caplens_subset += self.caplens[(i*self.cpi):((i+1)*self.cpi-1)]
        captions_subset += self.captions[i*self.cpi:((i+1)*self.cpi-1)]
    self.caplens = caplens_subset
    self.captions = captions_subset
    # Total number of datapoints
    self.dataset_size = len(self.captions)
    print(len(self.imgs))
    print(self.dataset_size)


def get_random_subsets(self, subset_size=0.1):
    """
    Generate a smaller dataset from the overall dataset for faster training and parameter tuning iteration
    """
    amount = floor(len(self.imgs)*subset_size)
    indices = sorted(sample(range(len(self.imgs)-1), amount)) # sort indices in assending order
    return indices
```

Above is the code to generate a much smaller subset from all data randomly. The subset size suggests how much percentage of data we want to have in the subset. For tuning parameters, we take 20% of the data.

3.1 Performance Evaluation

There are in total three parameters we used in this task to measure a model performance, loss, accuracy, and BLEU.

BLEU, or the Bilingual Evaluation Understudy, is a score for comparing a candidate translation of text to one or more reference translations. Although developed for translation, it can be used to evaluate text generated for a suite of natural language processing tasks[12].

The reason we are not only using accuracy scores is that, "accuracy" is the percentage of correctly predicted tokens, "accuracy per sequence" is the percentage of correctly predicted sequences (sentences), i.e. sequences where all tokens are correctly predicted. On the other hand, BLEU is a more document-level metric. It measures not only on how much the model gets correct, but also grammatical quality.

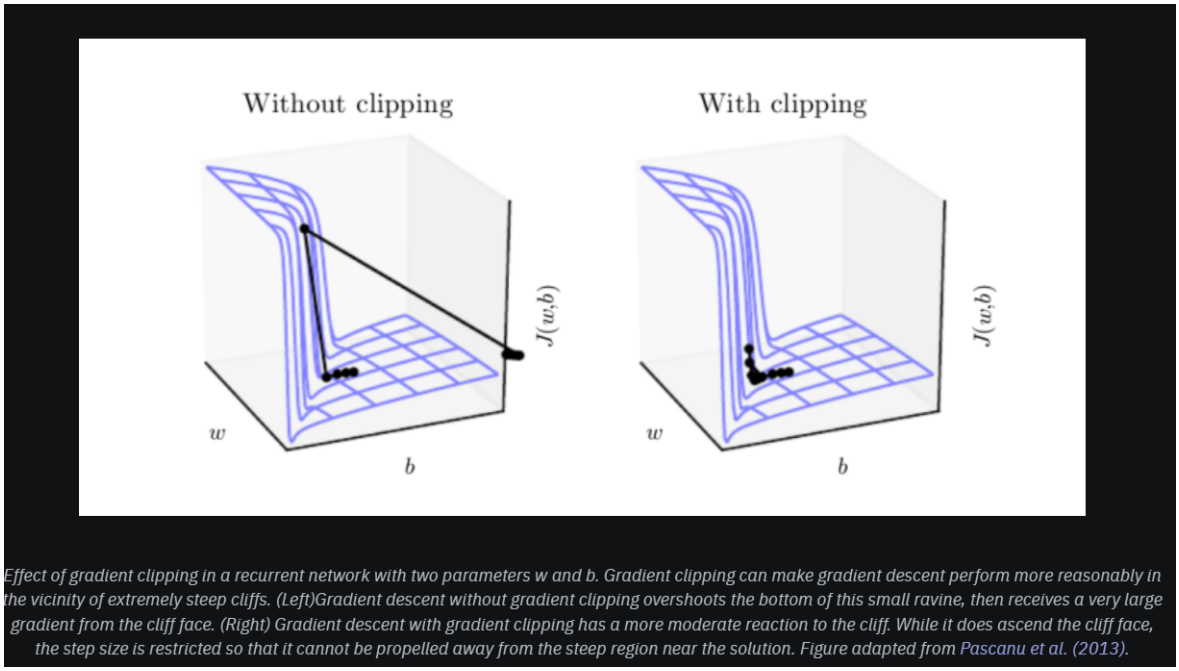
BLEU scores fall between 0-100. When a BLEU score is lower than 20, the prediction is almost useless. When the score is between 20 to 30, the description is good but has significant grammatical errors. When it goes above 30, we can generally say the generated text is good enough. Once the score is over 60, the quality is often better than human[13].

In this section, since we only use a small dataset, it is under prediction that the BLEU scores and accuracy metrics may not be ideal. Therefore, the goal is to get the hyperparameter combination that provides an ideal learning trend on the model.

3.2 Gradient Clipping

Adam optimization algorithm is used in the training process. It is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing[15].

Gradient clipping is also introduced to avoid exploding gradients, especially in back propagation. It is a method where the error derivative is changed or clipped to a threshold during backward propagation through the network, and using the clipped gradients to update the weights. By rescaling the error derivative, the updates to the weights will also be rescaled, dramatically decreasing the likelihood of an overflow or underflow.



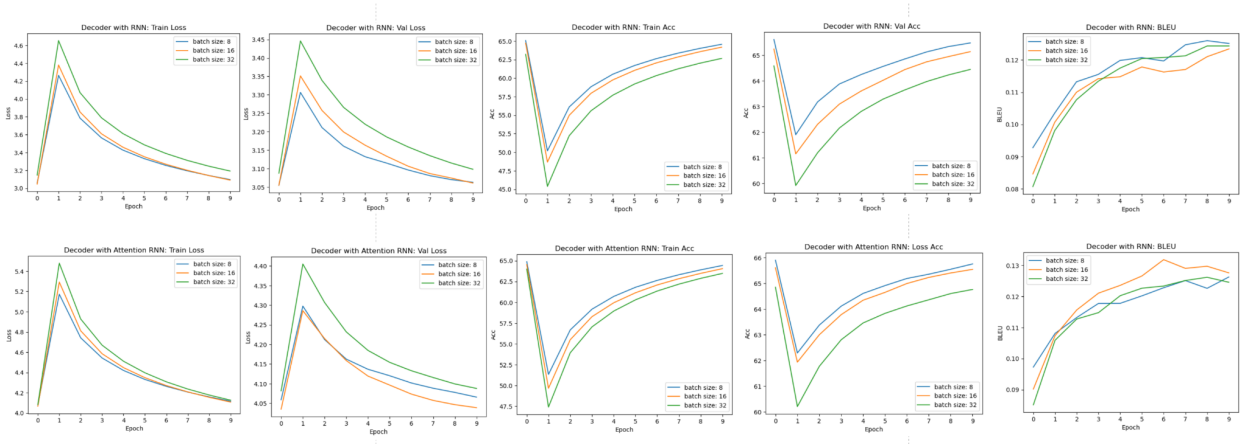
Effect of gradient clipping in a recurrent network with two parameters w and b. Gradient clipping can make gradient descent perform more reasonably in the vicinity of extremely steep cliffs. (Left)Gradient descent without gradient clipping overshoots the bottom of this small ravine, then receives a very large gradient from the cliff face. (Right) Gradient descent with gradient clipping has a more moderate reaction to the cliff. While it does ascend the cliff face, the step size is restricted so that it cannot be propelled away from the steep region near the solution. Figure adapted from Pascanu et al. (2013). [16].

3.3 Batch Size

The batch size affects some indicators such as overall training time, training time per epoch, quality of the model, and similar. Usually, we chose the batch size as a power of two, in the range between 16 and 512. But generally, the size of 32 is a rule of thumb and a good initial choice[14].

Fine tuning on the pre-trained encoder is off for this set of experiments, and the decoder learning rate is 4e-4. If a model has not improved from its best performance, the learning rate will follow a schedule to shrink(decay).

We tested models with batch size: 8, 16 and 32.



Observations:

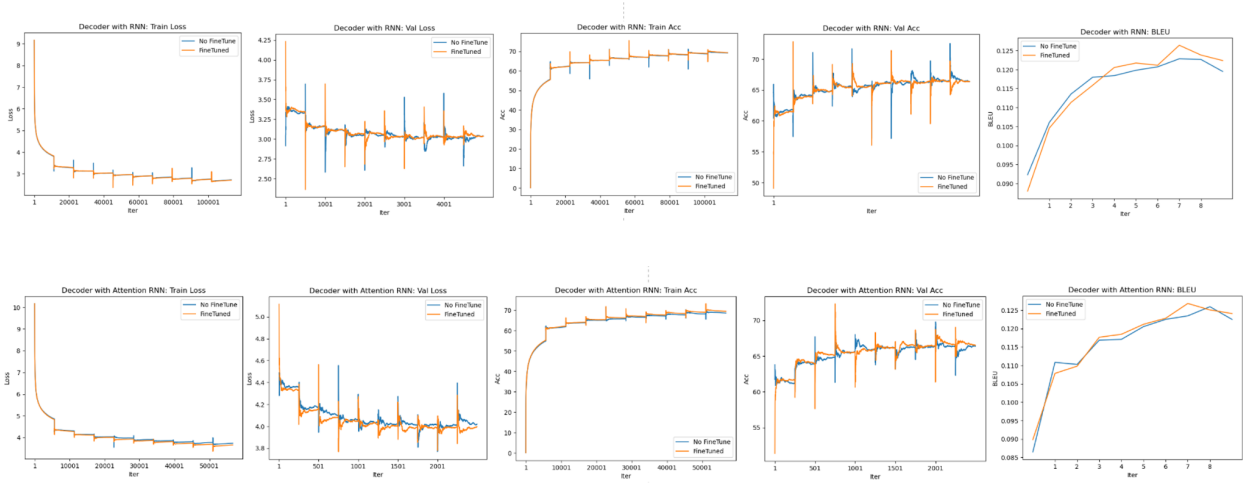
1. The accuracy score of a model has negative correlation with the loss. The lower the loss gets, the higher the accuracy will reach.
2. With different batch sizes, the loss first increases drastically from epoch 1 to 2, then follows a downward trend. One potential cause of this event is the annealing on the learning rate, or the gradient clipping has affected the convergence.
3. Decoder with RNN:
    - The smaller the batch size is, the faster the model converges. It allows the model to start learning before seeing a comparatively large amount of data, and also updates the model in a more frequent manner than with a larger dataset.
    - Batch size 32 has the highest loss and lowest accuracy on both the training set and the validation set.
    - Batch size 8 has the best accuracy score and BLEU metrics. Batch 32 has the second best BLEU score. Loss curve and accuracy curve with batch size 16 starts to approximate batch 8's after epoch 6, but has a comparatively low BLEU score. This suggests that a. Accuracy and BLEU do not necessarily correlate; b. Within a certain range, the batch size provides a more accurate direction of decline in training loss, while reaching to a certain degree, the direction of decline is not changing[17].
4. Decoder with RNN and Attention:
    - Unlike pure RNN, batch size 16 has the best performance among the three values we tested with Attention RNN.
    - Batch size 32 has the worst performance with the highest loss and the lowest accuracy. Its BLEU score is close to batch size 8, and gets better than batch size 8 after epoch 4. This is potentially caused as the model starts to learn more features that can be generalized to the unknown data, rather than picking up those that could result in overfitting.
5. The highest BLEU score is at around 0.12-0.13, which indicates that the captions are not of usable quality according to Section 3.1. However, again, since we only used a quite small dataset, there can be many features that are not exposed to the model in the training process. We only want to select the hyperparameter that has the correct and comparatively ideal learning trend in this section. We will use RNN with batch size 8 and Attention RNN with batch 16 in the later tuning stage.

3.4 Fine Tune Encoder

For encoding an input image to feature encoders, we use a pretrained ResNet101. The weight is trained on the ImageNet dataset.

Here we introduced a concept as “transfer learning”. The intuition behind transfer learning for computer vision tasks is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset[18]. With this technique, the training time and cost can be massively reduced.

Fine-Tuning in this case is to unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. It allows the pretrained model to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task[18].



Observations:

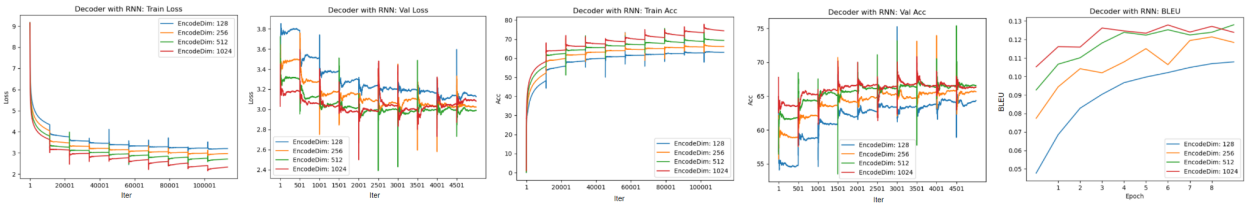
1. Decoder with RNN:
    - Loss curves and accuracy curves with encoder and fine-tuned encoder are close to each other. The performance of the model is not decisively affected by enabling the fine-tuning on the pretrained encoder. One reason could be that we are using a small dataset and a quite small batch size. Although using transfer learning means that we need way less data than training from scratch, we still need enough data to cover the diversity.
    - Fine-tuned encoder has a higher BLEU score than the non-tuned encoder. However, it is only higher by around 0.5. This would not be acknowledged as a significant improvement, given the BLEU score is already low enough.
2. Decoder with RNN and attention:
    - Similar to decoder with RNN only, fine-tuning does not bring in huge improvements in the model performance. However, the improvement is larger than with only RNN. This could be because we are using a larger batch size, and therefore more divergence is covered in the training and fine-tuning.

3. An overfitting seems to happen to the 4 models we tested in this section, after epoch 7. But we are not going to decrease the max epoch amount for final models or later parameter tuning, since the dataset here is small.
4. All the loss curves have a similar shape within every epoch: decrease, climb up aggressively, then temps to stabilize with a gentle upward trend. This is because the model finds a local minima, then pushes to the next optimal as the learning continues.
5. In conclusion, since the dataset we used here is of a small size, the divergence of the data is not fully exposed to the encoder. Therefore, the fine-tuning does not introduce significant changes, but only improvements at an extremely small rate. We will use fine-tuning on the final models to verify if greater changes can be made.

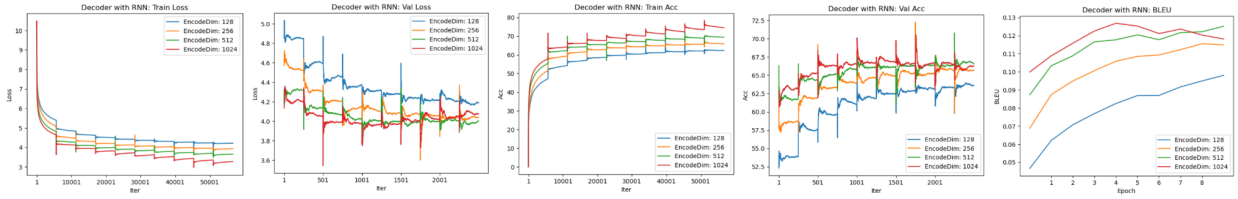
3.5 Embed and Decode Dimension

The concept of encoder and decoder are generally used for a Sequence to Sequence Model. The Sequence to Sequence Model consists of 3 parts: encoder, intermediate (encoder) vector, and decoder. The intermediate vector is the final hidden state produced by the encoder, and acts as the initial hidden state of the decoder. By increasing the dimension of embeddings and decodings, we are actually increasing the complexity and capacity of a neural network. Expressive capacity and effective model complexity are closely related. Generally speaking, a complex model is more likely to have good expressive capacity over a bunch of complicated hypotheses. However, a model that is too complex can end up over-fitting to random effects that are present only in the specific data-set used for training[19].

Decoder with RNN



Decoder with RNN and Attention



Observations:

1. For decoders RNN and attention with RNN, small embed and decode dimensions can result in low quality learning results. When the dimension is set to 128, the models have the worst performances, with the highest loss and lowest accuracy on both the train set and the validation set. As analyzed at the head of this section, the small dimension value corresponds to low

complexity and low expressive capacity of the model, that is, such a model will find it difficult to find a regression or expression for comparatively complicated relationships.
2. Before the dimension amount gets to 1024, the greater the dimension amount is, the better the model behaves. Dimension 1024 although having the best performances at the early stage of training, dimension 512 exceeds it after epoch 8 or 9. Also, due to the computation cost being way too much for the machine we used, we will not use this dimension for the final model.

3.6 Summary

Decoding with RNN and with Attention RNN both have similar learning results at the first few iterations. Given the same batch size and learning rate, RNN decoder has a faster convergence compared to Attention RNN. However, the highest accuracy and BLEU scores that RNN with attention has achieved are higher than RNN only on this subset of data. This indicates that the learning on captions, especially grammatical wise, should not only rely on the feature of the images, but also the importance of words. The attention mechanism has its advantages in this, although it can slow down the model efficiency, since there is more to learn.

Transfer learning provides the possibility to have a promising performance without the need of much training cost and data. In this section, models with either decoder mechanism have slightly improved the learning results with fine tuning on the pretrained encoder. The improvement is not significant, but highly possible that it is limited because the subset data does not cover enough diversities. Another possibility of small improvements is the limitation of transfer learning, that is, if the source data or the task for pretrain doesn't adequately represent the target domain, this mismatch will make the model performance not ideal enough. Further investigation needs to be run to determine if this data mismatch limitation has a decisive impact in our case.

The model complexity also has an influence on the training results. With too simple parameters and too few dimensions(neuron units), a model will find it difficult to express complicated relationships between features and to learn any complicated hypothesis. With too complex models, overfitting can easily happen in the training stage.

With more time, it would be worth investigating the impact on the model behavior from encoder learning rate for fine-tuning, decoder learning rate, attention linear dimension and so on.

4 Final Model

Based on the analysis above, we will run two models for conclusion:

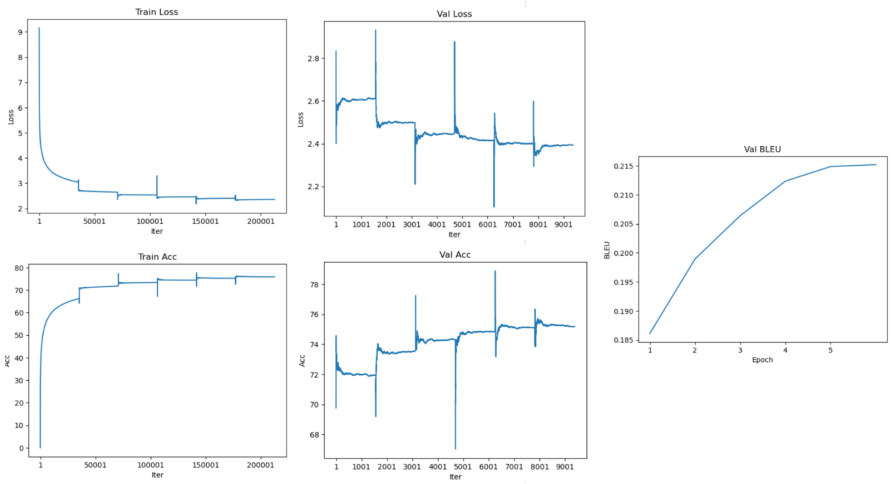
1. Fine-tuned encoder, decoder with RNN, Batch size 16, embed and decode dimension 512, with Adam optimizer and gradient clip threshold set to 8;
2. Fine-tuned encoder, decoder with RNN and Attention, Batch size 32, embed and decode dimension 512, with Adam optimizer and gradient clip threshold set to 8.

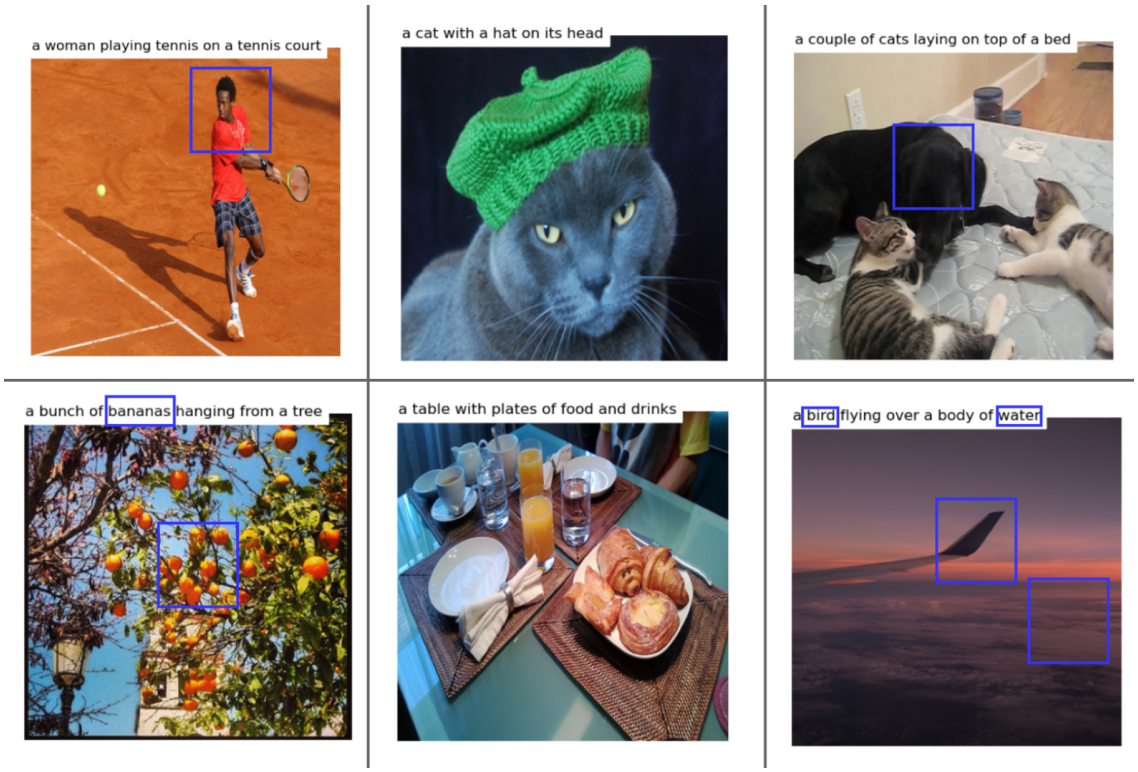
The gradient threshold was increased from 5 to 8 to allow more aggressive updates on the model now that the dataset is much larger. Accordingly, we will double the best batch size we have chosen from

Section 3, to prevent small batches updating the model too often. This also helps to avoid the risk that too little diversity is exposed to the model per step, since now the data amount is greater.

The original plan is to run each model for 15 epochs with early stopping and weight decaying based on contiguous epochs that have no improvements from the best performing scores during the training process. Due to the limitation of time and computation source, we only run limited epochs.

4.1 Decoder with RNN



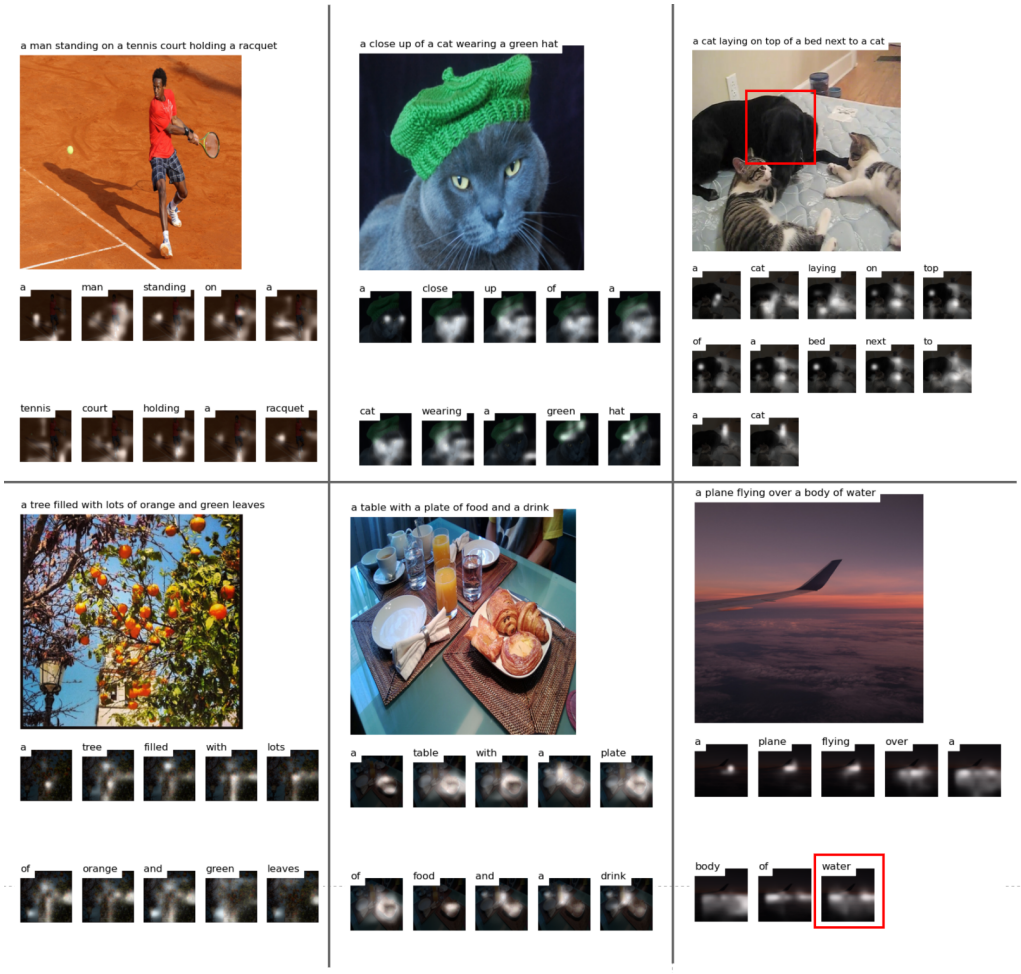


As can seen from the diagrams above, the model has not finished convergence, more epochs need to be run to get out of the current underfitting scenario. In total, **6 epochs** are run. The best performance is with the 6th epoch. The scores on the validation set is: **LOSS - 2.393, TOP-5 ACCURACY - 75.175, BLEU-4 - 0.21520169228251065**.

4.2 Decoder with RNN and Attention

Most of the training log data got lost due to interruption in the process, so no training log curves could be drawn. As far as remembered, the model keeps converging in the first 5 epochs. It reaches higher accuracy scores and higher BLEU scores within a shorter time compared to decoding with RNN only. However, the loss is higher in this model compared to Section 4.1. Therefore, it is safe to assume that small batch sizes do allow faster convergence, and learning on attention could reduce the model efficiency.

We run in total **7 epochs** on the full dataset. The best performance is on the 7th epoch. The scores achieved on the validation set for epoch 7 is: **LOSS - 3.291, TOP-5 ACCURACY - 76.559, BLEU-4 - 0.23356710818685755.** The BLEU score on the test set is



4.3 Summary

As the BLEU scores have a continuous increasing trend, we could safely say that the model is underfitting. It can be predicted that better semantic captions can be generated if we keep training the model for more epochs.

For the image-caption diagrams above, the rectangles circle out typical errors on features and on texts. Decoding with RNN has lower accuracy and BLEU scores. This phenomenon can also be reflected from the diagrams that many objects are misclassified.

Attention RNN has more correct classification over a diversity of object types. Wrong classification still happens when an object type is less represented in the training data, for example, the "cloud" is recognized as "water".

One typical limitation with RNN decoder is that the generated captions are missing levels of details. For example, the image with a cat wearing a hat. RNN decoder could express the contents of the image, but Attention RNN can fill in details such as "close up" and the color of the hat. The AttentionRNN decoder can generate synthetic captions with details, but not handle the order of phrases, preposition words, etc. It is worth trying some other attention calculation ways, or consider transformers for improvements. Both two decoders have difficulties to detect and describe all objects in an image, especially when multiple object types exist. An example is the image with cats and a dog. No description for the dog is generated by neither of the decoders. This can also be because the encoder fails to detect the object, or just the priority of words is not well learnt by the decoders.

5 Summary

On our final model, we run RNN decoder with 6 epochs and Attention RNN decoder with 7 epochs on the full data set.

With RNN decoder, the scores on the validation set is: **LOSS - 2.393, TOP-5 ACCURACY - 75.175, BLEU-4 - 0.21520169228251065**. BLEU score on the test set is: **0.269535**. With Attention RNN decoder, the scores on the validation set is: **LOSS - 3.291, TOP-5 ACCURACY - 76.559, BLEU-4 - 0.23356710818685755**. The BLEU score on the test set is: **0.29279**.

The BLEU scores have a continuous increasing trend throughout the epochs we run, therefore we could safely say that the model is underfitting. It can be predicted that better semantic captions can be generated if we keep training the model for more epochs.

For tuning parameters, we first run batches of experiments on a subset of data which has a much smaller amount of sample points.

Given the same batch size and learning rate, RNN decoder has a faster convergence compared to Attention RNN. However, the highest accuracy and BLEU scores that RNN with attention has achieved is higher than RNN only on this subset of data. This indicates that the learning on captions, especially grammatical wise, should not only rely on the feature of the images, but also the importance of words.

The attention mechanism has its advantages in this, although it can slow down the model efficiency, since there is more to learn.

The model complexity also has an influence on the training results. With too simple parameters and too few dimensions(neuron units), a model will find it difficult to express complicated relationships between features and to learn any complicated hypothesis. With too complex models, overfitting can easily happen in the training stage.

One typical limitation with RNN decoder is that the generated captions are missing levels of details. For example, the image with a cat wearing a hat. RNN decoder could express the contents of the image, but Attention RNN can fill in details such as "close up" and the color of the hat. The AttentionRNN decoder can generate synthetic captions with details, but not handle the order of phrases, preposition words, etc. It is worth trying some other attention calculation ways, or consider transformers for improvements. Both two decoders have difficulties to detect and describe all objects in an image, especially when multiple object types exist. An example is the image with cats and a dog. No description for the dog is generated by neither of the decoders. This can also be because the encoder fails to detect the object, or just the priority of words is not well learnt by the decoders.

6 References

[1] Image captioning https://huggingface.co/docs/transformers/main/tasks/image_captioning

[2] Coco Dataset https://cocodataset.org/#home

[3] caption_dataset http://cs.stanford.edu/people/karpathy/deepimagesent/caption_datasets.zip

[4] Deep Residual Learning for Image Recognition https://arxiv.org/abs/1512.03385

[5] Fine-tuning (deep learning) https://en.wikipedia.org/wiki/Fine-tuning_(deep_learning)

[6] Vinyals, O. et al. (2015) 'Show and tell: A neural image caption generator', 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [Preprint]. doi:10.1109/cvpr.2015.7298935.

[7] Xu K, Ba J, Kiros R, et al. Show, attend and tell: Neural image caption generation with visual attention[C]//International conference on machine learning. 2015: 2048-2057.

[8] Teacher Forcing — Summary Lesser know, but very important, training technique in NLP https://nishu-jain.medium.com/teacher-forcing-summary-f1bd790840ad

[9] What is teacher forcing? https://spotintelligence.com/2023/10/12/teacher-forcing-in-recurrent-neural-networks-rnns-an-advanced-concept-made-simple/

[10] A Hands-on Tutorial to Learn Attention Mechanism For Image Caption Generation in Python https://www.analyticsvidhya.com/blog/2020/11/attention-mechanism-for-caption-generation/

[11] a-PyTorch-Tutorial-to-Image-Captioning
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning/tree/master

[12] A Gentle Introduction to Calculating the BLEU Score for Text in Python
https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

[13] Evaluating models https://cloud.google.com/translate/automl/docs/evaluate

[14] Relation Between Learning Rate and Batch Size
https://www.baeldung.com/cs/learning-rate-batch-size#:~:text=The%20batch%20size%20affects%20some,and%20a%20good%20initial%20choice

[15] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[16] Understanding Gradient Clipping (and How It Can Fix Exploding Gradients Problem)
https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem

[17] Deep Learning Classification Model for English Translation Styles Introducing Attention Mechanism
https://www.hindawi.com/journals/mpe/2022/6798505/

[18] Transfer learning and fine-tuning https://www.tensorflow.org/tutorials/images/transfer_learning

[19] Overfitting, Variance, Bias and Model Complexity in Machine Learning
https://www.pico.net/kb/overfitting-variance-bias-and-model-complexity-in-machine-learning/#:~:text=However%2C%20a%20model%20that%20is,it%20tries%20to%20use%20them.