PA 4: Image Captioning

20222421 GSAI SimJaeYoon

0. Overview

This project is about Image Captioning. The goal of this project is to understand the concepts of Recurrent Neural Network(RNN) and Sequence to Sequence(SeqtoSeq) models by generating the image captions.

1. [Problem #1] Training and Testing the Basic Image Captioning Model

Problem 1 is basically a problem of training and testing an image capturing model. To this end, it is necessary to understand the concept of NIC model and image captioning and to proceed with learning using the code that actually implements it. The NIC model is an image captioning model released by Google researchers at CVPR in 2015. In order to implement a NIC model, we try to proceed with learning by git clone a model from the following github repositories.

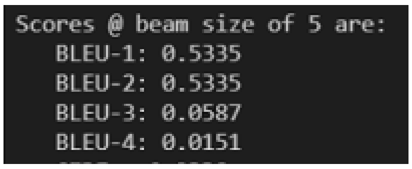
- https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning
- https://github.com/Renovamen/Image-Captioning

And from the conditions in problem 1, we will use ResNet101 for NIC model, and we will use Flickr8k dataset, a representative dataset of image capturing, as dataset.

- https://academictorrents.com/details/9dea07ba660a722ae1008c4c8afdd303b6f6e53b

■ **Fill out the following blanks in terms of Bilingual Evaluation Understudy (BLEU) score.**

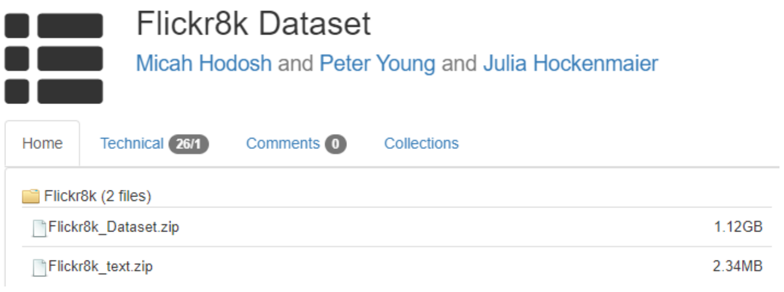
	BLEU-1	**BLEU-2**	**BLEU-3**	**BLEU-4**
NIC	0.5335	0.5335	0.0587	0.0151



```
Scores @ beam size of 5 are:
   BLEU-1: 0.5335
   BLEU-2: 0.5335
   BLEU-3: 0.0587
   BLEU-4: 0.0151
```

- **Try the efforts to improve the performances on your network models, such as your learning techniques or your network improvements that are not provided by the basic codes.**

In order to establish a learning model implementation environment, a basic source code was first downloaded from github. I tried to download the code from the reference and download the Flickr8k dataset separately.

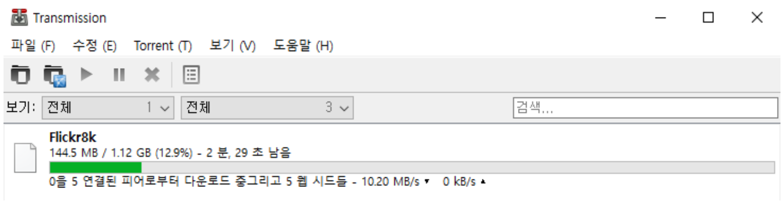


Flickr8k Dataset

Micah Hodosh and Peter Young and Julia Hockenmaier

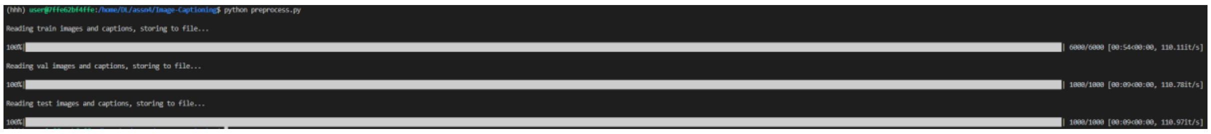
Flickr8k (2 files)	
Flickr8k_Dataset.zip	1.12GB
Flickr8k_text.zip	2.34MB

To download the Flickr8k dataset, the transmission-qt program was downloaded and the dataset was downloaded as follows.



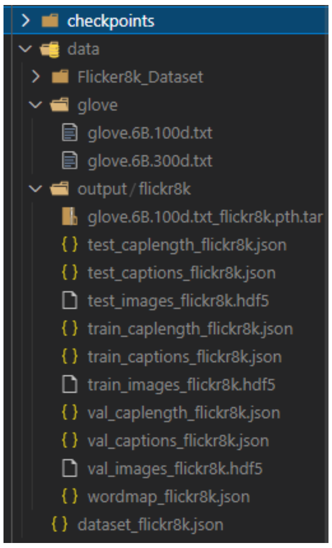
Flickr8k
144.5 MB / 1.12 GB (12.9%) - 2 분, 29 초 남음
0을 5 연결된 피어로부터 다운로드 중그리고 5 웹 시드들 - 10.20 MB/s ▾ 0 kB/s ▴

And "preprocess.py" was run to generate annotations.



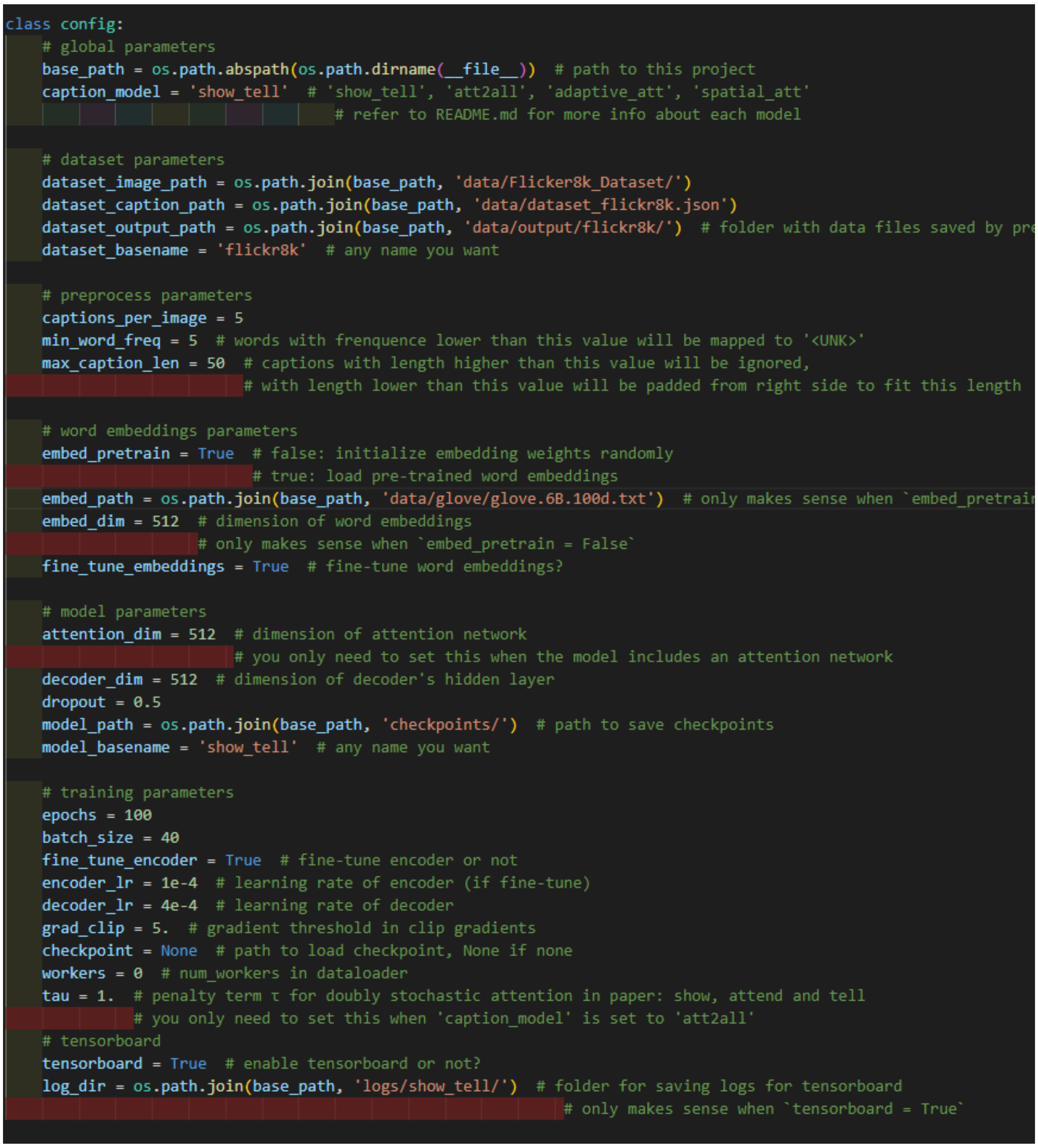
```
(hhh) user@7ffe62bf4ffe:/home/DL/assn4/Image-Captioning$ python preprocess.py
Reading train images and captions, storing to file...
100%|██████████████████████████████████████████████████| 6000/6000 [00:54<00:00, 110.11it/s]
Reading val images and captions, storing to file...
100%|██████████████████████████████████████████████████| 1000/1000 [00:09<00:00, 110.78it/s]
Reading test images and captions, storing to file...
100%|██████████████████████████████████████████████████| 1000/1000 [00:09<00:00, 110.97it/s]
```

After preprocessing for annotations, I downloaded the "glove.6B.300d.txt file" for pretraining. And as follows, checkpoints and data folders were created and prepared for learning.



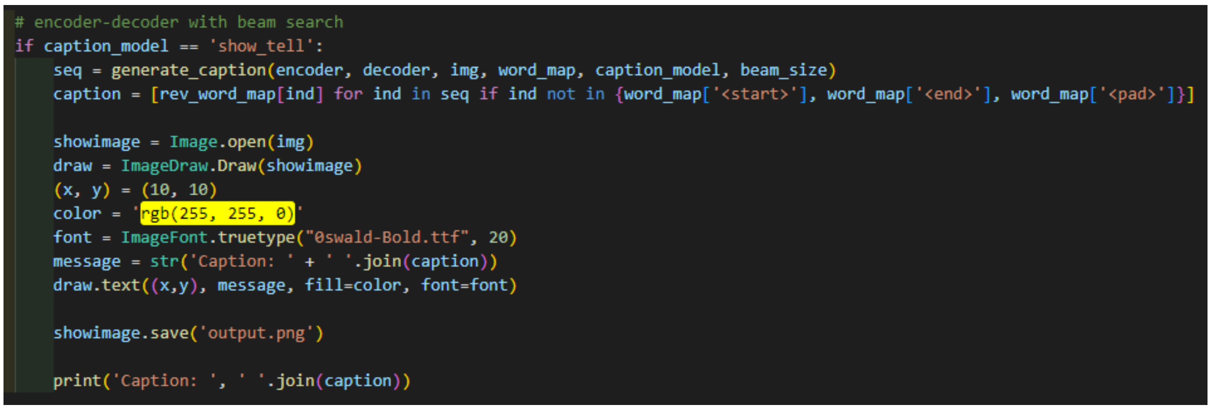
```
> checkpoints
v data
  > Flicker8k_Dataset
  v glove
      glove.6B.100d.txt
      glove.6B.300d.txt
  v output / flickr8k
      glove.6B.100d.txt_flickr8k.pth.tar
      test_caplength_flickr8k.json
      test_captions_flickr8k.json
      test_images_flickr8k.hdf5
      train_caplength_flickr8k.json
      train_captions_flickr8k.json
      train_images_flickr8k.hdf5
      val_caplength_flickr8k.json
      val_captions_flickr8k.json
      val_images_flickr8k.hdf5
      wordmap_flickr8k.json
    dataset_flickr8k.json
```

I modified "config.py" as follows for training. I changed the caption model to 'show_tell' for the first question. And the dataset parameter was set according to the desired path. Word embedding parameters set the path for pretraining. And the most important training parameters were tuned to achieve good performance.



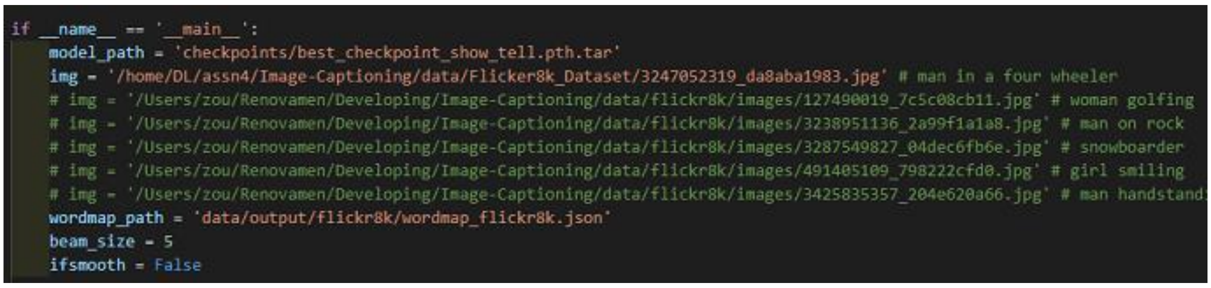
```python
class config:
    # global parameters
    base_path = os.path.abspath(os.path.dirname(__file__))  # path to this project
    caption_model = 'show_tell'  # 'show_tell', 'att2all', 'adaptive_att', 'spatial_att'
                                    # refer to README.md for more info about each model

    # dataset parameters
    dataset_image_path = os.path.join(base_path, 'data/Flicker8k_Dataset/')
    dataset_caption_path = os.path.join(base_path, 'data/dataset_flickr8k.json')
    dataset_output_path = os.path.join(base_path, 'data/output/flickr8k/')  # folder with data files saved by pre
    dataset_basename = 'flickr8k'  # any name you want

    # preprocess parameters
    captions_per_image = 5
    min_word_freq = 5  # words with frenquence lower than this value will be mapped to '<UNK>'
    max_caption_len = 50  # captions with length higher than this value will be ignored,
                          # with length lower than this value will be padded from right side to fit this length

    # word embeddings parameters
    embed_pretrain = True  # false: initialize embedding weights randomly
                           # true: load pre-trained word embeddings
    embed_path = os.path.join(base_path, 'data/glove/glove.6B.100d.txt')  # only makes sense when `embed_pretrai
    embed_dim = 512  # dimension of word embeddings
                     # only makes sense when `embed_pretrain = False`
    fine_tune_embeddings = True  # fine-tune word embeddings?

    # model parameters
    attention_dim = 512  # dimension of attention network
                         # you only need to set this when the model includes an attention network
    decoder_dim = 512  # dimension of decoder's hidden layer
    dropout = 0.5
    model_path = os.path.join(base_path, 'checkpoints/')  # path to save checkpoints
    model_basename = 'show_tell'  # any name you want

    # training parameters
    epochs = 100
    batch_size = 40
    fine_tune_encoder = True  # fine-tune encoder or not
    encoder_lr = 1e-4  # learning rate of encoder (if fine-tune)
    decoder_lr = 4e-4  # learning rate of decoder
    grad_clip = 5.  # gradient threshold in clip gradients
    checkpoint = None  # path to load checkpoint, None if none
    workers = 0  # num_workers in dataloader
    tau = 1.  # penalty term τ for doubly stochastic attention in paper: show, attend and tell
              # you only need to set this when 'caption_model' is set to 'att2all'
    # tensorboard
    tensorboard = True  # enable tensorboard or not?
    log_dir = os.path.join(base_path, 'logs/show_tell/')  # folder for saving logs for tensorboard
                                                          # only makes sense when `tensorboard = True`
```

And to visualize the results, I modified the "reference.py" file as follows. To this end, ImageDraw and ImageFont module were imported and used.



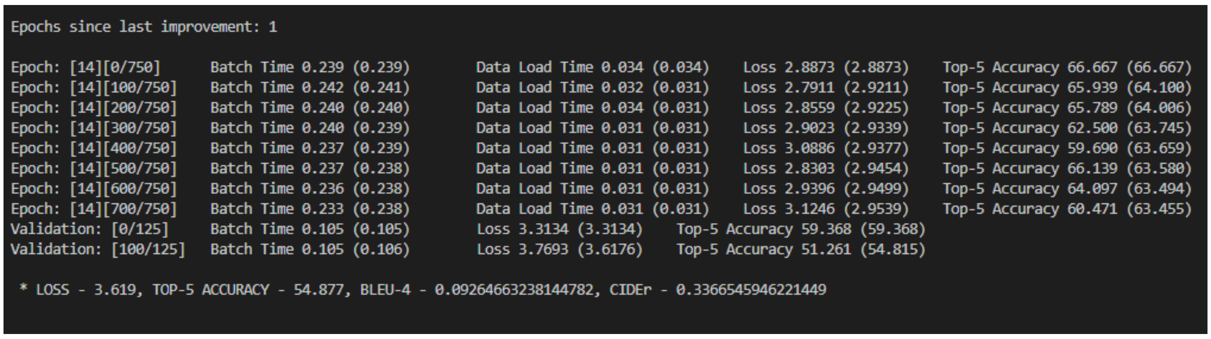
```python
# encoder-decoder with beam search
if caption_model == 'show_tell':
    seq = generate_caption(encoder, decoder, img, word_map, caption_model, beam_size)
    caption = [rev_word_map[ind] for ind in seq if ind not in {word_map['<start>'], word_map['<end>'], word_map['<pad>']}]

    showimage = Image.open(img)
    draw = ImageDraw.Draw(showimage)
    (x, y) = (10, 10)
    color = 'rgb(255, 255, 0)'
    font = ImageFont.truetype("Oswald-Bold.ttf", 20)
    message = str('Caption: ' + ' '.join(caption))
    draw.text((x,y), message, fill=color, font=font)

    showimage.save('output.png')

    print('Caption: ', ' '.join(caption))
```

And I changed the model and image path to generate the output image.



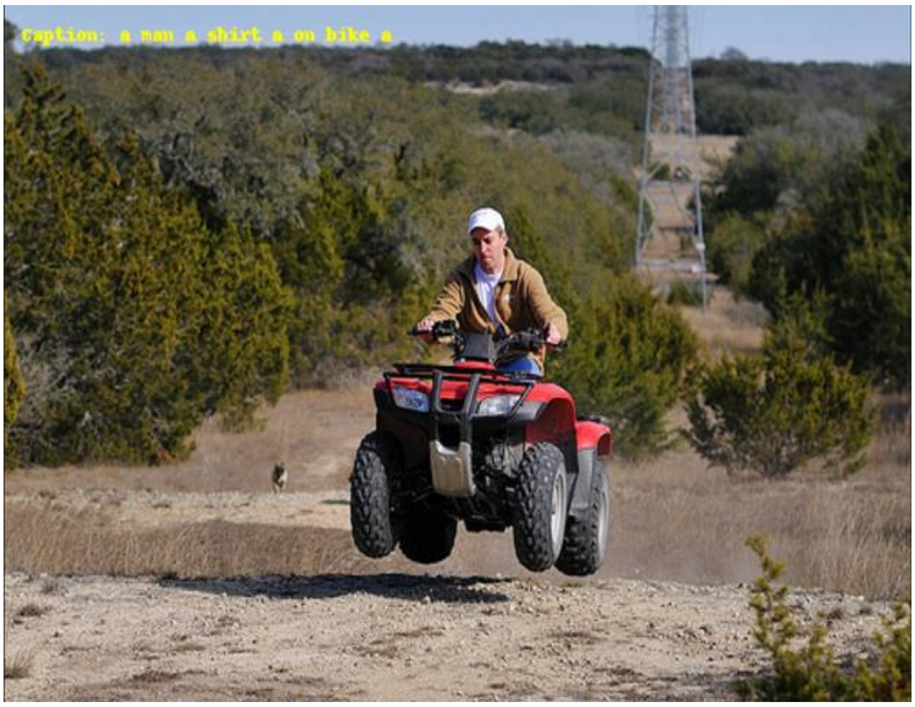
```python
if __name__ == '__main__':
    model_path = 'checkpoints/best_checkpoint_show_tell.pth.tar'
    img = '/home/DL/assn4/Image-Captioning/data/Flicker8k_Dataset/3247052319_da8aba1983.jpg' # man in a four wheeler
    # img = '/Users/zou/Renovamen/Developing/Image-Captioning/data/flickr8k/images/127490019_7c5c08cb11.jpg' # woman golfing
    # img = '/Users/zou/Renovamen/Developing/Image-Captioning/data/flickr8k/images/3238951136_2a99f1a1a8.jpg' # man on rock
    # img = '/Users/zou/Renovamen/Developing/Image-Captioning/data/flickr8k/images/3287549827_04dec6fb6e.jpg' # snowboarder
    # img = '/Users/zou/Renovamen/Developing/Image-Captioning/data/flickr8k/images/491405109_798222cfd0.jpg' # girl smiling
    # img = '/Users/zou/Renovamen/Developing/Image-Captioning/data/flickr8k/images/3425835357_204e620a66.jpg' # man handstand
    wordmap_path = 'data/output/flickr8k/wordmap_flickr8k.json'
    beam_size = 5
    ifsmooth = False
```

When "train.py" is executed, the learning proceeds as set in "config.py".

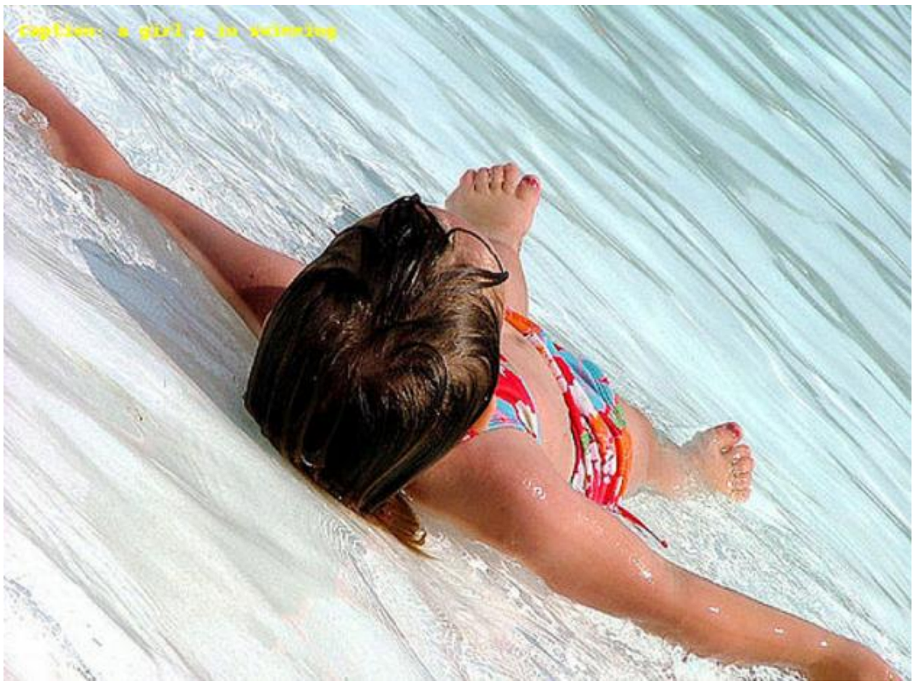


```
Epochs since last improvement: 1

Epoch: [14][0/750]      Batch Time 0.239 (0.239)       Data Load Time 0.034 (0.034)    Loss 2.8873 (2.8873)    Top-5 Accuracy 66.667 (66.667)
Epoch: [14][100/750]    Batch Time 0.242 (0.241)       Data Load Time 0.032 (0.031)    Loss 2.7911 (2.9211)    Top-5 Accuracy 65.939 (64.100)
Epoch: [14][200/750]    Batch Time 0.240 (0.240)       Data Load Time 0.034 (0.031)    Loss 2.8559 (2.9225)    Top-5 Accuracy 65.789 (64.006)
Epoch: [14][300/750]    Batch Time 0.240 (0.239)       Data Load Time 0.031 (0.031)    Loss 2.9023 (2.9339)    Top-5 Accuracy 62.500 (63.745)
Epoch: [14][400/750]    Batch Time 0.237 (0.239)       Data Load Time 0.031 (0.031)    Loss 3.0886 (2.9377)    Top-5 Accuracy 59.690 (63.659)
Epoch: [14][500/750]    Batch Time 0.237 (0.238)       Data Load Time 0.031 (0.031)    Loss 2.8303 (2.9454)    Top-5 Accuracy 66.139 (63.580)
Epoch: [14][600/750]    Batch Time 0.236 (0.238)       Data Load Time 0.031 (0.031)    Loss 2.9396 (2.9499)    Top-5 Accuracy 64.097 (63.494)
Epoch: [14][700/750]    Batch Time 0.233 (0.238)       Data Load Time 0.031 (0.031)    Loss 3.1246 (2.9539)    Top-5 Accuracy 60.471 (63.455)
Validation: [0/125]     Batch Time 0.105 (0.105)       Loss 3.3134 (3.3134)    Top-5 Accuracy 59.368 (59.368)
Validation: [100/125]   Batch Time 0.105 (0.106)       Loss 3.7693 (3.6176)    Top-5 Accuracy 51.261 (54.815)

 * LOSS - 3.619, TOP-5 ACCURACY - 54.877, BLEU-4 - 0.09264663238144782, CIDEr - 0.3366545946221449
```

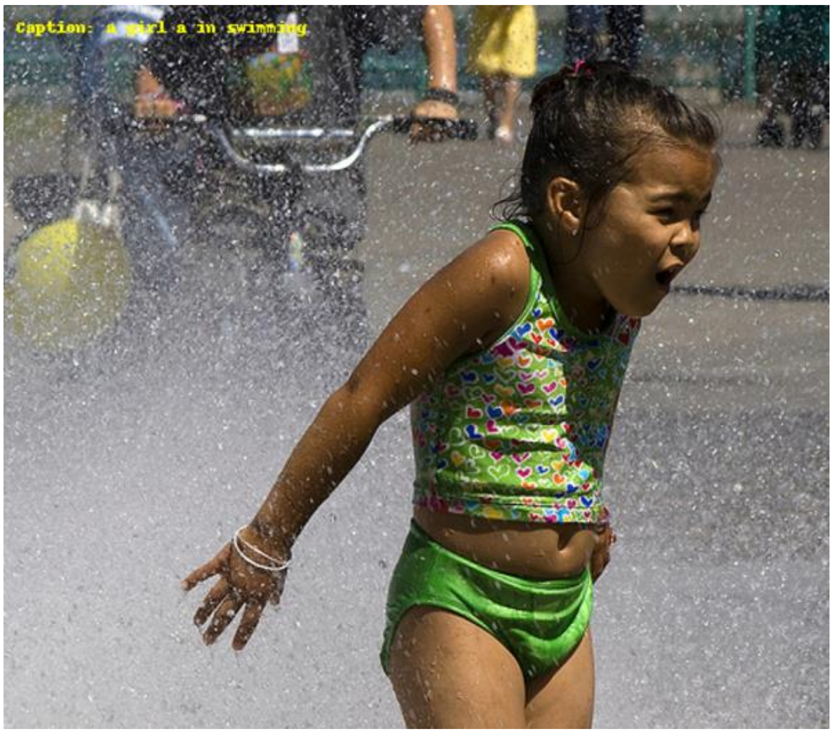
■ **Some result images including generated captions.**



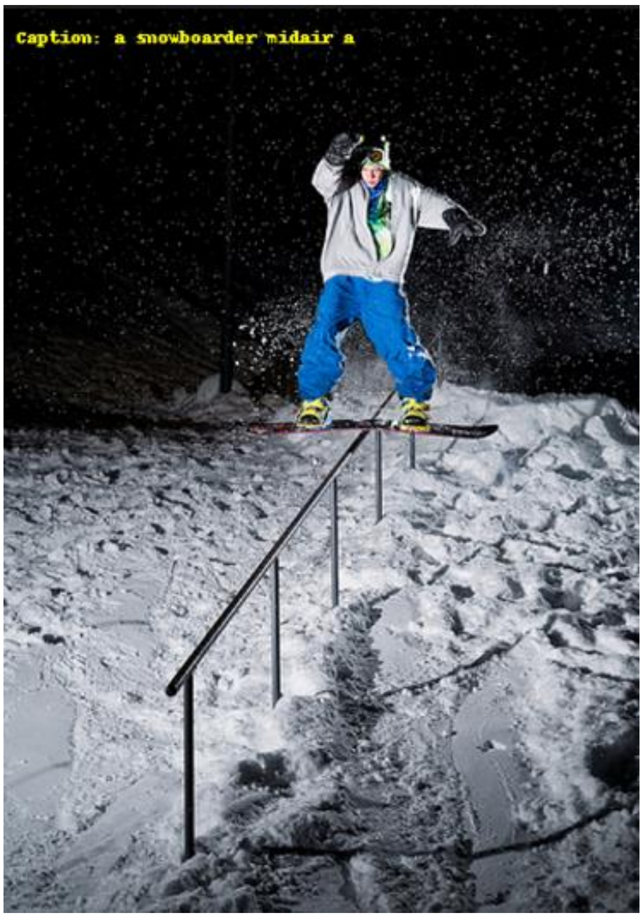
Caption: a man a shirt a on bike a



Caption: a girl a in swimming



Caption: a girl a in swimming



Caption: a snowboarder midair a

■ What did you learn through this problem #1.

Concept of image captioning

Image captioning is the task of describing the content of an image in words. This task lies at the intersection of computer vision and natural language processing. Most image captioning systems use an encoder-decoder framework, where an input image is encoded into an intermediate representation of the information in the image, and then decoded into a descriptive text sequence.

Concept of BLEU Score

Score Calculation in BLEU

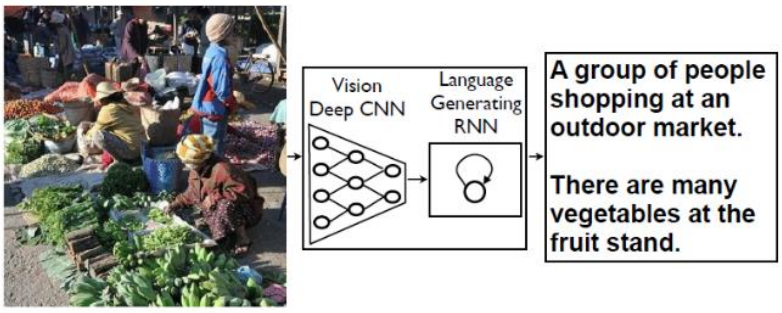
Unigram precision $\mathrm{P} = \frac{m}{w_t}$

Brevity penalty $p = \begin{cases} 1 & \text{if } c > r \\ e^{\left(1-\frac{r}{c}\right)} & \text{if } c \leq r \end{cases}$

$$\text{BLEU} = p.\, e^{\sum_{n=1}^{N} \left(\frac{1}{N} * \log Pn\right)}$$

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. A perfect match results in a score of 1, whereas a perfect mismatch results in a score of 0. BLEU may not be the perfect way, but it has several advantages. It can be used regardless of language, and calculation speed is fast. Unlike PPL, BLEU means higher performance is better. BLEU counts the word as correct if it is included in any one of the references.

Neural Image Caption (NIC)

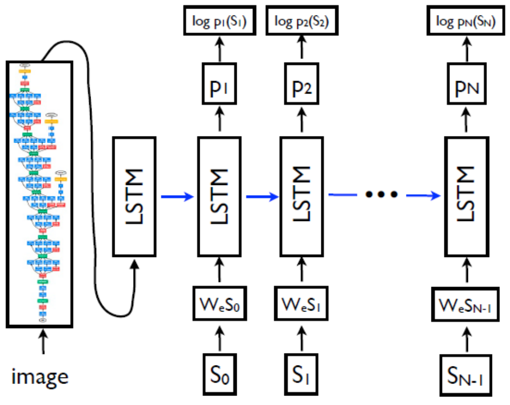

Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

Neural Image Caption (NIC) : CNN + RNN end-to-end Network

Given the image as above, features extracted by pre-trained CNNs are used as inputs to the (Encoding) Decoder, RNNs, to generate sentences describing the image. In the existing machine translation, the goal is to maximize the following probability, which is converted to maximizing the probability of a sense when an image is inserted.

$$\theta^\star = \arg\max_\theta \sum_{(I,S)} \log p(S|I;\theta)$$

For RNNs to learn well, LSTM was used, which is known to be able to solve vanishing or expanding gradients. LSTM performs learning by adjusting memory cell to three gates (input, get, output). As multiplication by chain rule in the backpropagtion used in the existing RNN is converted into addition, it not only solves the vanishing or expanding gradient, but also has the advantage of storing longer memory than RNN. The following figure shows how NIC was learned in the paper.



Neural Image Caption (NIC) Network Architecture

Greedy Search / Beam Search of NLP

Greedy search will simply take the highest probability word at each position in the sequence and predict that in the output sequence. Choosing just one candidate at a step might be optimal at the current spot in the sequence, but as we move through the rest of the full sentence, it might turn out to be worse than we thought, given we couldn't see later predicted positions. As we'll see later and you can probably predict, as our outputs become longer the greedy search algorithm begins to struggle.

The beam search algorithm selects multiple tokens for a position in a given sequence based on conditional probability. The algorithm can take any number of N best alternatives through a hyperparameter know as Beam width. In greedy search we simply took the best word for each position in the sequence, where here we broaden our search or "width" to include other words that might fit better.

Greedy search looks at each position in the output sequence in isolation. A word is decided based on highest probability and we continue moving down the rest of the sentence, not going back to earlier ones. With Beam search, we also take the N best output sequences and look at the current preceding words and the probabilities compared to the current position we are decoding in the sequence. Let's walk through an example to see the steps we must take to use beam search effectively.

Summary

While solving Problem 1, I learned about the image captioning method. We learned the basic structure and implementation method of NIC model, and we also learned the encoder and decoder structure. And I learned how to train NIC model through actual programming. I learned how to use BLEU score as an evaluation method of Image captioning and calculate and implement it. In addition, I learned the difference between Greedy search and Beam search of NLP.

- **Discuss about the experimental results, network architecture, and training techniques.**

As shown in the image above, the generated caption is not clearly represented in a perfect sentence format. In other words, the accuracy of captioning is poor. Since network architecture in NIC basic model matches the entire image with caption, learning performance may vary depending on the accuracy of labeling in captioning for learning. Like the object detection task of the previous task, if there are other image situations that are not labeled in one image, there is a problem that the performance of learning is degraded. In order to solve this problem, an attention mechanism may be used thereafter.

2. [Problem #2] Training and Testing the Image Captioning Model using Hard Attention Mechanism

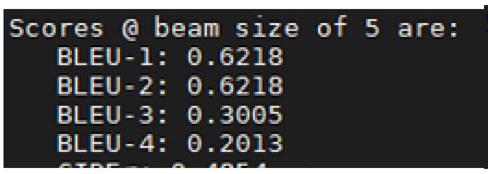
Problem 2 is to understand the hard attention mechanism among the attention techniques and to use an image captioning model using it. In order to understand the Hard Attention concept, it is necessary to understand the paper "Show, attend and tell: Neural image capture generation with visual attention" published in 2015. Therefore, it is necessary to implement a captioning model using hard attention using a soft attention-based model. In order to implement hard attention, we try to proceed with learning by git clone models from the following github repositories.

- https://github.com/Renovamen/Image-Captioning
- GitHub - sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning: Show, Attend, and Tell | a PyTorch Tutorial to Image Captioning

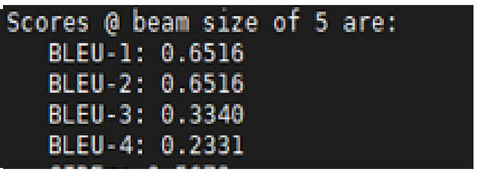
And from the conditions in problem 2, we will use VGG19 network and ResNet101 network for hard attention, and we will use Flickr8k dataset for dataset.

■ **Fill out the following blanks in terms of the BLEU score.**

	BLEU-1	**BLEU-2**	**BLEU-3**	**BLEU-4**
Soft Attention with VGG19	0.6218	0.6218	0.3005	0.2013
Soft Attention with ResNet101	0.6516	0.6516	0.3340	0.2331



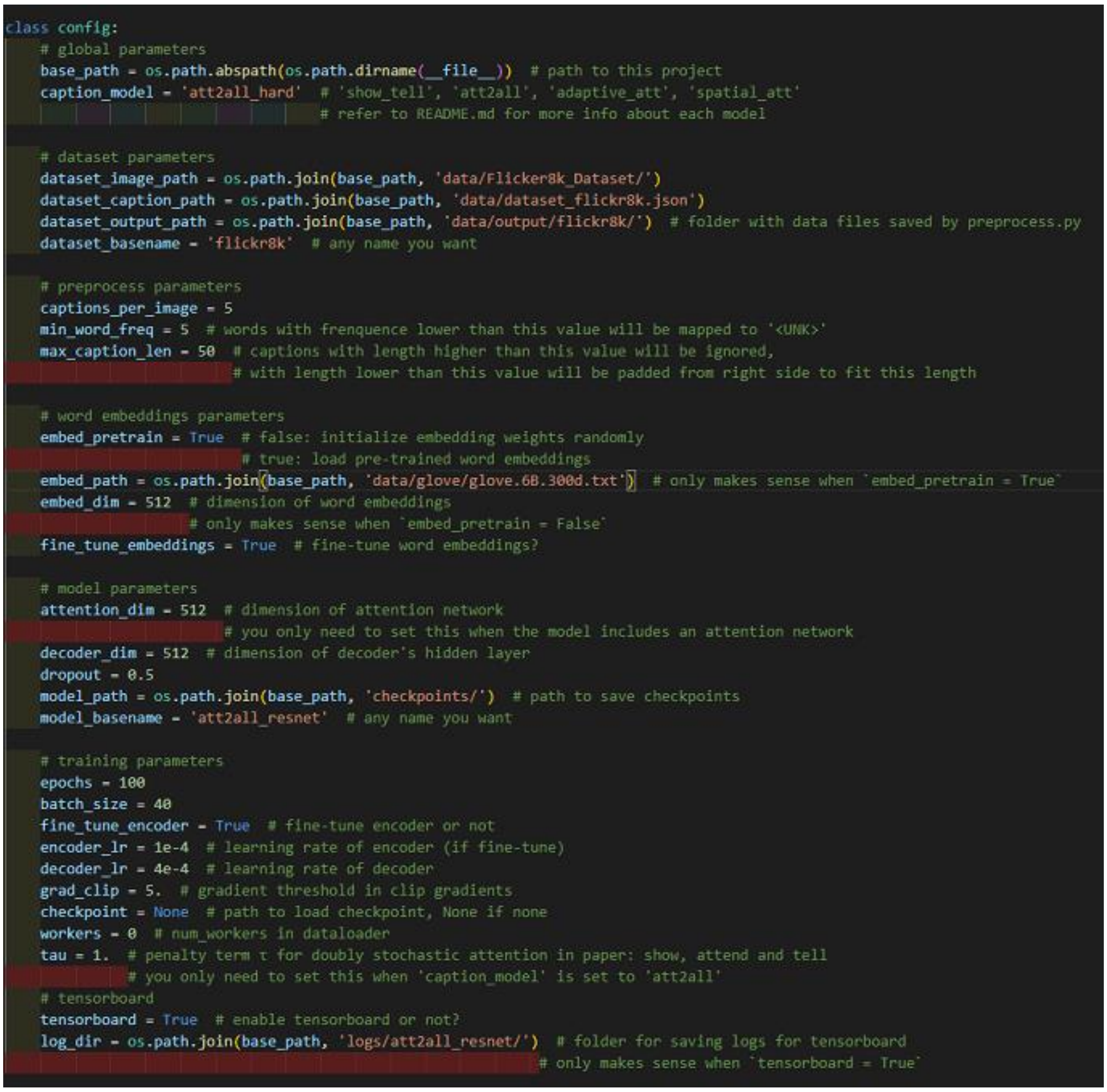
```
Scores @ beam size of 5 are:
   BLEU-1: 0.6218
   BLEU-2: 0.6218
   BLEU-3: 0.3005
   BLEU-4: 0.2013
```



```
Scores @ beam size of 5 are:
   BLEU-1: 0.6516
   BLEU-2: 0.6516
   BLEU-3: 0.3340
   BLEU-4: 0.2331
```

■ **Try the efforts to improve the performances on your network models, such as your learning techniques or your network improvements that are not provided by the basic codes.**

This time, it is a matter of using hard attention. So, I modified the file "config.py" as follows. Basically, there is a reset model, so it is necessary to add the VGG model to the encoder and decoder later.

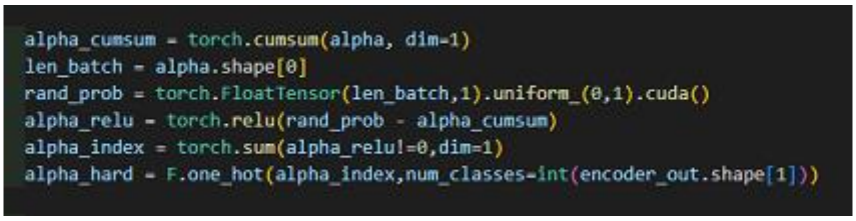


```python
class config:
    # global parameters
    base_path = os.path.abspath(os.path.dirname(__file__))  # path to this project
    caption_model = 'att2all_hard'  # 'show_tell', 'att2all', 'adaptive_att', 'spatial_att'
                                    # refer to README.md for more info about each model

    # dataset parameters
    dataset_image_path = os.path.join(base_path, 'data/Flicker8k_Dataset/')
    dataset_caption_path = os.path.join(base_path, 'data/dataset_flickr8k.json')
    dataset_output_path = os.path.join(base_path, 'data/output/flickr8k/')  # folder with data files saved by preprocess.py
    dataset_basename = 'flickr8k'  # any name you want

    # preprocess parameters
    captions_per_image = 5
    min_word_freq = 5  # words with frenquence lower than this value will be mapped to '<UNK>'
    max_caption_len = 50  # captions with length higher than this value will be ignored,
                          # with length lower than this value will be padded from right side to fit this length

    # word embeddings parameters
    embed_pretrain = True  # false: initialize embedding weights randomly
                           # true: load pre-trained word embeddings
    embed_path = os.path.join(base_path, 'data/glove/glove.6B.300d.txt')  # only makes sense when `embed_pretrain = True`
    embed_dim = 512  # dimension of word embeddings
                     # only makes sense when `embed_pretrain = False`
    fine_tune_embeddings = True  # fine-tune word embeddings?

    # model parameters
    attention_dim = 512  # dimension of attention network
                         # you only need to set this when the model includes an attention network
    decoder_dim = 512  # dimension of decoder's hidden layer
    dropout = 0.5
    model_path = os.path.join(base_path, 'checkpoints/')  # path to save checkpoints
    model_basename = 'att2all_resnet'  # any name you want

    # training parameters
    epochs = 100
    batch_size = 40
    fine_tune_encoder = True  # fine-tune encoder or not
    encoder_lr = 1e-4  # learning rate of encoder (if fine-tune)
    decoder_lr = 4e-4  # learning rate of decoder
    grad_clip = 5.  # gradient threshold in clip gradients
    checkpoint = None  # path to load checkpoint, None if none
    workers = 0  # num_workers in dataloader
    tau = 1.  # penalty term τ for doubly stochastic attention in paper: show, attend and tell
              # you only need to set this when 'caption_model' is set to 'att2all'
    # tensorboard
    tensorboard = True  # enable tensorboard or not?
    log_dir = os.path.join(base_path, 'logs/att2all_resnet/')  # folder for saving logs for tensorboard
                                                               # only makes sense when `tensorboard = True`
```

Since the source code att2all used soft attention basically, I have look at this code to make my attention hard. I find the code from

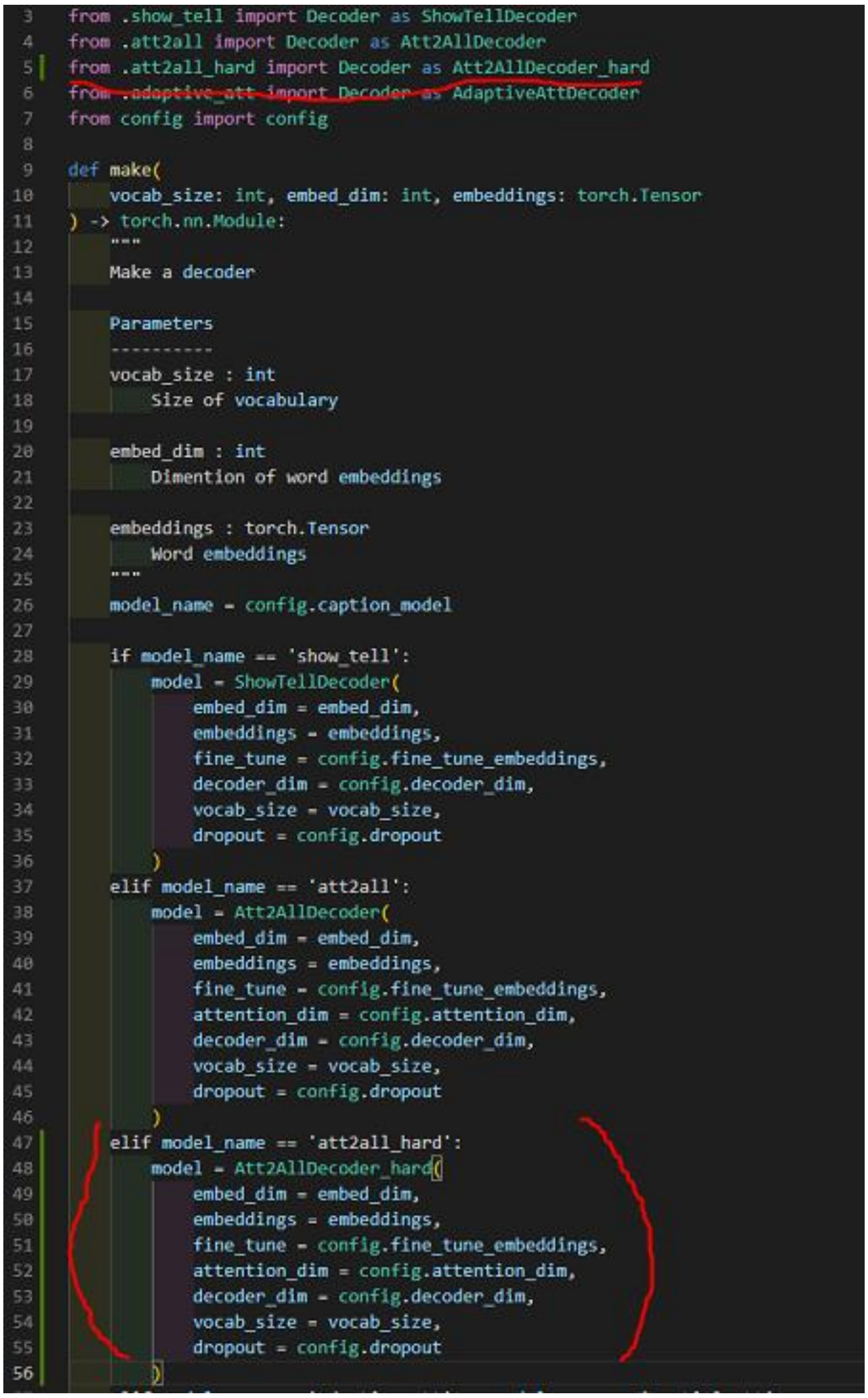
- https://github.com/Kyushik/Attention/blob/master/Hard_Attention_MNIST.ipynb

And put this code named "att2all_hard.py" in the decoder folder and also change the __init__.py in the decoder folder. Here, Monte-Carlo sampling for alpha sampling for hard attrition was separately implemented. The implementation method was implemented by converting it into pytorch by referring to the existing code.



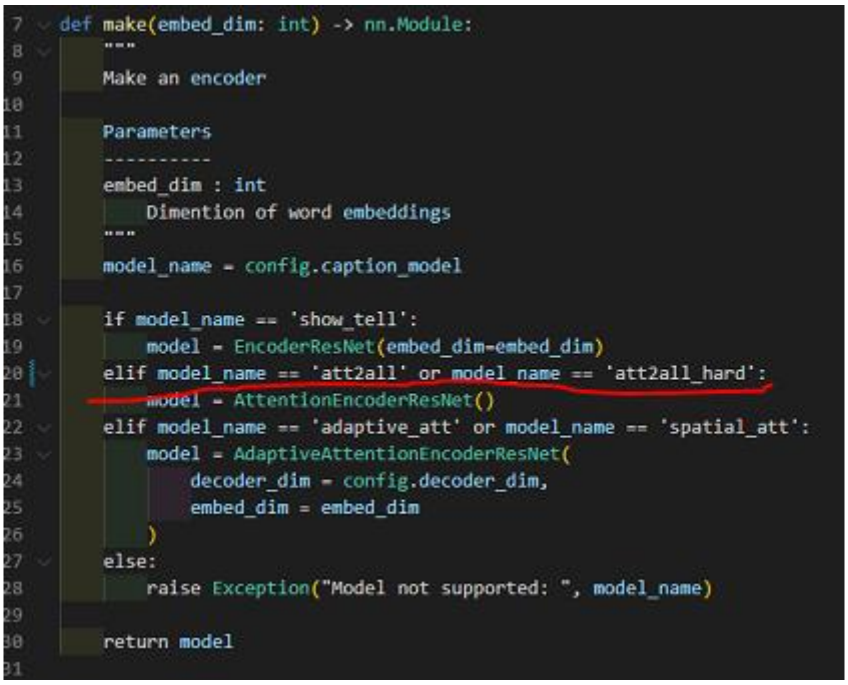
```python
alpha_cumsum = torch.cumsum(alpha, dim=1)
len_batch = alpha.shape[0]
rand_prob = torch.FloatTensor(len_batch,1).uniform_(0,1).cuda()
alpha_relu = torch.relu(rand_prob - alpha_cumsum)
alpha_index = torch.sum(alpha_relu!=0,dim=1)
alpha_hard = F.one_hot(alpha_index,num_classes=int(encoder_out.shape[1]))
```

And now, in order to apply att2all_hard model, we have to make some changes in the basic code. First, in models/decoder/__init__.py, the code was written to use att2all_hard as follows.



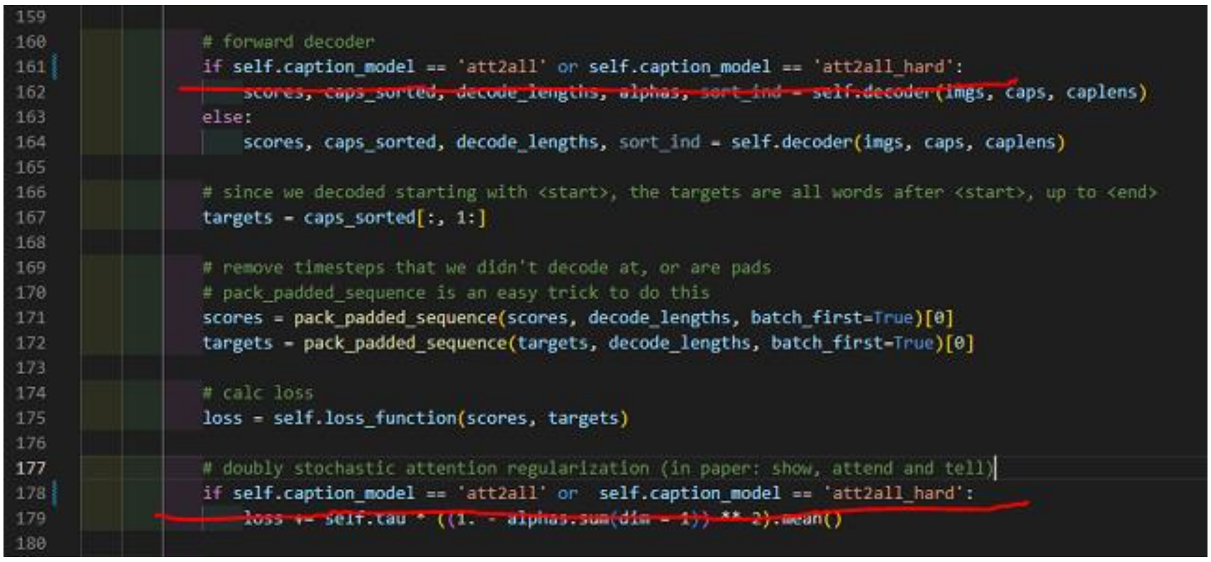
```python
from .show_tell import Decoder as ShowTellDecoder
from .att2all import Decoder as Att2AllDecoder
from .att2all_hard import Decoder as Att2AllDecoder_hard
from .adaptive_att import Decoder as AdaptiveAttDecoder
from config import config

def make(
    vocab_size: int, embed_dim: int, embeddings: torch.Tensor
) -> torch.nn.Module:
    """
    Make a decoder

    Parameters
    ----------
    vocab_size : int
        Size of vocabulary

    embed_dim : int
        Dimention of word embeddings

    embeddings : torch.Tensor
        Word embeddings
    """
    model_name = config.caption_model

    if model_name == 'show_tell':
        model = ShowTellDecoder(
            embed_dim = embed_dim,
            embeddings = embeddings,
            fine_tune = config.fine_tune_embeddings,
            decoder_dim = config.decoder_dim,
            vocab_size = vocab_size,
            dropout = config.dropout
        )
    elif model_name == 'att2all':
        model = Att2AllDecoder(
            embed_dim = embed_dim,
            embeddings = embeddings,
            fine_tune = config.fine_tune_embeddings,
            attention_dim = config.attention_dim,
            decoder_dim = config.decoder_dim,
            vocab_size = vocab_size,
            dropout = config.dropout
        )
    elif model_name == 'att2all_hard':
        model = Att2AllDecoder_hard(
            embed_dim = embed_dim,
            embeddings = embeddings,
            fine_tune = config.fine_tune_embeddings,
            attention_dim = config.attention_dim,
            decoder_dim = config.decoder_dim,
            vocab_size = vocab_size,
            dropout = config.dropout
        )
```

And model/encoder/__init__.py also added as follows.

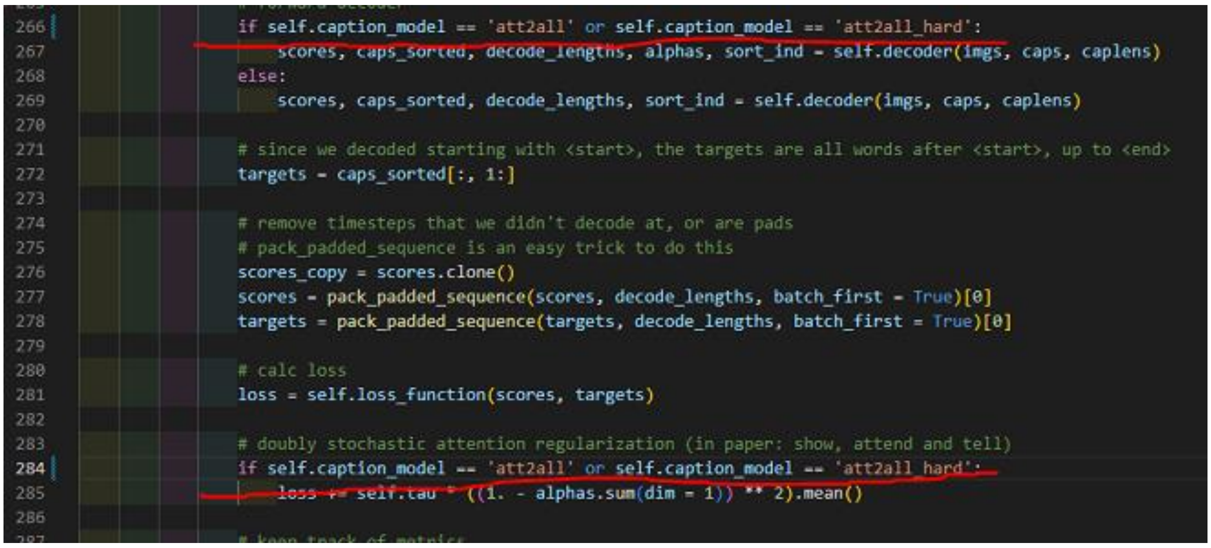


```python
def make(embed_dim: int) -> nn.Module:
    """
    Make an encoder

    Parameters
    ----------
    embed_dim : int
        Dimention of word embeddings
    """
    model_name = config.caption_model

    if model_name == 'show_tell':
        model = EncoderResNet(embed_dim=embed_dim)
    elif model_name == 'att2all' or model_name == 'att2all_hard':
        model = AttentionEncoderResNet()
    elif model_name == 'adaptive_att' or model_name == 'spatial_att':
        model = AdaptiveAttentionEncoderResNet(
            decoder_dim = config.decoder_dim,
            embed_dim = embed_dim
        )
    else:
        raise Exception("Model not supported: ", model_name)

    return model
```

In addition, the trainer/trainer.py added as follows.



```python
        # forward decoder
        if self.caption_model == 'att2all' or self.caption_model == 'att2all_hard':
            scores, caps_sorted, decode_lengths, alphas, sort_ind = self.decoder(imgs, caps, caplens)
        else:
            scores, caps_sorted, decode_lengths, sort_ind = self.decoder(imgs, caps, caplens)

        # since we decoded starting with <start>, the targets are all words after <start>, up to <end>
        targets = caps_sorted[:, 1:]

        # remove timesteps that we didn't decode at, or are pads
        # pack_padded_sequence is an easy trick to do this
        scores = pack_padded_sequence(scores, decode_lengths, batch_first=True)[0]
        targets = pack_padded_sequence(targets, decode_lengths, batch_first=True)[0]

        # calc loss
        loss = self.loss_function(scores, targets)

        # doubly stochastic attention regularization (in paper: show, attend and tell)
        if self.caption_model == 'att2all' or  self.caption_model == 'att2all_hard':
            loss += self.tau * ((1. - alphas.sum(dim = 1)) ** 2).mean()
```



```python
        if self.caption_model == 'att2all' or self.caption_model == 'att2all_hard':
            scores, caps_sorted, decode_lengths, alphas, sort_ind = self.decoder(imgs, caps, caplens)
        else:
            scores, caps_sorted, decode_lengths, sort_ind = self.decoder(imgs, caps, caplens)

        # since we decoded starting with <start>, the targets are all words after <start>, up to <end>
        targets = caps_sorted[:, 1:]

        # remove timesteps that we didn't decode at, or are pads
        # pack_padded_sequence is an easy trick to do this
        scores_copy = scores.clone()
        scores = pack_padded_sequence(scores, decode_lengths, batch_first = True)[0]
        targets = pack_padded_sequence(targets, decode_lengths, batch_first = True)[0]

        # calc loss
        loss = self.loss_function(scores, targets)

        # doubly stochastic attention regularization (in paper: show, attend and tell)
        if self.caption_model == 'att2all' or self.caption_model == 'att2all_hard':
            loss += self.tau * ((1. - alphas.sum(dim = 1)) ** 2).mean()
```

In order to use VGGNet19, models/encoder/vggnet19.py model code was generated as follows.

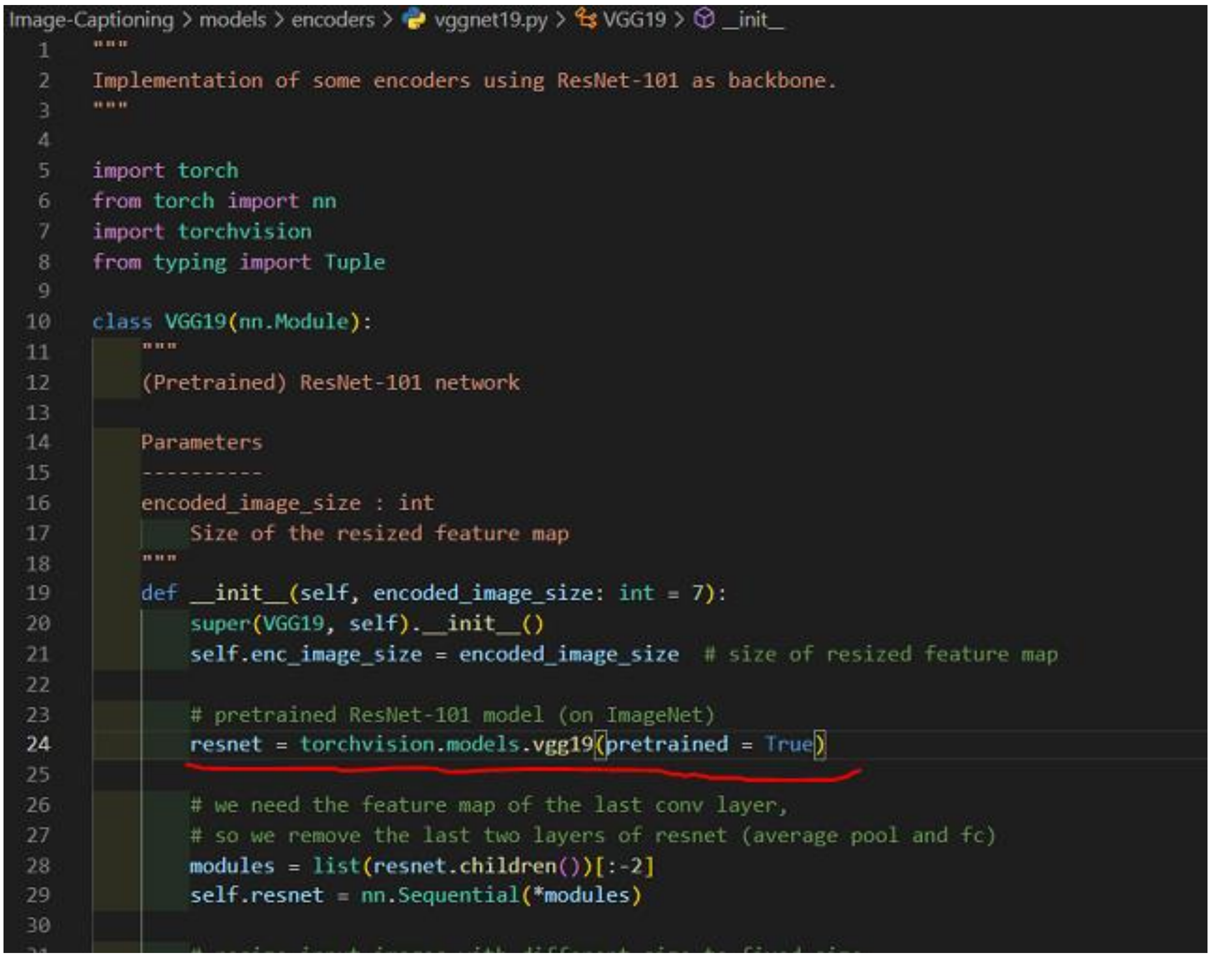
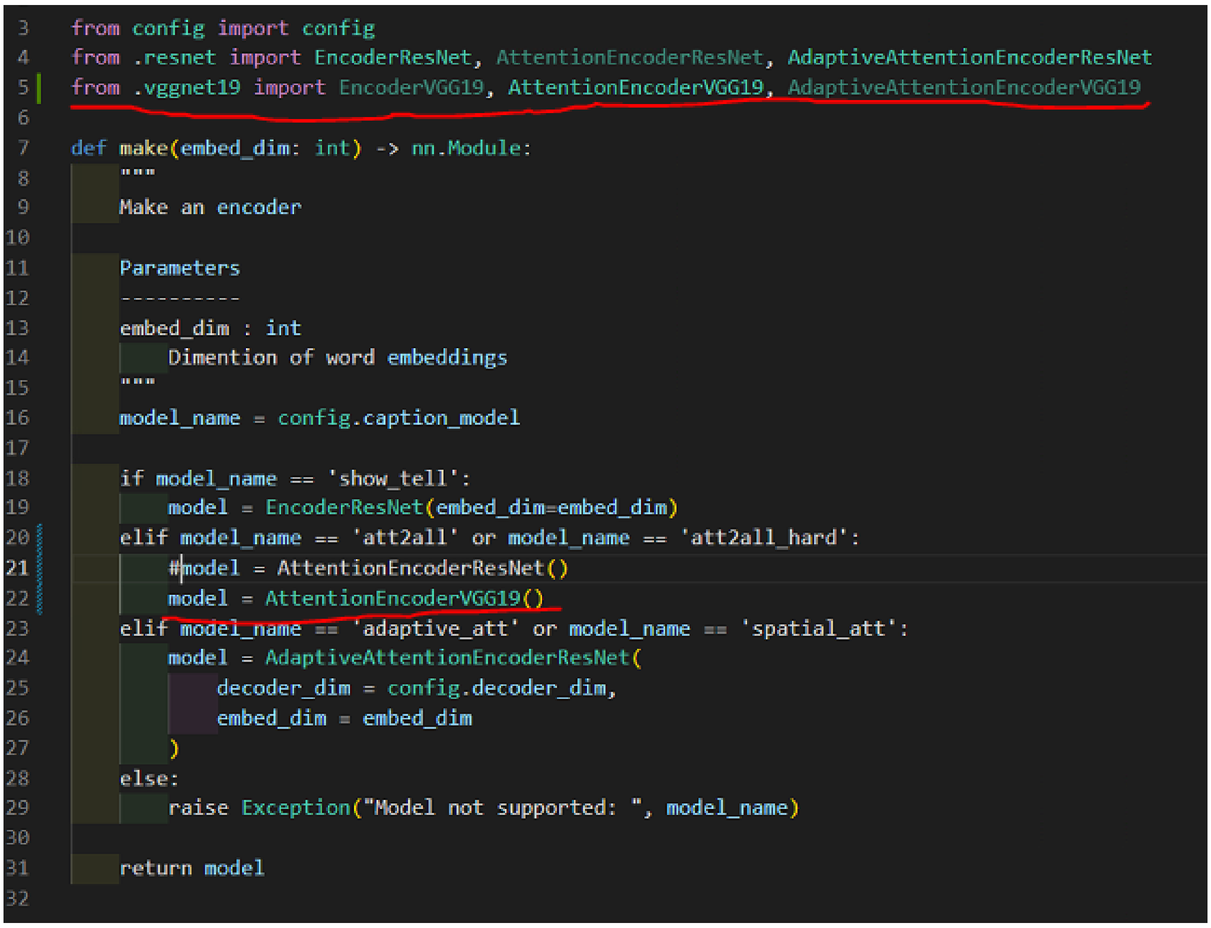


Image-Captioning > models > encoders > vggnet19.py > VGG19 > __init__

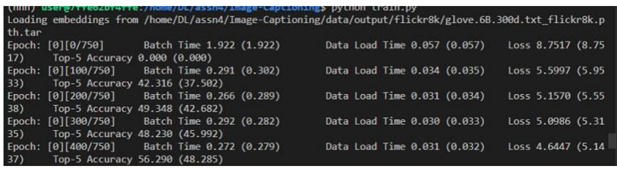
```python
"""
Implementation of some encoders using ResNet-101 as backbone.
"""

import torch
from torch import nn
import torchvision
from typing import Tuple

class VGG19(nn.Module):
    """
    (Pretrained) ResNet-101 network

    Parameters
    ----------
    encoded_image_size : int
        Size of the resized feature map
    """
    def __init__(self, encoded_image_size: int = 7):
        super(VGG19, self).__init__()
        self.enc_image_size = encoded_image_size  # size of resized feature map

        # pretrained ResNet-101 model (on ImageNet)
        resnet = torchvision.models.vgg19(pretrained = True)

        # we need the feature map of the last conv layer,
        # so we remove the last two layers of resnet (average pool and fc)
        modules = list(resnet.children())[:-2]
        self.resnet = nn.Sequential(*modules)
```

Similarly, models/encoder/__init__. py, it was added to as follows.

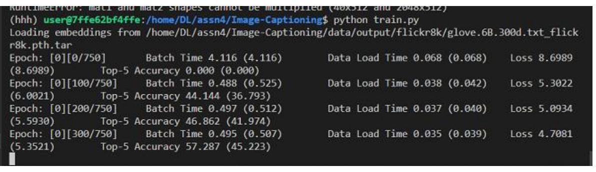


```python
from config import config
from .resnet import EncoderResNet, AttentionEncoderResNet, AdaptiveAttentionEncoderResNet
from .vggnet19 import EncoderVGG19, AttentionEncoderVGG19, AdaptiveAttentionEncoderVGG19

def make(embed_dim: int) -> nn.Module:
    """
    Make an encoder

    Parameters
    ----------
    embed_dim : int
        Dimention of word embeddings
    """
    model_name = config.caption_model

    if model_name == 'show_tell':
        model = EncoderResNet(embed_dim=embed_dim)
    elif model_name == 'att2all' or model_name == 'att2all_hard':
        #model = AttentionEncoderResNet()
        model = AttentionEncoderVGG19()
    elif model_name == 'adaptive_att' or model_name == 'spatial_att':
        model = AdaptiveAttentionEncoderResNet(
            decoder_dim = config.decoder_dim,
            embed_dim = embed_dim
        )
    else:
        raise Exception("Model not supported: ", model_name)

    return model

```

The att2all_hard caption model was trained as follows. ResNet on the left and VGGNet on the right.



```
(hhh) user@7ffe62bf4ffe:/home/DL/assn4/Image-Captioning$ python train.py
Loading embeddings from /home/DL/assn4/Image-Captioning/data/output/flickr8k/glove.6B.300d.txt_flickr8k.p
th.tar
Epoch: [0][0/750]       Batch Time 1.922 (1.922)        Data Load Time 0.057 (0.057)    Loss 8.7517 (8.75
17)     Top-5 Accuracy 0.000 (0.000)
Epoch: [0][100/750]     Batch Time 0.291 (0.302)        Data Load Time 0.034 (0.035)    Loss 5.5997 (5.95
33)     Top-5 Accuracy 42.316 (37.502)
Epoch: [0][200/750]     Batch Time 0.266 (0.289)        Data Load Time 0.031 (0.034)    Loss 5.1570 (5.55
38)     Top-5 Accuracy 49.348 (42.682)
Epoch: [0][300/750]     Batch Time 0.292 (0.282)        Data Load Time 0.030 (0.033)    Loss 5.0986 (5.31
35)     Top-5 Accuracy 48.230 (45.992)
Epoch: [0][400/750]     Batch Time 0.272 (0.279)        Data Load Time 0.031 (0.032)    Loss 4.6447 (5.14
37)     Top-5 Accuracy 56.290 (48.285)
```



```
RuntimeError: mat1 and mat2 shapes cannot be multiplied (40x512 and 2048x512)
(hhh) user@7ffe62bf4ffe:/home/DL/assn4/Image-Captioning$ python train.py
Loading embeddings from /home/DL/assn4/Image-Captioning/data/output/flickr8k/glove.6B.300d.txt_flick
r8k.pth.tar
Epoch: [0][0/750]       Batch Time 4.116 (4.116)        Data Load Time 0.068 (0.068)    Loss 8.6989
(8.6989)        Top-5 Accuracy 0.000 (0.000)
Epoch: [0][100/750]     Batch Time 0.488 (0.525)        Data Load Time 0.038 (0.042)    Loss 5.3022
(6.0021)        Top-5 Accuracy 44.144 (36.793)
Epoch: [0][200/750]     Batch Time 0.497 (0.512)        Data Load Time 0.037 (0.040)    Loss 5.0934
(5.5930)        Top-5 Accuracy 46.862 (41.974)
Epoch: [0][300/750]     Batch Time 0.495 (0.507)        Data Load Time 0.035 (0.039)    Loss 4.7081
(5.3521)        Top-5 Accuracy 57.287 (45.223)
```

- **Some result images similar with the Fig 1.**

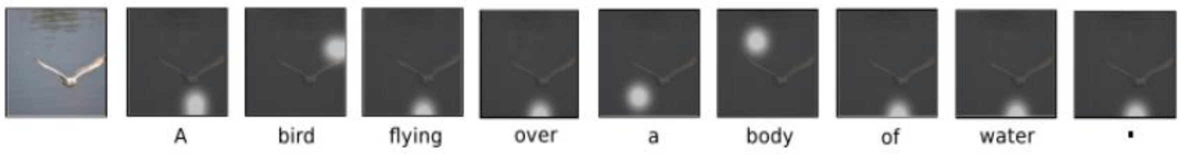
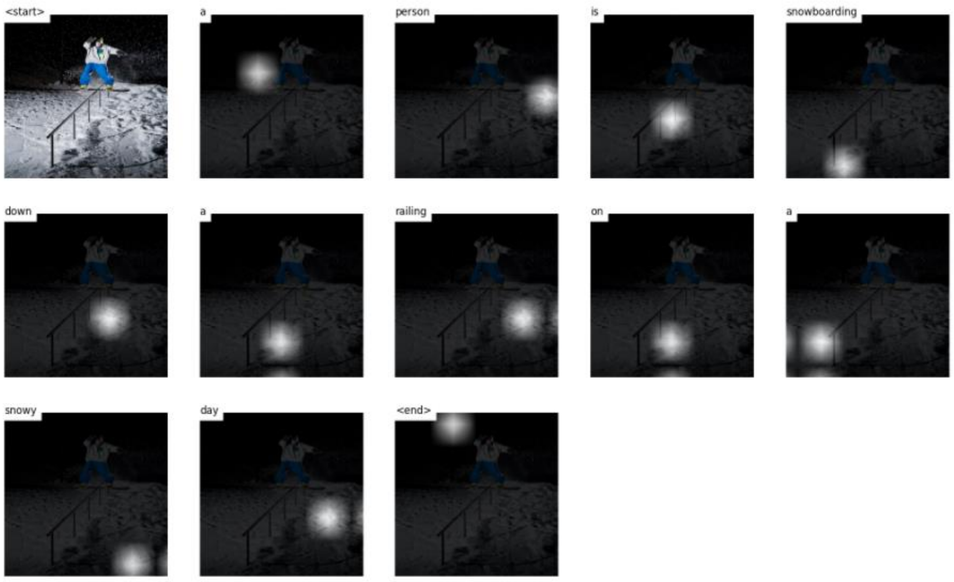
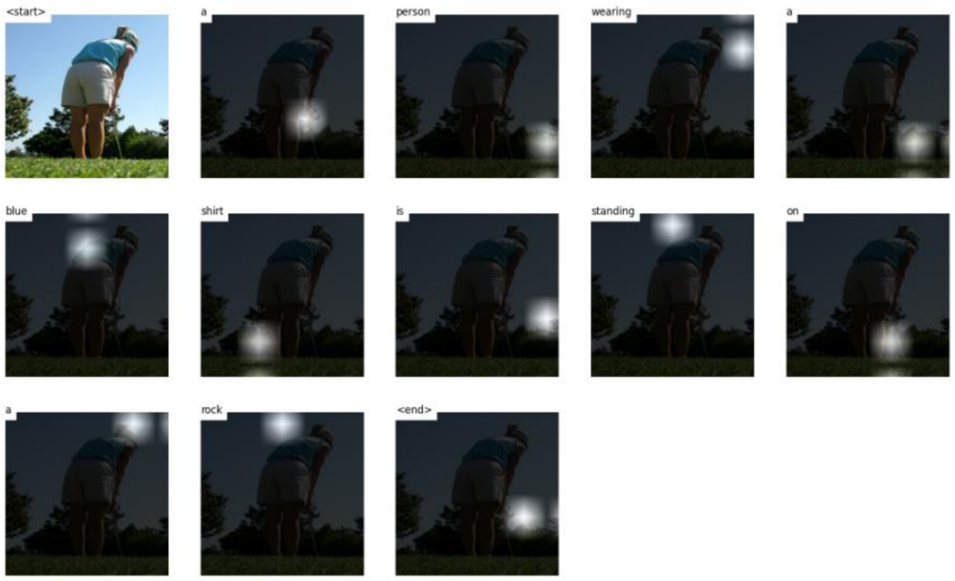


Figure 1. Example of the Hard Attention based Image Captioning result

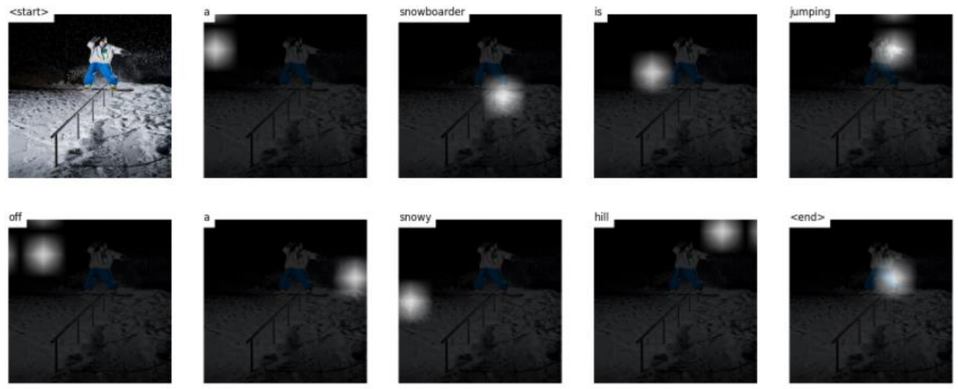
I used the image which is '3287549827_04dec6fb6e.jpg' and '127490019_7c5c08cb11.jpg' to generate hard attention image with ResNet and VGGNet.

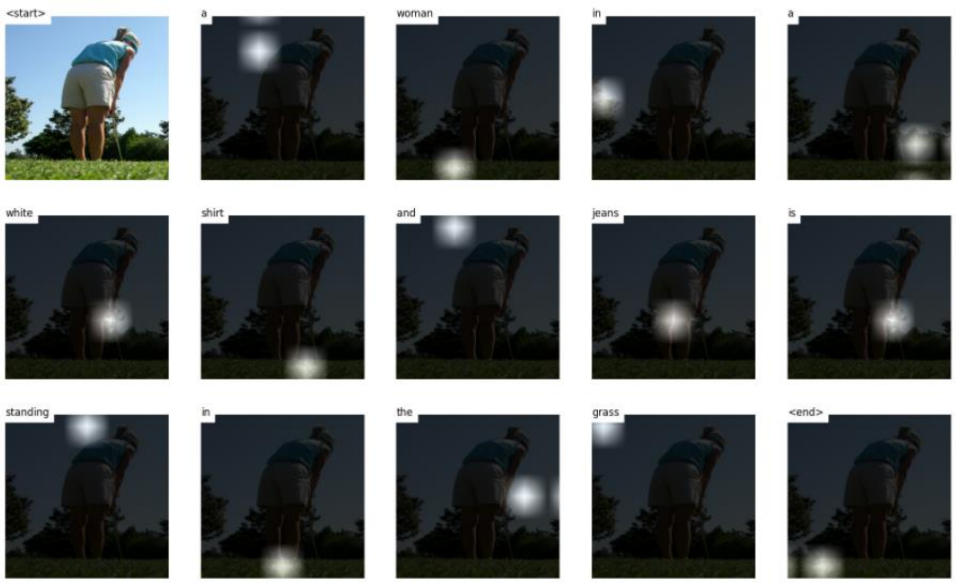
<resnet>


<start>
a
person
is
snowboarding
down
a
railing
on
a
snowy
day
<end>


<start>
a
person
wearing
a
blue
shirt
is
standing
on
a
rock
<end>

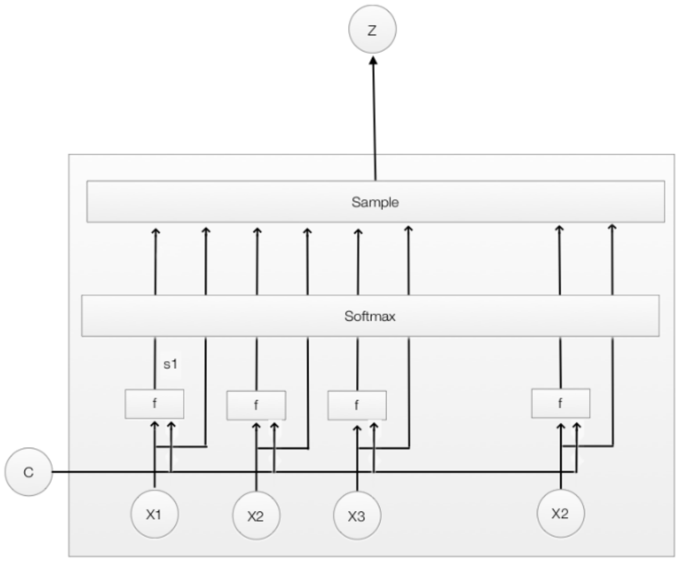
<vgg>


<start>
a
snowboarder
is
jumping
off
a
snowy
hill
<end>


<start>
a
woman
in
a
white
shirt
and
jeans
is
standing
in
the
grass
<end>

■ **What did you learn through this problem #2.**

Concept of Hard Attention



In soft attention, they compute a weight α for each x, and use it to calculate a weighted average for x as the LSTM input. α adds up to 1 which can be interpreted as the probability that x is the area that we should pay attention to. So instead of a weighted average, hard attention uses α as a sample rate to pick one x as the input to the LSTM. Hard attention replaces a deterministic method with a stochastic sampling model. To calculate the gradient descent correctly in the backpropagation, they perform samplings and average our results using the Monte Carlo method. Monte Carlo performs end-to-end episodes to compute an average for all sampling results.

Concept and Implementation of Captioning Accuracy Improvement Using Attention

For every timestamp that predicts the output word from the decoder during Attention, refer back to the entire input sentence from the encoder. At this time, not all inputs are referred to by the same weight, but when the decoder hands over the current timestamp, the encoder uses the softmax function to digitize and convey the degree to which it helps to infer the decoder, and the decoder uses this information. In other words, the decoder has a higher probability of predicting the output word more accurately because it delivers crop information on a specific area of the image and the decoder uses this crop result to predict it.

■ **Discuss about the experimental results, network architecture, and training techniques.**

As shown in the generated image, clearer capturing is possible when using Attention. Even when comparing the attentioned image with the caption word, it can be seen that it is relatively relevant.

Moreover, I have noticed that the caption from VGGNet was better in the snowboard man image. So, I concluded that using the attention is better than not using it.

3. [Problem #3] Training and Testing the Image Captioning Model using Soft Attention Mechanism

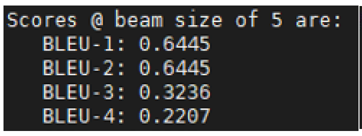
Problem 3 is to understand the soft attraction mechanism among the attention techniques and to use an image captioning model using it. In order to understand the concept of soft attention, it is necessary to understand the paper "Show, attend and tell: Neural image capture generation with visual attrition" published in 2015 as in problem 2. So what we need to do this time is to understand the soft attention mechanism and implement an image captioning model based on soft attention. In order to implement soft attention, we try to proceed with learning by git clone models from the following github repositories.

- https://github.com/Renovamen/Image-Captioning
- GitHub - sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning: Show, Attend, and Tell | a PyTorch Tutorial to Image Captioning

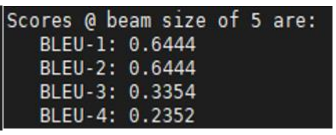
And from the conditions in problem 3, we will use VGG19 network and ResNet101 network for soft attention, and we will use Flickr8k dataset for dataset.

■ **Fill out the following blanks in terms of the BLEU score.**

	BLEU-1	**BLEU-2**	**BLEU-3**	**BLEU-4**
Soft Attention with VGG19	0.6445	0.6445	0.3236	0.2207
Soft Attention with ResNet101	0.6444	0.6444	0.3354	0.2352



```
Scores @ beam size of 5 are:
    BLEU-1: 0.6445
    BLEU-2: 0.6445
    BLEU-3: 0.3236
    BLEU-4: 0.2207
```



```
Scores @ beam size of 5 are:
    BLEU-1: 0.6444
    BLEU-2: 0.6444
    BLEU-3: 0.3354
    BLEU-4: 0.2352
```

■ **Try the efforts to improve the performances on your network models, such as your learning techniques or your network improvements that are not provided by the basic codes.**

The source code provided the soft attention with the name of att2all. So, I have used that source code and changed some configuration (like batch size and the naming function) in the “config.py” like I had mentioned in the problem 1 and 2. If "att2all_hard" was used as the caption model in problem 2, then "att2all" was used in problem 3. The other content is the same as problem 2.

- **Some result images similar with the Fig 2.**

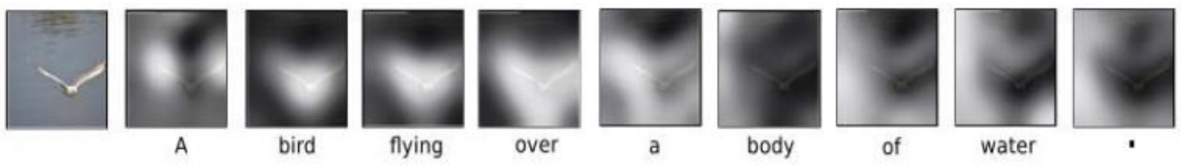
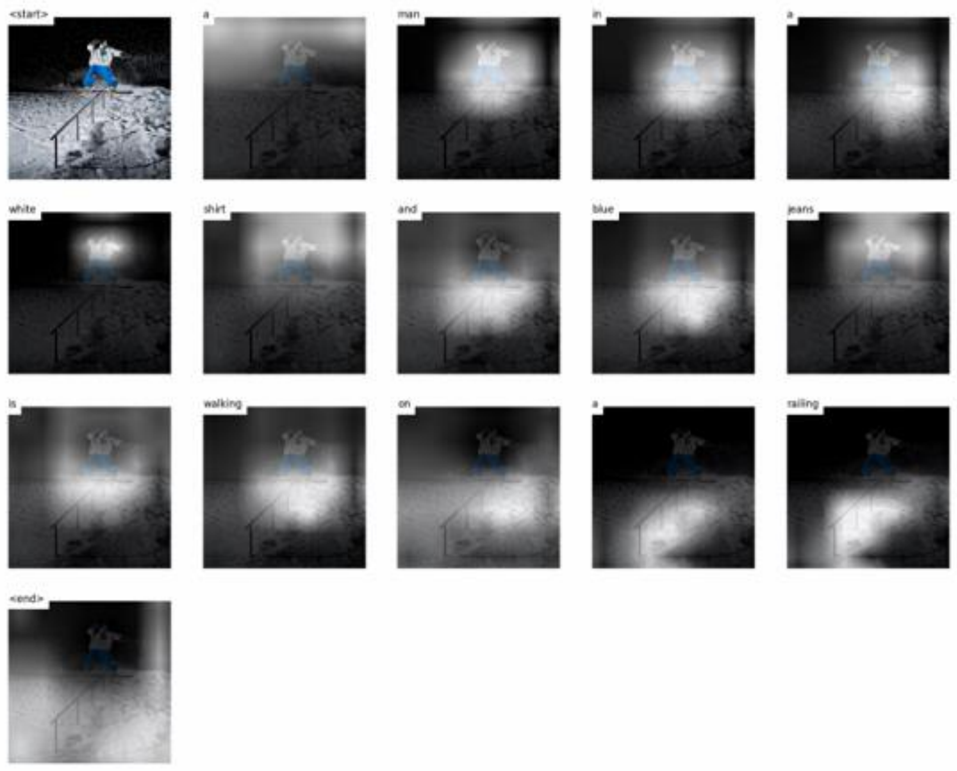
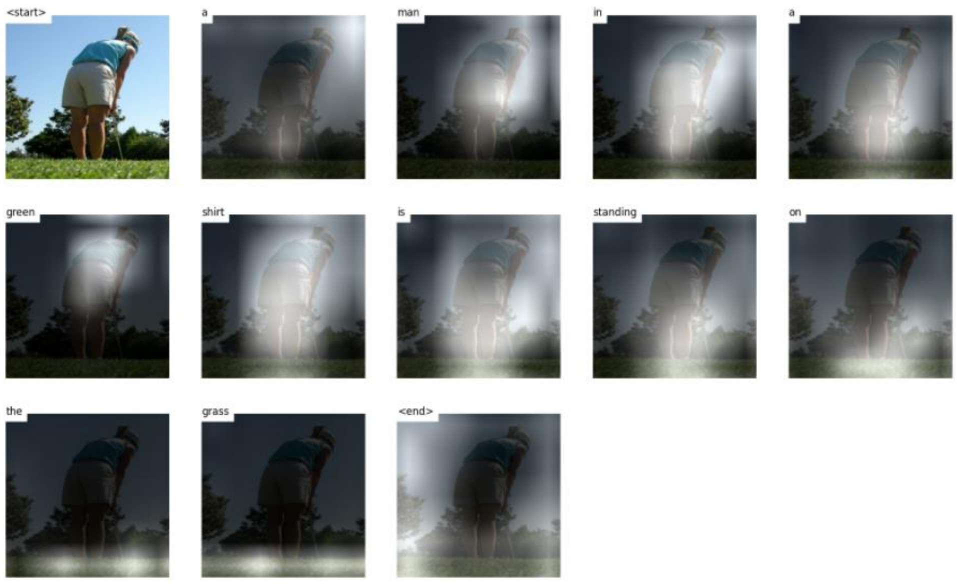


Figure 2. Example of the Soft Attention based Image Captioning result

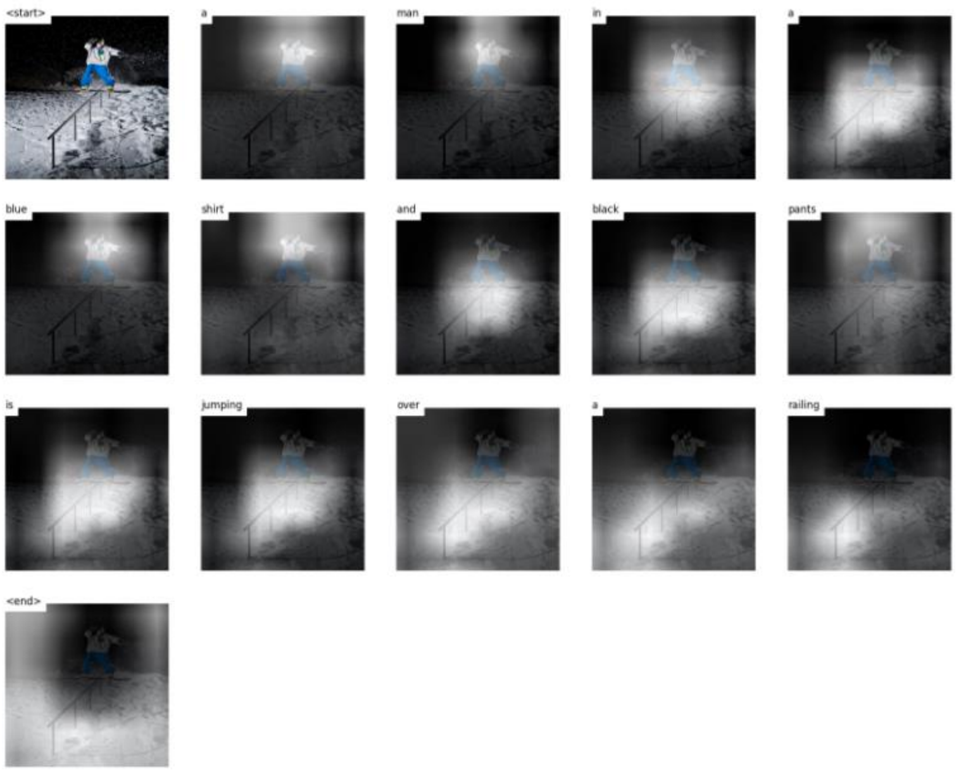
I used the image which is ‘3287549827_04dec6fb6e.jpg’ and ‘127490019_7c5c08cb11.jpg’ to generate soft attention image with ResNet and VGGNet.

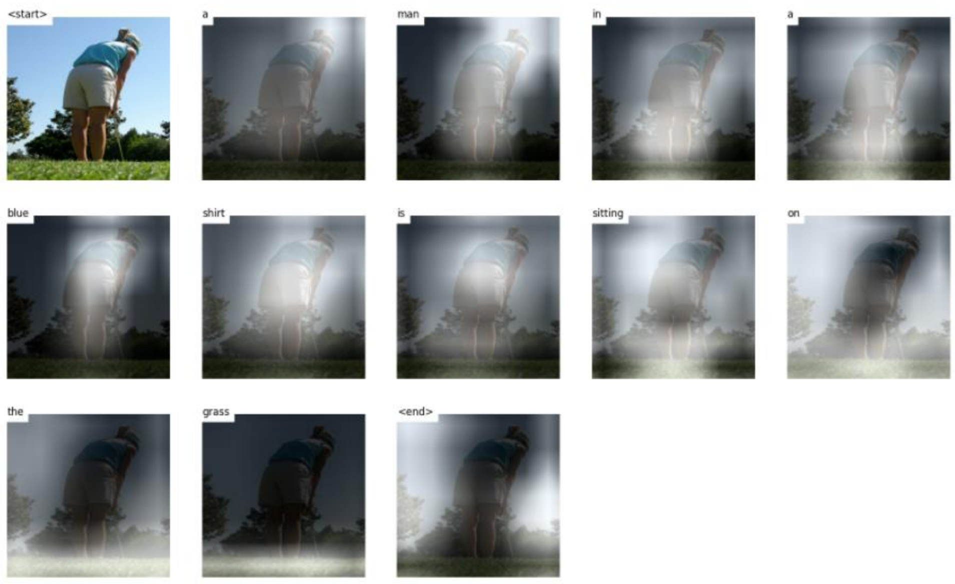
<resnet>




<start>
a
man
in
a
green
shirt
is
standing
on
the
grass
<end>

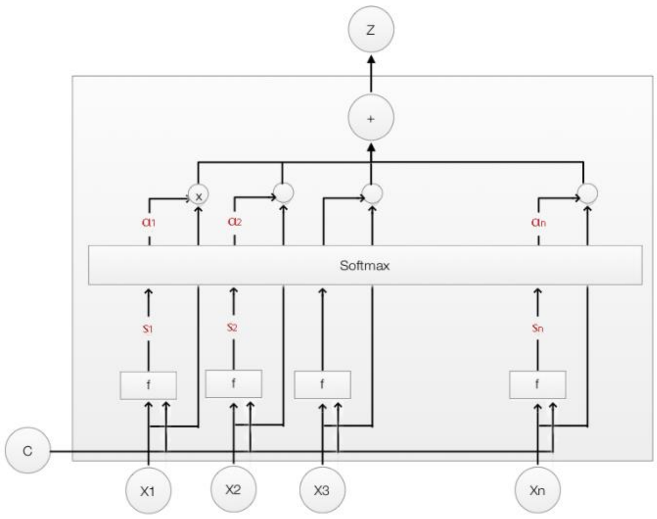
<vgg>


<start>
a
man
in
a
blue
shirt
and
black
pants
is
jumping
over
a
railing
<end>



■ **What did you learn through this problem #3.**

Concept of Soft Attention



In soft attention, instead of using the image x as an input to the LSTM, we input weighted image features accounted for attention. Soft attention discredits irrelevant areas by multiply the corresponding features map with a low weight. Accordingly, high attention area keeps the original value while low attention areas get closer to 0.

$$s_i = \tanh(W_c C + W_x X_i) = \tanh(W_c h_{t-1} + W_x x_i)$$

$$\alpha_i = softmax(s_1, s_2, \dots, s_i, \dots)$$

$$Z = \sum_i \alpha_i x_i$$

■ **Discuss about the experimental results, network architecture, and training techniques.**

Like hard attention, it can be seen that soft attention is learned with weights of some areas of the image. And unlike hard attention, attention is made for a wide range of areas. As we can see in the above example about the snowboarding man image, the caption is so nice I think. I was quite surprised about the result because this was the first time, I have ever used the attention model. The caption of the soft attention is so best and I think this is because soft attention used continuous attention whereas hard attention used 1 or 0. Through this task, it was good to apply the attention technique to the image capturing task. I think it was a good experience for me, majoring in a different field than the NLP field.