Deep Learning Project
Team Member & Contribution:
Yang Lei (1002361): Implement algorithms to build Encoder & Decoder, program sampling code for GUI and write report
Gao YiMing (1002185):Test model and write report
Zang Xueqing (1002180):Build GUI and test model
Submission Date: 21 April 2019

Performance on the validation set

At the beginning, we decided to use GRU instead of using LSTM to train the decode model.

Performance of using GRU:

Epoch 1: Step [2000/6333], Loss: 998.8727 , Accuracy: 0.4646, BLEU score: 0.0738, Time: 2.91 mins target sentense: this is an image of a train at a station . <end> pred sentense: a is a old of a train on a station <end> <end>
Epoch 2: Step [2000/6333], Loss: 911.6866 , Accuracy: 0.4698, BLEU score: 0.0789, Time: 2.92 mins target sentense: a couple of elephants out in the field having a snack <end> pred sentense: a group of elephants standing in a water <end> a drink .
Epoch 3: Step [4000/6333], Loss: 960.7203 , Accuracy: 0.4867, BLEU score: 0.0846, Time: 6.21 mins target sentense: a plate of <unk> , a pizza and garlic bread on a table in a <unk> . <end> pred sentense: a plate with food and a sandwich and a bread . a table . a restaurant restaurant <end>
Best_accuracy: 5.98714064365865 BLEU score: 0.0845

From the BLEU score and the predicted sentence from the model, it can be seen that the performance in GRU is not good.Thus, discussed to change back to use LSTM instead of GRU.

Performance of using LSTM:

Step [100/6333], Loss: 1616.0096 , Accuracy: 0.3176, BLEU score: 0.0165, Time: 7.48 mins target sentence: a man sitting in a room with a food on a table <end> pred sentence: a man is in a table of a a . a table .
Step [1000/6333], Loss: 1614.5359 , Accuracy: 0.3184, BLEU score: 0.0169, Time: 54.69 mins target sentence: a train travels around the bend of a grassy area . <end> pred sentence: a man of on a side . a table field . <end>
Step [1200/6333], Loss: 1655.2610 , Accuracy: 0.3185, BLEU score: 0.0174, Time: 61.76 mins target sentence: a man in green pants suspenders and a white shirt . <end> pred sentence: a man room with a man a a . a . .

Conclusion:

Comparing with the accuracy and BLEU score on GRU and LSTM model, it can be seen that the performance in GRU is better than LSTM model. Thus, we used GRU model as our decoder model to predict the single image in GUI.

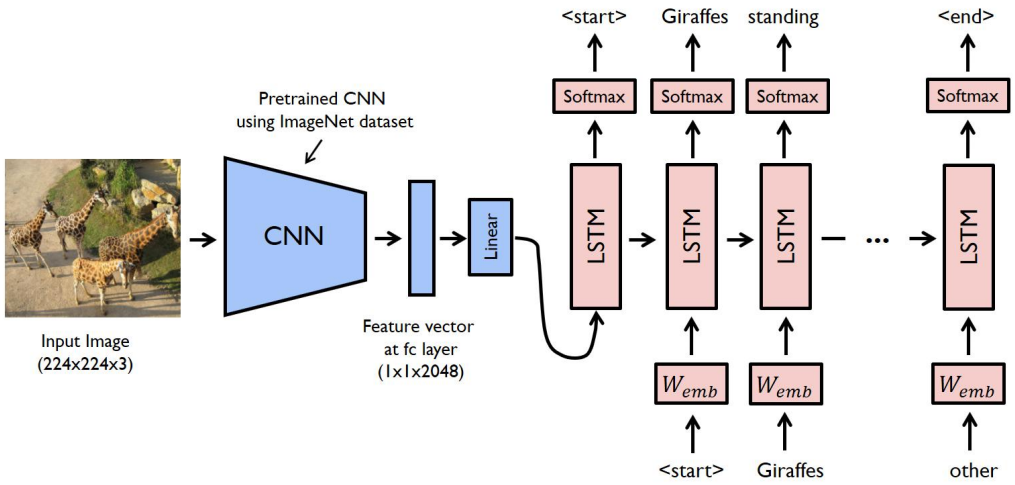
To run the code to start training

1. Run “resize.py” to resize the each image.
- Store the images at “./data/resized_train2014” and “./data/resized_val2014” folders.
2. Run “image_captioning.py”.
- Store the best model after epochs in current directory with name “best_model_encoder.pth” and “best_model_decoder.pth”.
- The checkpoints of training will save in “./data/models/”.
3. To adjust parameters.
- Open “image_captioning.py” file, adjust the parameters under ‘if __name__ == '__main__' .

* Node:

1. Please ensure the data is in current directory under “/data” folder, i.e “./data/*”
2. “/train2014” folder contains all train images, “/val2014” folder contains all validation images and “/annotations” folders contains “captions_train2014.json” and ‘captions_val2014.json” json files in order to load the captions.

Framework



In the project, the team decided to use an attention model as RNN decode model while used densenet121 as CNN encoder model with initialized weights in all layers, comparing the performance of using LSTM and GRU in decoder model to find out the which model will give higher score.

Attention model source: https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning
This is the git source we used to help implement an attention model. In this source, it used resnet101 as encoder model and LSTM with attention model as decode model. Thus, we change the model from resnet101 to densenet121 and LSTM to GRU, keep the attention model.

resize.py

The code used to resize each images in train dataset and val dataset. In the end will get all image with size (256 * 256) in “resized_train2014” and “resized_val2014”

image_captioning.py

The code to train and test the image data set.

Build_Vocab section

Generate the vocabulary list that include all the words appear in train dataset and map into an index list for model training.

1. Vocabulary class

Functions:

add_word: map each word into both word list and index list.

call: map the word to its index in index list and return index.

get_word: map the index to its word in word list and return word.

2. Build_vocab function

Generate a vocabulary mapping list in index form including <pad> in 0 position, <start> in 1 position, <end> in 2 position and <unk> in 3 position.

Data_loader section

Create a data loader for train as train_loader and for validation as val_loader.

1. CocoDataset class

Load the image from train set and val set and return image and target(caption) in tensor form.

2. Collate_fn function

Creates mini-batch tensors from the list of tuples (image, caption) and return image, caption and list of valid length for each padded caption.

3. get_loader function

Return the dataloader for train and val.

Model section

Overall structure: Encoder using densenet121 layers; Decoder using GRU/LSTM with an attention model

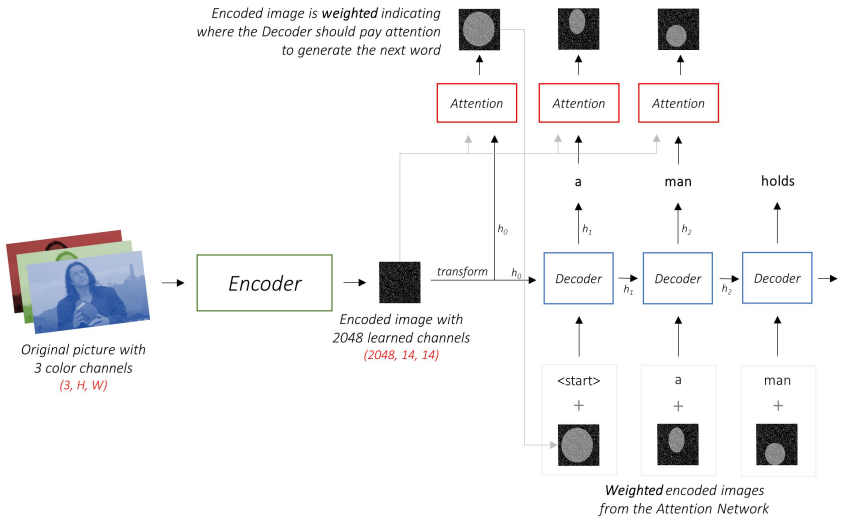


Figure 1: flow diagram from git, in the project, the dim of encoded image is (1024, 14, 14)

EncoderCNN(nn.Module):

- Used densenet121 pretrain model layers except last classifier layer with initialized weights. It will train all the weights in all layers in the model and output the features with shape (batch size, 1024, 7 ,7)
- Resize the image with default value (14*14) to fixed size to allow input images with variable size using an AdaptiveAvgPool2d layer. The dimension become (batch size, 1024, 14 ,14)
- Change the dimension from (batch size, 1024, 14 ,14) to (batch size, 14 , 14 1024). 1024 will be the size of feature of encoder output that put into decoder model as decoder input. Let 196 (14*14=196) become the pixels number. In the attention model, the weights of the pixels will add up to 1. If there are P pixels in our encoded image, then at each timestep t:

$$\sum_{p}^{P} \alpha_{p,t} = 1$$

This process is used to compute the probability that a pixel is the place to look to generate the next word using softmax to get the attention weights in attention model.

Attention(nn.Module):

* The concept in attention model:

The Encoder's output has multiple points in space or time. In image captioning, consider some pixels more important than others. In sequence to sequence tasks like machine translation, consider some words more important than others. There is an example that shows in Figure 2.

1. Add encoder output features and hidden size from previous time step together.
2. The output takes a relu and then a linear layer, resulting in the dimension of (batch size * number of pixel)
3. After that, taking a softmax to compute the probability of each pixel and multiple them with encoder features to bring out the more important pixels and give attention to those weights as decoder input.

Decoder model with attention model:

1. Once the Encoder generates the encoded image, transform the encoder features to create the initial hidden state h for the GRU decoder/initial hidden state h and cell state c for the LSTM decoder.
2. At each decode step,
    (1) the encoded image and the previous hidden state is used to generate weights for each pixel in the Attention network.
    (2) the previously generated word and the weighted average of the encoding are fed to the GRU/LSTM decoder to generate the next word.

Attention model is useful to give higher performance as predicting the words according to the attention area i.e the regions with higher weights.

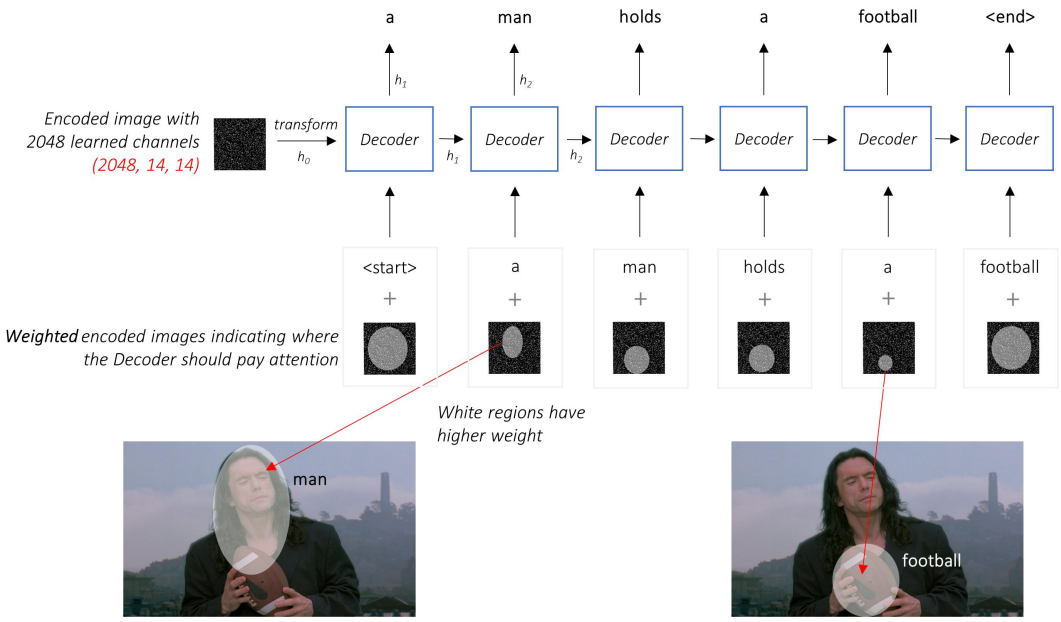
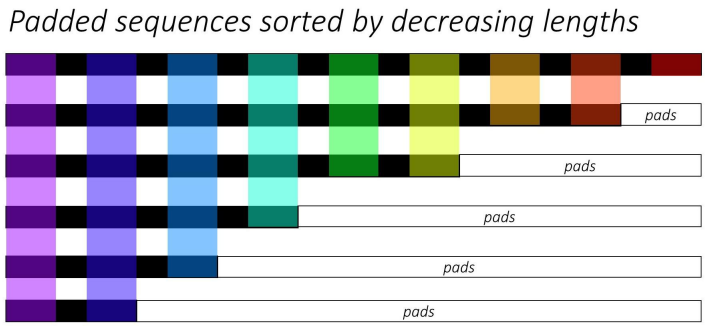


Figure 2: a sample of training an image with an attention model

3. Since the sequence may have different lengths and padding the sequence according to the max length of each batch size, Train the input with padding with GRU/LSTM cells may reduce the performance. The source code introduce an way to solve the issue. Sort the N images and captions by decreasing caption lengths, allowing to process only valid timesteps.



It can iterate over each timestep, processing only the colored regions, which are the effective batch size N_t at that timestep. The sorting allows the top N_t at any timestep to align with the outputs from the previous step.

Train section

The train function to start training the data from train dataset, train the weights in each layer and minimize the loss the function to get a best model within number of epochs, return average loss. During train, it will print loss, perplexity and using time for each log_step which can adjust in 'Main section'.

Validation section

The validate function to test the performance of training model and save the best model from epochs. Return validation loss, validation accuracy base on words and BLEU score base on each sentence. During validation, it will also print the three values and using time for each 1000

images which can adjust in ‘validation section’

Main section

It is the section to setup train and validation models.

1. Image normalization (transfer learning with transforms)
2. Train loader and val loader
3. Encoder and Decoder models
4. Loss function and optimizer

Run section

It is the section to select and initiate each parameter for training and validating.

1. Path for loading image and captions. The default path is ‘<current directory>/data/*’
2. Path for saving training models. The default path is ‘<current directory>/data/models/’
3. Path for load vocab list. The default path is ‘<current directory/vocab.pkl’
4. Crop_size for transfer learning. The default value is 224.
5. Log_step: print training loss at each log step. The default value is 100.
6. Save_step: step size for saving trained models the default value is 10000.
7. Model parameters: the default value => embed_size = 256, hidden_size = 512, num_epochs = 3, batch_size = 32, num_workers = 2, learning_rate = 0.001, attention_dim = 512, dropout = 0.5 (*if dropout is not define, the default value is 0.5, can be adjusted in DecoderRNN class)

*The reason to pick number of epoch to be 3 and batch size to be 32:

- During training, we realized that after the first epoch, the loss will float up and down within a range, then try with different learning rate, lr =0.001, lr = 0.0001, lr = 0.05 and lr =0.01; different dropout ratio, dropout = 0.1, dropout = 0.2 and dropout = 0.5. As the result, the change in loss is not obvious. Thus, due to the time consuming reason after many different trying, we set up the num_epochs to be 3. Base on what we have observed, the possible reason could be that:
    1. Not enough size in vocabulary list as only 9900+ words from train set.
    2. Not enough size of training images as only 80000+ images for image captioning.

- The default value for batch_size is 128, yet when the team added attention model which increase the calculation and operation in GPU, causing highly increase in timing for each epoch. Thus, after discussing, we plan to use 32 as batch size to train the model faster.

GUI

In “gui.py”, there is extra function called sample inside the decode model. This is the function used to sampling the captions in GUI prediction. Predict the first word from the encoder features as caption <start> and attention area, then predict the next word base on the previous predicted caption and attention area according to image features.

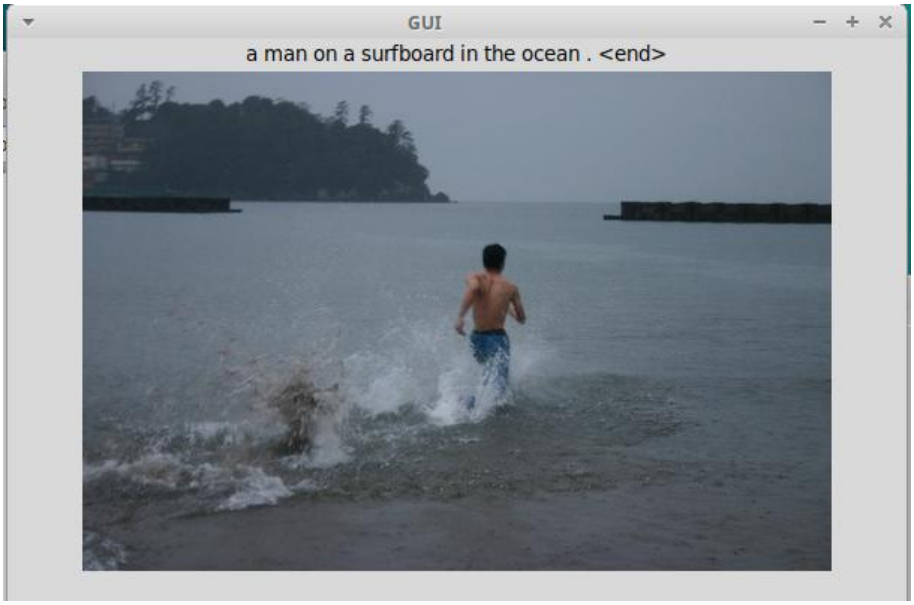
Some predicted samples:

GUI

a boat in a canal . <end>

GUI

a living room with a large window . <end>


GUI
a man on a surfboard in the ocean . <end>


GUI
a small airplane on a runway . <end>

Two weirdest/funniest wrong predictions