Image Captioning Final Report

Andrew Chen (aachen3) · Edward Huang (ezhuang2) · Bennett Ip (hip2)
Patrick Cole (pacole2) · Zih-Siou Hung (zhung2)

December 15th, 2019

1 Introduction and Related Work

In the final project, we focused on image captioning. In this task, one has to generate a sentence to describe the content of an image. We re-implemented the paper "Show and Tell: A Neural Image Caption Generator" [1], which uses a pretrained CNN network to extract the image features, and combined them with a recurrent neural net to produce the caption. We conduct all our experiments on the COCO image captioning dataset, and achieve similar or even better results to when compared against the TA's using the BLEU, CIDEr, and ROUGE_L metrics[2].

The image captioning task is in the intersection of computer vision models and natural language processing. The system not only has to understand the image to describe it in words, but also has to know the way to generate a logically correct sentence. In the show and tell model presented by Google, image features are extracted from a pretrained neural network for classification and we can pass these features to an RNN in order to generate a sentence about the features.

2 Model Architecture and Objective Function

The full caption generation model is shown in Figure 1. We first attempted to use Google's InceptionV3 [3] CNN model which receives a 0.779 top-1 accuracy and 0.937 top-5 accuracy on ImageNet to extract the image features. We started with this model to get our image features because that is what they used in the original paper [1]. Through experimentation we found that utilizing Resnet34 [4] to obtain our image features performed much better. A more detailed summary of these results and their performance on the given metrics can be found in Section 6. During training time, We feed the concatenation of the image feature and the ground truth caption embedding into a LSTM layer to generate the sentence, and train our model to predict the correct sentences given a certain image. In test time, we apply greedy strategy, i.e. beam search with beam size being 1, to generate the caption word by word.

2.1 PyTorch Model Description

We put our model definition which is written in Pytorch in this section. There is a CNN embedding layer to extract image features, an embedding layer for a vocabulary size of 12004, a LSTM layer to generate captions, and finally a decoding layer that takes the output of the LSTM and maps it to the most likely word in our vocabulary.

```
Show_and_tell(
  (cnn_embedding): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    ...
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        ...
    )
    (fc): Linear(in_features=512, out_features=512, bias=True)
  )
  (embedding): Embedding(12004, 512)
  (lstm): LSTM(512, 1024, batch_first=True, dropout=0.3)
  (decode): Linear(in_features=1024, out_features=12004, bias=True)
)
```

2.2 Objective Function

Recurrent neural networks are similar to hidden Markov models with a more powerful memory technique. The probability of outputting a word is dependent on the "hidden" state that we're in and the current input. Moreover, the "hidden" state is conditioned on the previous hidden states that we have seen so far in our recurrent net.

Intuitively, we want to maximize the probability of predicting the labeled caption of an image, and thus our objective is to minimize the following loss function:

$$L(I, S) = -\sum_{t=1}^{N} \log p_t(S_t) \tag{1}$$

The notations used in the loss function are described below.

- I: input image
- S: sentence
  - S_i: i'th word in the sentence
  - S_0: special start word
  - S_n: special stop word
- x: image/word embeddings
  - x_{-1}: image embedding (features) from the CNN ($CNN(I)$)
  - x_t: word embedding of the t'th workd
- p: probability vector of the predicted words
  - p_{t+1}: probability vector of the $t+1$'th word
- $t \in [0, N-1]$: time step

Note that this loss is just the summation of the cross entropy loss at each time step.

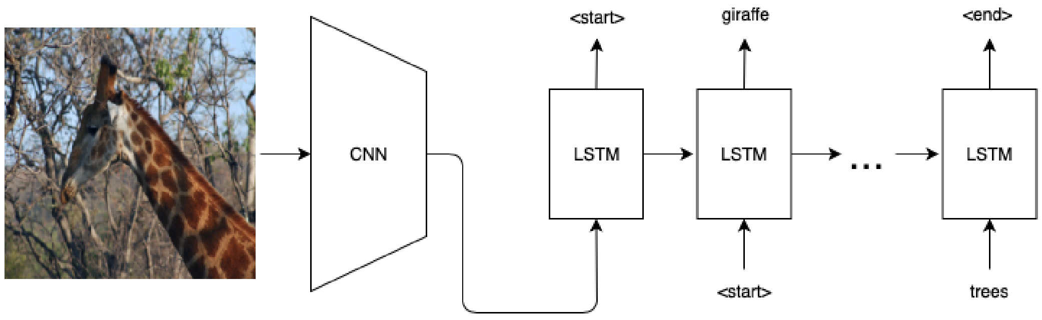


Figure 1: The network is composed of a pretrained CNN model (either InceptionV3 or Resnet34) followed by an LSTM with a fully connected layer to decode output token in between

3 Training Methods

During training, we first initialize the CNN portion of our network with a classification model pretrained on ImageNet. As for all other learnable weights, we use the default weight initializing method in Pytorch. We didn't freeze any parameters in the networks, meaning that the convolution neural network is also trainable. We used Adam as our optimizer with a initial learning rate of 0.001 and beta values of 0.9 and 0.999 to train the whole model. We trained for 30 epochs with batch size of 64 and decayed the learning rate by a factor of 10 every 10 epochs. We applied a dropout layer with probability 0.3 after LSTM as our regularization method. We did not utilize weight decay. In order to train with these pretrained architectures we also need to resize the image's based on which model we used. For InceptionV3 it requires images of size 299×299 and for Resnet34 requires images of size 224×224.

As for the hyperparameters, we used one layer of LSTM with 1024 hidden units. Our vocabulary size is 12004, including four special character: $< start >$, $< end >$, $< pad >$, and $< unknown >$. The first two indicate the start and the end of a caption. The $< pad >$ is used for batch processing so that the short caption can be padded to have the same length of the long one. The last special symbol, $< unknown >$, is used to represent words that are out of the normal 12000 words in our vocabulary. We obtained our normal 12000 word vocabulary by utilizing the `CountVectorizer` package from `sklearn`. This package allowed us to keep track of the frequency of the words in all of the captions from the training set and then select the top 12000 words for our vocabulary.

4 Dataset (COCO)

We use the Common Objects in Context (COCO) dataset to evaluate our model [2]. The COCO dataset mainly consists of images of day-to-day objects with more images being added each year. The dataset contains images with many hand-labeled features including image segmentation, categorical labels, and closed captions, but we just use the closed captions for this project. We use the 2017 version, which has 118,287 training images and 5,000 validation images. Moreover, each image had approximately 5 captions which usually ranged from 5 to 15 words. During training, we just sampled one caption randomly.

5 Computational Cost of Training

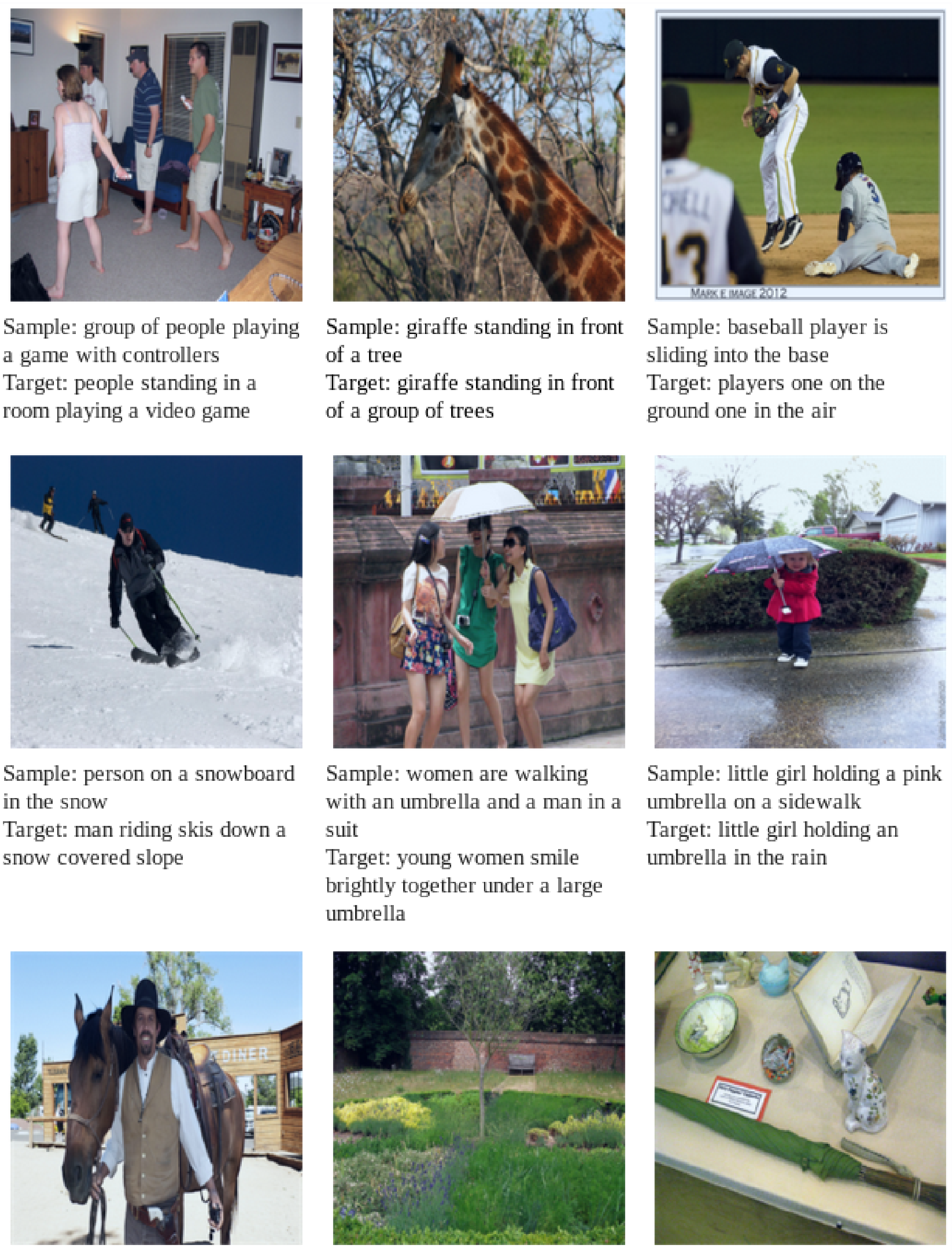
Initially we used Google Colab with a single K80 GPU for drafting our ended up training it on there for approximately 5 hours to verify our code was training correct.
After we had a working code set we utilized a Blue Waters' GPU node with a single K20 GPU for training. In our implementation, we use *pack_padded_sequence* in PyTorch to speed up the training. This is a fancy way to batch the variable length sequences so that we use less computation resources. However, the training still took approximately 60 minutes per epoch.
Each member of our group ran various experiments, that tested different architectures and hyper-parameters. In total we ran about 12 different experiments each taking approximately 20 hours in total to train. Some of the better performing variations are expanded upon in the Results section of this report.

6 Results

For our results we ran multiple experiments with different architectures and hyperparameters. Table 1 shows the main architectures that we used for experiments. At first we attempted to use InceptionV3 to obtain our image features and we only trained the final linear layer. This resulted in less than desirable results, so then we changed the architecture to be Resnet34. This improved the results but they were still not comparable to the TA's results. So we opted to train the entire network. We used Resnet34 still because it provided better results than InceptionV3 with the previous experiments and training would easier without having to worry about the auxillary output. When training the full model we also attempted to use a different number of hidden units in the LSTM layer. We found that using Resnet34 to get our image features, and an LSTM layer with 1024 hidden units gave the best result. Note that although we obtained slightly worse ROUGE_L score, we get a much high CIDEr score. ROUGE_L score uses Longest Common Subsequence (LCS) based statistics, which have been shown to correlate weakly with human judgment. On the contrary, CIDEr score have shown high agreement with consensus as assessed by humans. We can also see this from our baseball player example in Figure 2. We actually generate some pretty good captions, but it falls short for ROUGE_L.

	BLEU1	**BLEU2**	**BLEU3**	**BLEU4**	**CIDEr**	**ROUGE_L**
TA	64.6	45.9	31.7	22.0	69.4	<u>47.6</u>
Paper	**71.3**	**54.2**	**40.7**	**27.7**	**85.5**	**53.0**
InceptionV3+LSTM512^1	56.0	36.5	23.5	15.4	51.2	37.7
Res34+LSTM512^1	59.2	40.4	27.3	18.6	60.2	40.3
Res34+LSTM512^2	64.2	45.9	32.2	22.5	74.1	43.9
Res34+LSTM1024^2	<u>65.0</u>	<u>46.6</u>	<u>32.6</u>	<u>22.8</u>	<u>76.9</u>	44.3

Table 1: Full Validation set performance on COCO captioning dataset. **Bold** indicates highest number, <u>underline</u> indicates the second highest. The superscripts ^{1} and ^{2} represent training the final linear layer and the entire network in the CNN respectively.



Sample: group of people playing a game with controllers
Target: people standing in a room playing a video game

Sample: giraffe standing in front of a tree
Target: giraffe standing in front of a group of trees

Sample: baseball player is sliding into the base
Target: players one on the ground one in the air

Sample: person on a snowboard in the snow
Target: man riding skis down a snow covered slope

Sample: women are walking with an umbrella and a man in a suit
Target: young women smile brightly together under a large umbrella

Sample: little girl holding a pink umbrella on a sidewalk
Target: little girl holding an umbrella in the rain

Sample: woman in a dress is standing next to a horse
Target: a man in a black hat standing next to and holding the reigns of a horse

Sample: brown cow standing in a field next to a tree
Target: tree sitting in the middle of a lush green field

Sample: table with a plate of food and a cup of coffee
Target: table with an umbrella and other things

Figure 2: Examples from our model

7 Code Description

We will discuss our code in this section. First, let's examine our model (*Show_and_tell*). In the initialization, we created ResNet for image feature extraction, a word embedding layer, LSTM layer to generate the captions, and the final decoding layer to convert the hidden state back to a word.

```python
def __init__(self, vocab_size=12004, embed_size=1024, hidden_units=1024,
                max_seq_length=25):
    super(Show_and_tell, self).__init__()

    self.cnn_embedding = resnet34(pretrained=True)

    out_features = self.cnn_embedding.fc.in_features
    self.cnn_embedding.fc = nn.Linear(out_features, embed_size)

    self.embedding = nn.Embedding(vocab_size, embed_size)

    self.lstm = nn.LSTM(embed_size, hidden_units, num_layers=num_lstm_layers,
                            dropout=0.3, batch_first=True)

    self.decode = nn.Linear(hidden_units, vocab_size)

    self.max_seq_length = max_seq_length
```

In our forward function during training, we first extract features using ResNet. Next, we concatenate the image feature with the ground truth caption embedding. Note that we get rid of the $< end >$ character during *pack_padded_sequence* so that we won't feed it into the LSTM. Finally, we use a decoding layer to get the words.

```python
def forward_train(self, imgs, captions, lengths):
    """
    Args:
        imgs: torch.FloatTensor([B, 3, 224, 224]), which is the input to resnet
        catptions: torch.LongTensor([B, max_seq_len]), which is the padded captions
        lengths: torch.LongTensor([B,]): the actual length of the captions
     """
    img_feats = self.cnn_embedding(imgs)

    B, seq_len = captions.shape
    word_feats = self.embedding(captions)

    inputs = torch.cat([img_feats.unsqueeze(1), word_feats], dim=1)
    packed_inputs = pack_padded_sequence(inputs, lengths, batch_first=True)
    packed_outputs = self.lstm(packed_inputs)
    outputs = self.decode(packed_outputs[0].data)
    return outputs
```

In the training loop, we also use *pack_padded_sequence* to make the ground truth captions have the same format of the generated outputs.

```python
for epoch in range(epochs):
    ...
    optimizer.zero_grad()
    targets = pack_padded_sequence(captions, lengths, batch_first=True).data
    outputs = model(images, captions, lengths)
    loss = criterion(outputs, targets)
    loss.backward()
    optimizer.step()
    ...
```

As for testing, we use greedy search to generate the captions. At every timestep, we feed the most possible previous word into the LSTM.

```python
def greedy_inference(self, imgs, states=None):
    predicted_ids = []
    output_probs = []
    img_feats = self.cnn_embedding(imgs)
    inputs = img_feats.unsqueeze(1)
    for i in range(self.max_seq_length):
        hiddens, states = self.lstm(inputs, states)
        outputs = self.decode(hiddens.squeeze(1))
        _, predicted = outputs.max(1)

        predicted_ids.append(predicted)
        output_probs.append(outputs)
        inputs = self.embedding(predicted).unsqueeze(1)
    predicted_ids = torch.stack(predicted_ids, 1)
    output_probs = torch.stack(output_probs, 1)
    return predicted_ids, output_probs
```

8 Acknowledgements

In this project we had to generate the BLEU, CIDEr, and ROUGE_L scores to compare our results to that of the TA and the paper. For this we wrote a script that utilized the libraries in the coco-captions GitHub repository [5]. We also found another GitHub repository that had implemented this project before [6]. We looked at this repository for inspiration in hyperparameters since they reported strong results with their experiments.

References

[1] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator, 2014.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[3] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[4] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition, 2015.

[5] Tsung-Yi Lin. coco-caption. `https://github.com/tylin/coco-caption`, 2018.

[6] Wojciech Kryscinski. Neural image captioning. `https://github.com/muggin/show-and-tell`, 2018.