Project 3: Image Captioning of Novel Objects

1. Introduction

The Image Captioning task is defined as generating a textual description of an image relative to the content of the image. An AI system needs not only to recognize the image, but also to understand and interpret the content of the image it sees, and to be able to describe the relationships between objects in the image as a human would. (Input: a picture -Output: a textual description)
Most of the currently used image caption models are still encoder-decoder models, which use CNN to do feature extraction and then feed into LSTM, but one drawback of such models is that they are based on a large number of image-caption pairs in the training set, or such models only have in-domain view, and the models do not work well when the content of the incoming image caption and the training set differ greatly. When the content of the training set differs greatly, the model will not work well. At the same time, real life is colorful, and a model that can describe novel scenes or objects in out-of-domain images is also valuable. In this project, I will try to implement the BLIP[3] and NOC[1] to solve the problem of image captioning on novel objects.

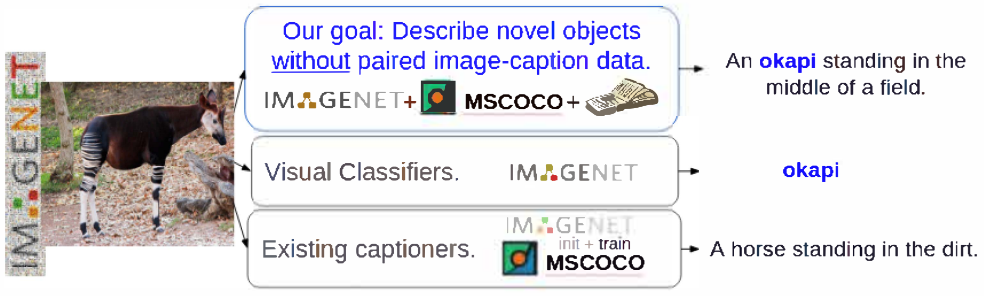

IMAGENET
Our goal: Describe novel objects without paired image-caption data.
IMAGENET+ MSCOCO+
An okapi standing in the middle of a field.
Visual Classifiers. IMAGENET
okapi
Existing captioners. IMAGENET init + train MSCOCO
A horse standing in the dirt.

Figure 1: Goal Description

MSCOCO is a large scale dataset for training of image captioning systems. It contains(2014 version) more than 600,000 image-caption pairs. It contains training and validation subsets, made respectively of 82, 783 and 40, 504 images, where every image has 5 human-written annotations in English. The dataset I used in this project is a subset of the MSCOCO. Deep Compositional Captioning(DCC)[2] offers a split of the COCO dataset. This subset removes eight categories of objects: bottels, bus, couch, microwave, pizza, racket, suitcase and zebra from the original dataset. I use this split to train and evaluate the model performance on unseen objects.

2. Related Work

Lisa Anne's paper "Deep Compositional Captioning" addresses for the first time thepurpose of describing the novel target(which does not exists in the training set)[2]. Past image captioning methods were unable to generate sentences about objects that were not visible in the caption corpus. In contrast, this model effectively integrates information from independent image datasets and text corpora to compose descriptions of new objects without any paired image-sentence data. This paper proposes DCC (Deep Compositional Captioning), which can compose sentences describing new objects and their interactions with other objects. The method consists of three phases:

(a) training a deep lexical classifier and deep language model with unpaired data

(b) combining the lexical classifier and language model into a caption model which is trained on paired image-sentence data

(c) transferring knowledge from words which appear in paired image-sentence data to words which do not appear in paired image-sentence data.

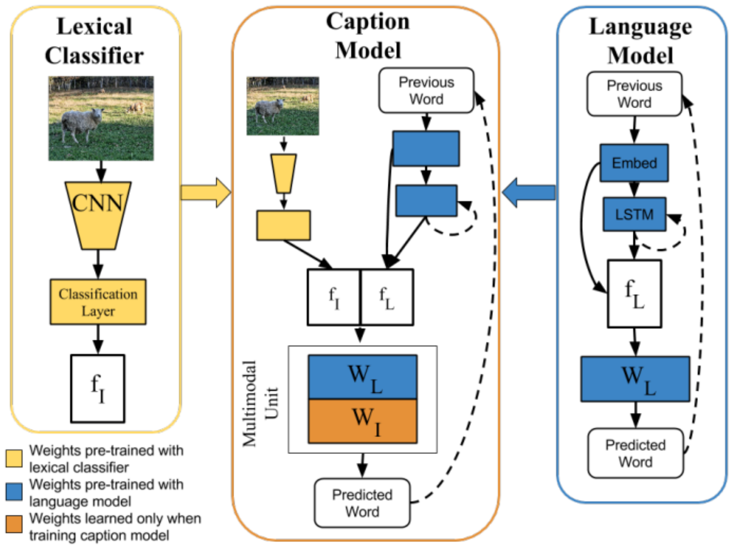


Figure 2: Structure of DCC

3. Model

3.1. BLIP

Modern neural network image description methods produce natural language descriptions that, while more fluent, rely on language models to generate sentences and tend to be less dependent on images.

BLIP[3] is a unified vision-language pre-training (VLP) framework that learns from noisy image text pairs. It is able to cover a wider range of downstream tasks than existing methods.

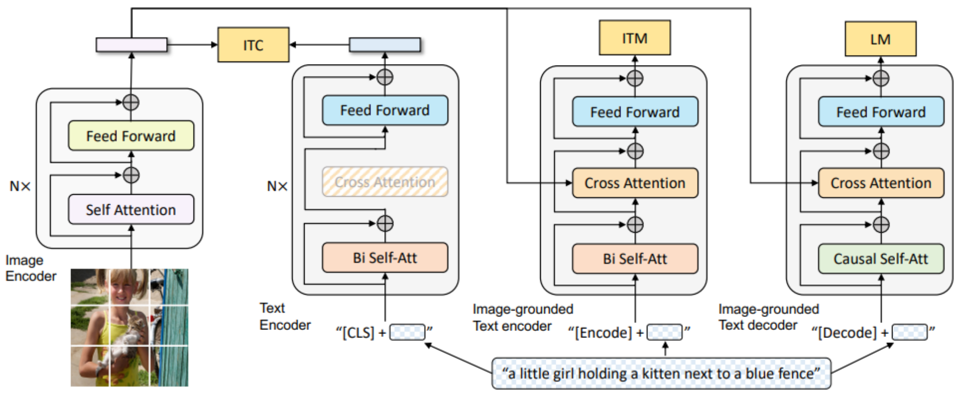


Figure 3: Structure of BLIP

BLIP uses Visual Transformer as an image encoder to divide the input image into patches, and then encodes the patches into an embedding sequence with an additional [CLS] token to represent the global image features.
The researchers jointly optimized three objectives during pre-training, two comprehension-based objectives and one generation-based objective. Each image text pair requires three forward propagations through the text transformer, which activates different functions to compute image text contrast loss (ITC) image text matching loss (ITM) and language modeling loss (LM).
The encoder uses bidirectional self-attention to construct a representation for the current input token, while the decoder uses causal self-attention to predict the next token.The authors also propose a new method for improving the quality of text corpora, CapFilt, which introduces a captioner to generate annotations for a given web image and a filter to eliminate noisy image text pairs.The researchers combine the filtered image text pairs with manually annotated pairs to generate a new dataset and use it to pre-train the new model.
It can be found that the main power of the model lies in the pre-training and data enhancement process and further fine-tuning on the dataset afterwards.

3.1.1. implementation details

Using BLIP w/ ViT-B and CapFilt-L models as initial training results, fine-tuning 1 epoch on the training set.

ViT	base
batch size	6
learning rate	5e-6
weight decay	0.05
optimizer	AdamW

3.2. NOC

"Novel Object Captioner (NOC)"[1], has the following network structure, which consists of three subtasks: language modeling + target recognition model + picture generation. Intuitively, it can be seen that when trained jointly, the model can achieve the following goals: (1) generate text that reads well; (2) recognize visual targets in images; (3) generate reasonable picture descriptions. When (2) and (3) are complementary, the generated text can be made to contain the results of (2), where (2) is the target of imagenet 1000 classes, and (3) is the MSCOCO caption dataset, which contains only 80 classes of targets, i.e., in the inference phase, the generated text can contain the remaining 920 kinds of targets.

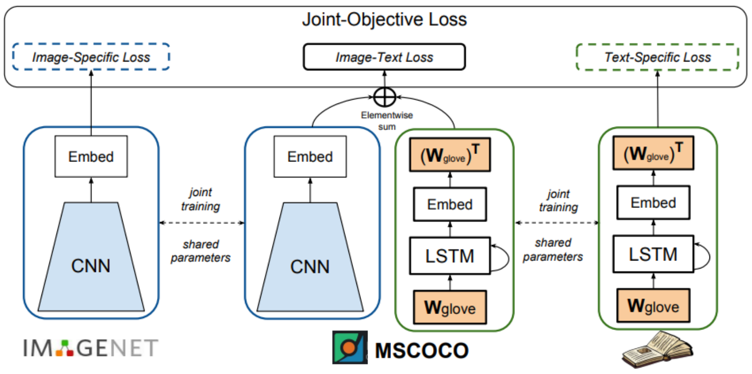


Figure 4: Structure of NOC

3.2.1. Encoder

I choose the pre-trained ResNet-152 model and remove the final fully connected layer. The output features are then mapped to vocabulary size dimensions using a linear layer. Finally, the features are normalized using Batch Normalization. I refer to github for the code implementation[6].

```python
class EncoderCNN(nn.Module):
    def __init__(self, hidden_size, vocab_size):
        super(EncoderCNN, self).__init__()
        resnet = models.resnet152(pretrained=True)
        modules = list(resnet.children())[:-1]
        self.resnet = nn.Sequential(*modules)
        self.linear = nn.Linear(resnet.fc.in_features, vocab_size)
        self.bn = nn.BatchNorm1d(vocab_size, momentum=0.01)

    def forward(self, images):
        with torch.no_grad():
            features = self.resnet(images)
        features = features.view(features.size(0), -1)
        features = self.linear(features)
        features = self.bn(features)
        return features
```

3.2.2. Decoder

The LSTM layer processes the input sequence of embedding vectors and outputs the hidden state. And a linear layer maps the output of the LSTM layer to the dimension of the embedding vectors.
And a sample method accept as input a PyTorch tensor features containing the embedded input features corresponding to a single image.

```
class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size,
num_layers, weights_matrix, max_seq_length=20):
        super(DecoderRNN, self).__init__()
        self.embed, num_embeddings, embedding_dim =
create_emb_layer(weights_matrix, False)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers,
batch_first=True)
        self.linear = nn.Linear(hidden_size, embed_size)
        self.max_seq_length = max_seq_length

    def forward(self, captions, lengths):
        embeddings = self.embed(captions)
        packed = pack_padded_sequence(embeddings, lengths,
batch_first=True)
        hiddens, _ = self.lstm(packed)
        x = self.linear(hiddens[0])
        x = x.matmul(self.embed.weight.t())
        return x

    def sample(self, features, start_id, states=None):
        sampled_ids = []
        inputs = torch.tensor([start_id], dtype=torch.long)
        if torch.cuda.is_available():
            inputs = inputs.cuda()
        inputs = self.embed(inputs)
        inputs = inputs.unsqueeze(1)

        for i in range(self.max_seq_length):
            hiddens, states = self.lstm(inputs, states)
            outputs = self.linear(hiddens.squeeze(1))
            outputs = outputs.matmul(self.embed.weight.t())
            final = outputs + features[0]
            values, index = torch.max(final, 0)
            sampled_ids.append(index.data.item())
            inputs = self.embed(index)
            inputs = inputs.view(1, 1, -1)
        return sampled_ids
```

3.2.3. Loss and Optimizer

```
visual_loss = torch.nn.BCEWithLogitsLoss()
criterion = nn.CrossEntropyLoss()
caption_loss = nn.CrossEntropyLoss()
params = list(decoder.parameters()) + list(encoder.parameters())
optimizer = torch.optim.Adam(params, lr=args.learning_rate)
```

4. Experiment Results

I train the models on the split introduced by DCC. And I measured the BLEU-4, CIDEr, METEOR, SPICE and F1 scores.

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}.$$

4.1. BLIP

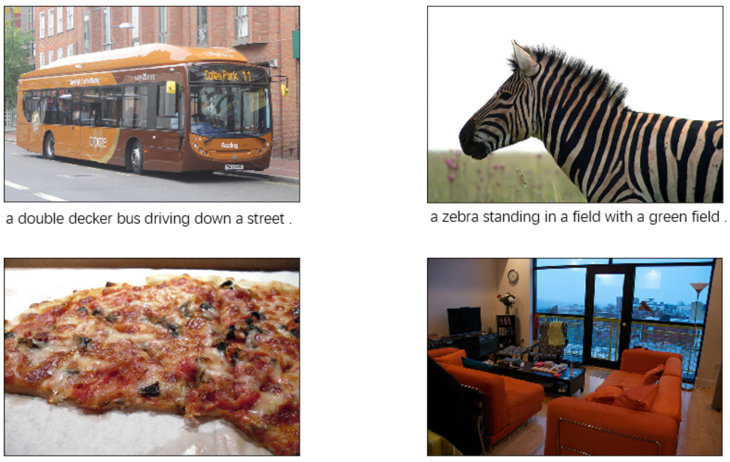
I use the library on github[4] and I referred to Referred to the reproduction and configuration process of others on github when I configured the relevant environment manually[5].

	BLEU-4	METEOR	CIDEr	SPICE	F1
bottle	35.44	28.41	113.78	20.2	0
bus	34.44	27.52	93.07	22.48	71.02
couch	36.27	28.93	93.98	21.53	8.53
microwave	32.23	27.37	75.09	18.87	20.17
pizza	30.15	25.14	80.89	18.44	7.82
racket	33.69	29.52	61.11	21.95	2.54
suitcase	33.58	26.59	96.61	19.87	0
zebra	23.44	22.46	50.81	15.22	37.75

Table 1 BLIP average score(%)

4.2. Novel Object Captioner

I also tried my simple model. I only run 3 epochs this time. And generated some predictions. Here are some examples of the generated caption by my model. Although there will be some deviations and grammatical errors in the details, the main objects can be basically described.



a double decker bus driving down a street .

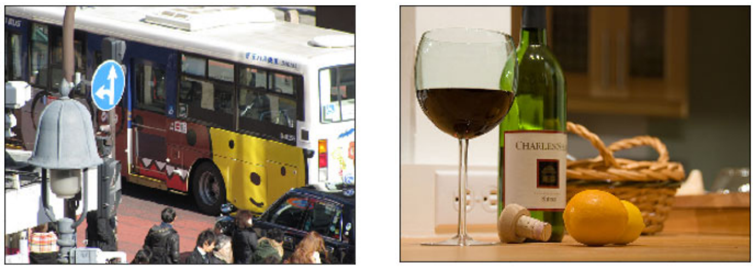
a zebra standing in a field with a green field .

A pizza with cheese , tomatoes , and cheese on a plate .

a living room with a couch , chair and television .

Figure 5: good samples

But on some images, it behaves badly.



a man is holding a bat on a baseball field .

a glass vase filled with flowers on a table .

Figure 6: bad samples

In these results, the generated descriptions barely correlate with the images. The main objects are also not identified.

Reference

[1] S. Venugopalan, L. A. Hendricks, M. Rohrbach, R. Mooney, T. Darrell and K. Saenko, "Captioning Images with Diverse Objects," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 1170-1178, doi: 10.1109/CVPR.2017.130.

[2] L. A. Hendricks, S. Venugopalan, M. Rohrbach, R. Mooney, K. Saenko and T. Darrell, "Deep Compositional Captioning: Describing Novel Object Categories without Paired

Training Data," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 1-10, doi: 10.1109/CVPR.2016.8.
[3] Li, Junnan, Li, Dongxu ,Xiong, Caiming, Hoi, Steven C. H. BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation.2022.CoRR , Vol. abs/2201.12086.
[4] https://github.com/salesforce/BLIP
[5]
https://github.com/huakyouin/NeuralNetwork/blob/main/Project203/BLIPmain/README.md
[6] https://github.com/willT97/Zero-shot-Image-Captioner