

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN CUỐI KỲ MÔN HỌC SÂU

SINH VIÊN : NGUYỄN KHẢ MINH

MSSV : 21124661

LỚP : DHKHMT17B

GVHD : THẦY LÊ VŨ HẠO

TP.Hồ Chí Minh – Ngày 6 Tháng 5 Năm 2025.

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HCM
KHOA CÔNG NGHỆ THÔNG TIN

PHIẾU NHIỆM VỤ

1. Họ và tên sinh viên

Nguyễn Khả Minh

MSSV: 21124661

2. Tên đề tài

Chú thích ảnh về cuộc sống quanh ta bằng mô hình học sâu.

3. Nhiệm vụ:

Tìm hiểu bài toán sinh chú thích ảnh (Image Captioning) trong lĩnh vực thị giác máy tính và xử lý ngôn ngữ tự nhiên.

Xây dựng mô hình học sâu gồm Encoder (CNN) và Decoder (LSTM) để mô tả nội dung ảnh thành văn bản tiếng Anh.

Thu thập và tiền xử lý dữ liệu ảnh và chú thích từ các bộ dữ liệu công khai như Flickr30k và MS COCO.

Huấn luyện mô hình trên tập huấn luyện và đánh giá chất lượng sinh chú thích bằng các chỉ số như BLEU score.

Trình bày kết quả, ví dụ đầu ra và phân tích chất lượng mô hình.

4. Kết quả dự kiến:

Mô hình học sâu có thể sinh ra các chú thích hợp lý, ngắn gọn, đúng ngữ cảnh cho ảnh đầu vào.

Đạt được BLEU score ở mức chấp nhận được trên tập kiểm thử.

Có báo cáo hoàn chỉnh trình bày rõ ràng các bước thực hiện, kết quả và đánh giá.

Có thể trình diễn mô hình bằng giao diện đơn giản hoặc notebook tương tác.

This image shows a full page of a handwriting practice worksheet. It consists of multiple sets of three horizontal dashed lines, providing a guide for letter height and placement. The lines are evenly spaced across the entire page, which is otherwise blank.

MỤC LỤC

PHIẾU NHIỆM VỤ	2
NHẬN XÉT CỦA GIẢNG VIÊN.....	3
CHƯƠNG 1 GIỚI THIỆU	4
1.1. Giới thiệu về bài toán sinh chú thích.....	4
1.2. Ý tưởng tổng thể	5
1.3. Mô hình cơ sở	6
1.4. Lý do lựa chọn mô hình.....	6
CHƯƠNG 2 DỮ LIỆU VÀ TIỀN XỬ LÝ	7
2.1 Dữ liệu sử dụng	7
2.2 Gộp hai tập dữ liệu thành một DataFrame:.....	7
2.3 Tiền xử lý ảnh.....	9
2.4 Tiền xử lý chú thích.....	10
CHƯƠNG 3 THIẾT KẾ VÀ HUẤN LUYỆN MÔ HÌNH	13
3.1 Thiết kế mô hình.....	13
3.2 Tham số huấn luyện.....	18
3.3 Huấn luyện mô hình.....	19
3.4 Thư viện sử dụng.....	23
CHƯƠNG 4 Đánh giá mô hình	24
4.1 Phương pháp đánh giá	24
4.2 Đánh giá mô hình bằng chỉ số BLUE:.....	26
KẾT LUẬN.....	29
DANH MỤC THAM KHẢO.....	29

CHƯƠNG 1 GIỚI THIỆU

1.1. Giới thiệu về bài toán sinh chú thích

Trong thời đại bùng nổ dữ liệu đa phương tiện, hình ảnh đã trở thành một dạng thông tin phổ biến và quan trọng. Tuy nhiên, việc hiểu và mô tả nội dung của hình ảnh theo cách mà con người làm vẫn là một thách thức lớn đối với máy tính. Bài toán sinh chú thích ảnh (Image Captioning) ra đời nhằm mục tiêu giúp máy có thể "nhìn" một bức ảnh và "nói" hoặc "viết" ra mô tả phù hợp bằng ngôn ngữ tự nhiên.

Sinh chú thích ảnh là một bài toán kết hợp giữa hai lĩnh vực quan trọng trong trí tuệ nhân tạo:

- Thị giác máy tính (Computer Vision): nhằm trích xuất và hiểu được nội dung trực quan từ ảnh.
- Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP): nhằm tạo ra câu mô tả một cách ngữ nghĩa và chính xác.

Cụ thể, đầu vào của bài toán là một hình ảnh bất kỳ, và đầu ra là một câu văn tiếng Anh mô tả nội dung chính của ảnh. Ví dụ, với một ảnh có người đang cưỡi xe đạp trên đường phố, mô hình có thể sinh ra chú thích như: "A man is riding a bicycle on the street."

Bài toán này có nhiều ứng dụng thực tiễn như:

- Hỗ trợ người khiếm thị tiếp cận nội dung ảnh.
- Tìm kiếm ảnh theo văn bản.
- Tự động sinh thuyết minh ảnh hoặc video.
- Góp phần vào các hệ thống thông minh như trợ lý ảo, robot quan sát.

Trong đề tài này, em tập trung xây dựng mô hình học sâu có khả năng sinh chú thích ảnh một cách tự động, chính xác và tự nhiên dựa trên các ảnh chụp đời sống hằng ngày.

1.2. Ý tưởng tổng thể

Bài toán sinh chú thích ảnh đòi hỏi mô hình phải hiểu được nội dung hình ảnh và biểu đạt nó bằng một câu văn. Do đó, ý tưởng tổng thể của mô hình được xây dựng dựa trên việc **kết hợp hai thành phần chính**:

- **Mạng nơ-ron tích chập (Convolutional Neural Network – CNN)**: dùng để trích xuất đặc trưng thị giác từ ảnh đầu vào. Trong đề tài này, mô hình ResNet50 được sử dụng làm Encoder để chuyển ảnh thành vector đặc trưng.
- **Mạng nơ-ron hồi tiếp (Recurrent Neural Network – RNN)**: cụ thể là **LSTM (Long Short-Term Memory)**, được sử dụng để sinh ra câu mô tả dựa trên đặc trưng ảnh đã trích xuất. Ngoài ra, mô hình cũng tích hợp cơ chế Attention giúp cải thiện độ chính xác trong việc sinh chú thích.

Mô hình hoạt động theo chu trình: ảnh \rightarrow vector đặc trưng (CNN) \rightarrow sinh từng từ mô tả (RNN).

1.3. Mô hình cơ sở

Mô hình em xây dựng được tham khảo và cải tiến dựa trên mô hình "**Show and Tell**" do Google đề xuất năm 2015 – một trong những mô hình tiên phong trong lĩnh vực sinh chú thích ảnh.

Mô hình "Show and Tell" sử dụng kiến trúc CNN + RNN với chu trình:

- ResNet/Inception để trích xuất đặc trưng ảnh.
- LSTM để sinh câu mô tả.

Ngoài ra, mô hình của em cũng học hỏi từ mô hình "**Show, Attend and Tell**" khi tích hợp thêm **cơ chế Attention** vào quá trình sinh từ, giúp mô hình tập trung vào các vùng ảnh quan trọng hơn tại từng bước sinh.

1.4. Lý do lựa chọn mô hình

Lý do lựa chọn mô hình "Show and Tell" và "Show, Attend and Tell" làm cơ sở để xây dựng mô hình gồm:

- **Tính hiệu quả đã được kiểm chứng:** Đây là những mô hình kinh điển với độ chính xác cao trong nhiều nghiên cứu và cuộc thi.
- **Cấu trúc rõ ràng, dễ mở rộng:** Việc tách Encoder và Decoder giúp dễ triển khai, tinh chỉnh và thay thế từng thành phần (ví dụ, thay LSTM bằng GRU, hoặc thay ResNet bằng EfficientNet).
- **Thích hợp với quy mô dữ liệu và mục tiêu học thuật:** Các mô hình này có thể hoạt động tốt trên bộ dữ liệu vừa phải như Flickr30k, MS COCO – phù hợp với năng lực phần cứng và thời gian làm đồ án.

CHƯƠNG 2 DỮ LIỆU VÀ TIỀN XỬ LÝ

2.1 Dữ liệu sử dụng

Trong đề tài này, em sử dụng hai tập dữ liệu lớn và phổ biến trong bài toán sinh chú thích ảnh:

- **Flickr30k**
- **MS COCO (Common Objects in Context)**

COCO (Common Objects in Context)

Bộ dữ liệu COCO có khoảng 330.000 ảnh, trong đó hơn 200.000 ảnh được gán nhãn. COCO bao gồm 80 lớp đối tượng như người, xe cộ, động vật, đồ vật và nhiều loại khác. Mỗi ảnh trong COCO đi kèm với 5 đến 6 mô tả văn bản ngắn gọn, giúp mô hình học được sự liên kết giữa hình ảnh và văn bản. Bộ dữ liệu này bao gồm các tác vụ như nhận diện đối tượng, phân đoạn ảnh và mô tả ảnh. Mỗi ảnh trong COCO chứa các thông tin chi tiết như mô tả văn bản, thông tin phân vùng đối tượng, nhãn đối tượng và bounding boxes.

Flickr30k

Bộ dữ liệu Flickr30k có khoảng 31.000 ảnh, với các chủ đề đa dạng như con người, động vật và các cảnh vật đời sống thường ngày. Mỗi ảnh trong bộ dữ liệu này có 5 mô tả văn bản khác nhau. Dữ liệu đến từ cộng đồng Flickr, với các mô tả được cung cấp bởi người dùng cộng đồng hoặc qua cơ chế đánh dấu. Bộ dữ liệu chủ yếu được sử dụng cho mô tả ảnh. Mỗi ảnh đi kèm với các mô tả và đôi khi có thông tin bổ sung về ngữ cảnh, ví dụ như các hành động hoặc mối quan hệ giữa các đối tượng trong ảnh.

2.2 Gộp hai tập dữ liệu thành một DataFrame:

Để tận dụng sức mạnh và sự đa dạng của cả hai bộ dữ liệu Flickr30k và MS COCO, em tiến hành gộp dữ liệu từ hai nguồn này vào một DataFrame duy nhất với cấu trúc chuẩn gồm:

image_path: đường dẫn đến ảnh.

caption: chú thích tương ứng với ảnh.

Các bước thực hiện:

Đọc dữ liệu từ hai tập:

- Với Flickr30k: đọc từ tệp captions.txt chứa đường dẫn ảnh và chú thích.
- Với MS COCO: sử dụng file chú thích .json (captions_train2017.json) để ánh xạ image_id với caption.

Tạo DataFrame riêng biệt cho từng tập:

```
flickr_df = pd.DataFrame({
    'image_path': flickr_image_paths,
    'caption': flickr_captions
})

coco_df = pd.DataFrame({
    'image_path': coco_image_paths,
    'caption': coco_captions
})
```

Nối hai DataFrame lại với nhau:

```
# Merge the train and validation DataFrames
df = pd.concat([flickr, coco_df], ignore_index=True)
```

Sau tất cả các bước trên, chia tập dữ liệu lớn thành 3 tập training, validation và testing:

```

from sklearn.model_selection import train_test_split

# Step 1: Get unique image names
image_names = df['image_name'].unique()

# Step 2: Split image names into training and temporary (validation + test) sets
train_image_names, temp_image_names = train_test_split(image_names, test_size=0.2, random_state=42)

# Step 3: Split the temporary image names into validation and test sets
val_image_names, test_image_names = train_test_split(temp_image_names, test_size=0.5, random_state=42)

# Step 4: Filter the original DataFrame based on the image names in each split
train_df = df[df['image_name'].isin(train_image_names)]
val_df = df[df['image_name'].isin(val_image_names)]
test_df = df[df['image_name'].isin(test_image_names)]

```

```

print("Length of training set", len(train_df))
print("Length of validation set", len(val_df))
print("Length of testing set", len(test_df))

```

```

Length of training set 478427
Length of validation set 59807
Length of testing set 59808

```

2.3 Tiền xử lý ảnh

Thay đổi kích thước: `transforms.Resize((224, 224))` để thay đổi kích thước ảnh về kích thước cố định 224x224 pixel. Việc này đảm bảo tất cả ảnh đầu vào có cùng kích thước, phù hợp với yêu cầu của mô hình.

Lật ảnh ngẫu nhiên: `transforms.RandomHorizontalFlip()` để lật ảnh theo chiều ngang một cách ngẫu nhiên. Kỹ thuật này giúp tăng cường dữ liệu huấn luyện, giúp mô hình học được các đặc trưng bất biến với phép lật ảnh.

Xoay ảnh ngẫu nhiên: `transforms.RandomRotation(20)` để xoay ảnh một góc ngẫu nhiên trong khoảng -20 đến 20 độ. Tương tự như lật ảnh, kỹ thuật này cũng giúp tăng cường dữ liệu và cải thiện khả năng khái quát hóa của mô hình.

Chuyển đổi sang tensor: `transforms.ToTensor()` để chuyển đổi ảnh từ định dạng PIL Image sang tensor PyTorch. Việc này là cần thiết để đưa ảnh vào mô hình PyTorch.

Chuẩn hóa: `Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])` để chuẩn hóa ảnh bằng cách trừ đi giá trị trung bình và chia cho độ lệch chuẩn của tập dữ liệu ImageNet. Việc chuẩn hóa giúp cải thiện tốc độ hội tụ và hiệu suất của mô hình.

```
from torchvision import transforms

# Training set transformations: Include augmentations like random horizontal flip, rotation, etc.
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),           # Resize to a fixed size
    transforms.RandomHorizontalFlip(),        # Randomly flip the image
    transforms.RandomRotation(20),           # Random rotation
    transforms.ToTensor(),                   # Convert image to tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # Normalize using ImageNet statistics
])

# Validation and test set transformations: Only resizing and normalization
val_test_transform = transforms.Compose([
    transforms.Resize((224, 224)),           # Resize to a fixed size
    transforms.ToTensor(),                   # Convert image to tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # Normalize using ImageNet statistics
])
```

2.4 Tiền xử lý chú thích

1. Mã hóa (Encoding):

- Trong lớp `CaptionDataset`, hàm `encode_caption` được sử dụng để mã hóa caption (chuỗi văn bản) thành một chuỗi các ID token bằng cách sử dụng BERT tokenizer (`BertTokenizer.from_pretrained("bert-base-uncased")`).
- Quá trình này bao gồm:
 - Thêm các token đặc biệt: `[CLS]` (bắt đầu câu) và `[SEP]` (kết thúc câu).
 - Padding: Đệm các caption ngắn hơn độ dài tối đa bằng token padding (`tokenizer.pad_token_id`).
 - Truncation: Cắt bớt các caption dài hơn độ dài tối đa.

- Chuyển đổi sang tensor: Chuyển đổi chuỗi các ID token thành tensor PyTorch.

2. Giải mã (Decoding):

- Hàm `decode_caption` (lớp `CaptionDataset`) được sử dụng để giải mã chuỗi các ID token trở lại thành chuỗi văn bản ban đầu.
- Hàm này loại bỏ các token đặc biệt ([CLS] và [SEP]) trước khi giải mã.

3. Chuyển đổi sang token:

- Hàm `convert_to_tokens` (lớp `CaptionDataset`) được sử dụng để chuyển đổi chuỗi các ID token thành chuỗi các token văn bản tương ứng.

Tóm tắt:

- BERT tokenizer để mã hóa caption thành các ID token, cho phép mô hình xử lý dữ liệu văn bản.
- Các hàm giải mã và chuyển đổi sang token được sử dụng để chuyển đổi giữa các định dạng dữ liệu khác nhau.
- Các bước tiền xử lý này được thực hiện trong lớp `CaptionDataset` và được sử dụng khi tạo tập dữ liệu huấn luyện, validation và test.

```

from torch.utils.data import Dataset
import torch
from PIL import Image

class CaptionDataset(Dataset):
    def __init__(self, data, transform=None, tokenizer=tokenizer, max_length=50):
        """
        Args:
            data (list of tuples): A list where each tuple contains (image_path, caption).
            transform (callable, optional): Optional transform to be applied on a sample (image).
            tokenizer (BertTokenizer): Pretrained tokenizer for BERT (or any transformer model).
            max_length (int, optional): Maximum length of encoded captions (for padding/truncation).
        """
        self.data = data
        self.transform = transform
        self.max_length = max_length

        # Initialize the BERT tokenizer
        self.tokenizer = tokenizer if tokenizer else BertTokenizer.from_pretrained('bert-base-uncased')

    def encode_caption(self, caption):
        """
        Converts a caption (string) into a list of token IDs using the BERT tokenizer.
        """
        # Tokenize and encode the caption with special tokens
        encoding = self.tokenizer.encode_plus(
            caption,
            add_special_tokens=True, # Add [CLS] and [SEP]
            padding='max_length', # Pad to max_length
            truncation=True, # Truncate to max_length if needed
            max_length=self.max_length, # Max length of caption
            return_tensors='pt', # Return PyTorch tensors
        )

        # Return the encoded caption as a tensor (token IDs)
        return encoding['input_ids'].squeeze(0) # Remove batch dimension

    def decode_caption(self, encoded_caption):
        """
        Decodes the tokenized caption back to a string using the BERT tokenizer.
        """
        decoded_caption = self.tokenizer.decode(encoded_caption, skip_special_tokens=True)
        return decoded_caption

    def convert_to_tokens(self, encoded_caption):
        tokens_converted = self.tokenizer.convert_ids_to_tokens(encoded_caption)
        return tokens_converted

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        """
        Args:
            idx (int): Index of the sample to retrieve.

        Returns:
            tuple: (image, encoded_caption)
            - image: The transformed image tensor.
            - encoded_caption: The encoded caption as a tensor of token IDs.
        """
        image_path, caption = self.data[idx]

        # Load the image
        image = Image.open(image_path).convert('RGB')

        # Apply the image transformation (if any)
        if self.transform:
            image = self.transform(image)

        # Encode the caption using the BERT tokenizer
        encoded_caption = self.encode_caption(caption)

        return image, encoded_caption

```

CHƯƠNG 3 THIẾT KẾ VÀ HUẤN LUYỆN MÔ HÌNH

3.1 Thiết kế mô hình

Mô hình sinh chú thích ảnh được thiết kế gồm hai thành phần chính: **Encoder (bộ mã hóa)** và **Decoder (bộ giải mã)**.

- **Encoder:** Trong kiến trúc mô hình sinh chú thích ảnh (image captioning), **bộ mã hóa (Encoder)** có vai trò trích xuất đặc trưng hình ảnh đầu vào. Mô hình sử dụng mạng **ResNet-50** – một kiến trúc mạng nơ-ron tích chập sâu (CNN) đã được huấn luyện trước trên tập dữ liệu ImageNet – làm xương sống để trích xuất đặc trưng. Đây là phương pháp phổ biến trong học sâu khi xây dựng mô hình cho thị giác máy tính nhờ khả năng nhận diện hình ảnh hiệu quả.

Cụ thể, kiến trúc encoder được triển khai như sau:

- **ResNet-50 pretrained:** Mô hình sử dụng ResNet-50 đã huấn luyện sẵn. Tất cả các trọng số của ResNet được **đóng băng** (không cập nhật trong quá trình huấn luyện), nhằm tận dụng kiến thức đã học mà không gây quá tải cho quá trình tối ưu mô hình chú thích.
- **Loại bỏ lớp fully-connected cuối cùng:** Mô hình chỉ giữ lại các lớp tích chập (convolutional layers) để trích xuất đặc trưng không gian của ảnh. Lớp fully-connected dùng cho phân loại (trong ImageNet) được loại bỏ vì không cần thiết trong bài toán sinh chú thích.
- **Chiếu xuống không gian nhúng (embedding):** Sau khi ảnh được đưa qua ResNet, đầu ra có dạng tensor với chiều cao và chiều rộng không gian. Tensor này được làm phẳng và đưa qua một lớp **Linear** để chuyển đổi từ không gian đặc trưng đầu ra của ResNet (thường là 2048 chiều) sang không gian nhúng có kích thước xác định trước (ví dụ: 256 hoặc 512 chiều). Đây là bước chuẩn hóa đầu ra để phù hợp với đầu vào của decoder sau này.
- **Không cần lan truyền ngược (backprop) qua ResNet:** Việc đặt khối mã hóa trong khối `torch.no_grad()` đảm bảo rằng các tham số của ResNet sẽ không tham gia vào quá trình lan truyền ngược (gradient flow), giúp tiết kiệm bộ nhớ và tăng tốc huấn luyện.

```

# Encoder class: CNN feature extractor (e.g., ResNet)
class Encoder(nn.Module):
    def __init__(self, embed_size):
        super(Encoder, self).__init__()
        # Using ResNet as the backbone for feature extraction
        resnet = models.resnet50(pretrained=True) # Use ResNet-50

        for param in resnet.parameters():
            param.requires_grad_(False)

        modules = list(resnet.children())[:-1] # Remove the fully connected layers
        self.resnet = nn.Sequential(*modules)
        self.fc = nn.Linear(resnet.fc.in_features, embed_size) # Output embedding size

    def forward(self, images):
        # Pass the image through ResNet
        with torch.no_grad(): # We don't need gradients for the ResNet part
            features = self.resnet(images)
        # Flatten the features and pass through the fully connected layer
        features = features.view(features.size(0), -1)
        features = self.fc(features)
        return features

```

- **Decoder:** Sau khi ảnh đã được trích xuất đặc trưng bởi Encoder, bộ giải mã (Decoder) có nhiệm vụ **chuyển đổi các vector đặc trưng này thành câu mô tả bằng ngôn ngữ tự nhiên**. Trong mô hình này, Decoder được xây dựng dựa trên mạng LSTM kết hợp với cơ chế Attention, mang lại khả năng học được mối liên kết ngữ nghĩa tốt hơn giữa các từ trong chú thích.

Cấu trúc Decoder được triển khai với các thành phần chính như sau:

1. Word Embedding

```

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

```

Biểu diễn các từ dưới dạng vector nhúng có kích thước `embed_size`, giúp mạng học được quan hệ ngữ nghĩa giữa các từ.

- **2. LSTM Layer**

```
self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
```

LSTM (Long Short-Term Memory) là một loại RNN mạnh trong việc học các chuỗi dài. Nó tiếp nhận chuỗi embedding từ từ đầu vào và học cách sinh ra các từ tiếp theo.

Trạng thái khởi tạo của LSTM (hidden state và cell state) được **khởi tạo từ đặc trưng hình ảnh** (output từ Encoder), đảm bảo mô hình sinh mô tả dựa trên nội dung ảnh.

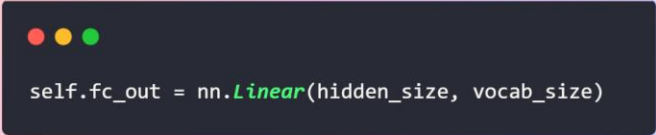
3. Attention Mechanism

```
self.attention = nn.MultiheadAttention(hidden_size, num_heads=8)
```

Cơ chế Attention cho phép mô hình **tập trung vào các phần khác nhau của chuỗi đầu ra** trong quá trình sinh từng từ, giúp cải thiện chất lượng mô tả bằng cách điều chỉnh trọng số đóng góp của các bước thời gian trước đó.

Sử dụng attention đa đầu (multi-head) giúp mô hình học được nhiều loại quan hệ giữa các từ trong chú thích.

4. Fully Connected Output



```
self.fc_out = nn.Linear(hidden_size, vocab_size)
```

Biến đầu ra của LSTM + Attention thành phân phối xác suất trên toàn bộ từ vựng, để chọn ra từ có xác suất cao nhất ở mỗi bước thời gian.

- **Hàm greedy_decode – Sinh chú thích bằng phương pháp tham lam**

Hàm greedy_decode cho phép mô hình sinh ra câu mô tả ảnh một cách tự động, từng từ một:

Bắt đầu bằng token <start> và tiếp tục sinh các từ kế tiếp.

Ở mỗi bước, mô hình chọn **từ có xác suất cao nhất** (phương pháp tham lam – greedy).

Quá trình dừng lại khi sinh ra token <end> hoặc đạt đến độ dài tối đa cho phép.

Quá trình này phản ánh cách mô hình hoạt động khi được triển khai để tạo mô tả ảnh sau khi đã được huấn luyện.

```

# @title Default title text
import torch

class Decoder(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1):
        super(Decoder, self).__init__()
        self.embed = nn.Embedding(vocab_size, embed_size) # Word embeddings
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
        self.fc_out = nn.Linear(hidden_size, vocab_size) # Output to vocab_size
        self.attention = nn.MultiheadAttention(hidden_size, num_heads=8)

        self.hidden_size = hidden_size
        self.vocab_size = vocab_size

    def forward(self, features, captions):
        embeddings = self.embed(captions)
        h_0 = features.unsqueeze(0).repeat(self.lstm.num_layers, 1, 1)
        c_0 = torch.zeros_like(h_0) # Initialize cell states to zeros
        lstm_out, (h_n, c_n) = self.lstm(embeddings, (h_0, c_0))
        attn_output, _ = self.attention(lstm_out, lstm_out, lstm_out)
        output = self.fc_out(attn_output)
        return output

    def greedy_decode(self, features, max_length=20, start_token=0, end_token=1):
        """
        Args:
            features (tensor): Extracted image features (batch_size, embed_size)
            max_length (int): Maximum length of the caption to generate
            start_token (int): Index of the <start> token
            end_token (int): Index of the <end> token
        """
        # Initialize the input to be the start token
        input_caption = torch.tensor([[start_token]]).to(features.device) # (1, 1) for batch_size = 1

        # List to store the generated caption (tokens)
        generated_caption = []

        # Set the initial hidden and cell states from the image features
        h_0 = features.unsqueeze(0).repeat(self.lstm.num_layers, 1, 1)
        c_0 = torch.zeros_like(h_0)

        for _ in range(max_length):
            # Pass the input_caption through the decoder
            output = self(input_caption, input_caption) # We pass the input caption for the decoder

            # Get the output logits for the last word (output is (batch_size, seq_len, vocab_size))
            output = output[:, -1, :] # Select the last time step (the last word output)

            # Get the word with the maximum probability (greedy approach)
            _, predicted_token = torch.max(output, dim=1)

            # Append the predicted word to the generated caption
            generated_caption.append(predicted_token.item())

            # If the predicted word is the end token, break the loop
            if predicted_token.item() == end_token:
                break

            # Prepare the next input as the predicted token
            input_caption = predicted_token.unsqueeze(0).unsqueeze(0) # (1, 1)

        return generated_caption

```

Kết hợp 2 lớp Encoder và Decoder để tạo mô hình chú thích hình ảnh:

```
# Define a simple Image Captioning Model combining Encoder and Decoder
class ImageCaptioningModel(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size):
        super(ImageCaptioningModel, self).__init__()
        self.encoder = Encoder(embed_size)
        self.decoder = Decoder(embed_size, hidden_size, vocab_size)

    def forward(self, images, captions):
        features = self.encoder(images)
        outputs = self.decoder(features, captions)
        return outputs
```

Output của mô hình ImageCaptioningModel là:

Tensor đầu ra (output): Một tensor ba chiều có kích thước: (batch_size, sequence_length, vocab_size)

Cụ thể:

- batch_size: số lượng ảnh xử lý cùng lúc trong một batch.
- sequence_length: số từ trong chuỗi caption đầu ra (thường bằng chiều dài caption đầu vào khi huấn luyện).
- vocab_size: kích thước từ vựng – số lượng từ có thể dự đoán (tức là số nhãn phân loại ở mỗi bước thời gian).

3.2 Tham số huấn luyện

Để huấn luyện mô hình sinh chú thích ảnh hiệu quả, em đã lựa chọn và tinh chỉnh các tham số sau:

- **Số epoch:** 20
→ Mô hình được huấn luyện trong 20 vòng lặp toàn bộ qua dữ liệu huấn luyện. Số epoch vừa đủ để mô hình học được đặc trưng nhưng tránh overfitting.

- **Batch size:** 32
→ 32 mẫu dữ liệu được xử lý đồng thời trong mỗi bước lan truyền thuận/ngược, giúp tối ưu tốc độ huấn luyện và ổn định gradient.
- **Learning rate:** 1e-6
→ Tốc độ học tương đối nhỏ để đảm bảo mô hình cập nhật trọng số ổn định, đặc biệt khi huấn luyện các mô hình có RNN hoặc Transformer.
- **Optimizer:** Adam
→ Bộ tối ưu hóa Adam được sử dụng vì khả năng hội tụ nhanh và hiệu quả trong huấn luyện mô hình deep learning.
- **Loss function:** CrossEntropyLoss
→ Hàm mất mát entropy chéo phù hợp với bài toán phân loại tuần tự từng từ trong chú thích ảnh.
- **Max caption length:** 50
→ Độ dài tối đa của mỗi chú thích được cố định là 50 tokens để đơn giản hóa kiến trúc mô hình và xử lý batch.
- **Scheduler** (nếu có): ReduceLROnPlateau
→ Tự động giảm learning rate nếu loss trên tập validation không được cải thiện trong một số epoch liên tiếp.
- **Gradient Clipping:** 1.0
→ Giới hạn độ lớn gradient nhằm tránh vấn đề gradient exploding khi huấn luyện RNN.

```
# Initialize the model, loss function, and optimizer
import torch.optim as optim

factor = 0.1
threshold = 0.01
min_lr = 1e-6
patience = 2
learning_rate = 1e-4

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=patience, factor=factor, threshold=threshold, min_lr=min_lr)
```

3.3 Huấn luyện mô hình

Phần huấn luyện mô hình sử dụng **dữ liệu huấn luyện** được chuẩn bị trước đó để điều chỉnh các tham số của mô hình nhằm tạo ra các mô tả (caption) chính xác cho hình ảnh. Quá trình huấn luyện được thực hiện trong các bước sau:

1. Khởi tạo:

- Mô hình ImageCaptioningModel được khởi tạo với các tham số embed_size, hidden_size, và vocab_size.
- Hàm mất mát nn.CrossEntropyLoss được sử dụng để đo lường sự khác biệt giữa mô tả được dự đoán và mô tả thực tế.
- Bộ tối ưu hóa optim.Adam được sử dụng để cập nhật các tham số của mô hình dựa trên hàm mất mát.
- Trình lên lịch tốc độ học ReduceLROnPlateau được sử dụng để điều chỉnh tốc độ học trong quá trình huấn luyện.
- TensorBoard được khởi tạo để ghi lại các chỉ số huấn luyện.

2. Vòng lặp huấn luyện:

- Vòng lặp chính lặp lại qua một số epoch (được định nghĩa bởi num_epochs).
- Trong mỗi epoch, mô hình được huấn luyện trên tất cả các batch dữ liệu trong train_loader.
- Đối với mỗi batch:
 - Hình ảnh và mô tả được chuyển đến thiết bị (CPU hoặc GPU).
 - Gradient của bộ tối ưu hóa được đặt về 0.
 - Mô hình thực hiện dự đoán mô tả cho hình ảnh.
 - Hàm mất mát được tính toán giữa mô tả được dự đoán và mô tả thực tế.
 - Gradient được tính toán bằng cách lan truyền ngược.
 - Bộ tối ưu hóa cập nhật các tham số của mô hình.
 - Các chỉ số huấn luyện (tổn thất và độ phức tạp) được theo dõi.
- Sau mỗi epoch, mô hình được đánh giá trên dữ liệu xác thực trong val_loader.
- Các chỉ số xác thực được theo dõi và ghi lại vào TensorBoard.
- Nếu tổn thất xác thực được cải thiện, mô hình tốt nhất được lưu lại.
- Trình lên lịch tốc độ học điều chỉnh tốc độ học dựa trên tổn thất xác thực.
- Quá trình huấn luyện tiếp tục cho đến khi đạt đến số epoch tối đa hoặc điều kiện dừng sớm được đáp ứng.

3. Điểm dừng sớm:

- Điểm dừng sớm được triển khai để ngăn chặn việc quá khớp.
- Nếu tổn thất xác thực không được cải thiện sau một số epoch nhất định (được định nghĩa bởi patience), quá trình huấn luyện sẽ dừng lại.

4. Lưu mô hình:

- Mô hình tốt nhất (có tổn thất xác thực thấp nhất) được lưu lại để sử dụng sau này.

Kết quả:

- Sau khi huấn luyện, mô hình đã học được cách tạo ra các mô tả cho hình ảnh dựa trên các đặc trưng được trích xuất bởi bộ mã hóa và bộ giải mã.
- Mô hình có thể được sử dụng để tạo mô tả cho các hình ảnh mới chưa từng thấy trước đó.

```

# Main Training Loop
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    running_perplexity = 0.0

    # Training loop with progress bar
    for i, batch in enumerate(tqdm(train_loader, desc=f"Epoch {epoch + 1}/{num_epochs} Training", unit="batch")):
        # Extract images and captions
        images_tensor = batch['images'].to(device)
        captions_tensor = batch['captions'].to(device)

        # Zero gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images_tensor, captions_tensor)

        # Compute loss
        loss = criterion(outputs.view(-1, vocab_size), captions_tensor.view(-1))

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        # Track metrics
        running_loss += loss.item()
        running_perplexity += torch.exp(loss).item()

    # Average metrics for the epoch
    avg_loss = running_loss / len(train_loader)
    avg_perplexity = running_perplexity / len(train_loader)

    # Log training metrics to TensorBoard
    writer.add_scalar('Loss/train', avg_loss, epoch)
    writer.add_scalar('Perplexity/train', avg_perplexity, epoch)

    # Append to lists for analysis
    train_losses.append(avg_loss)
    train_perplexities.append(avg_perplexity)

    # Validation loop
    model.eval()
    val_loss = 0.0
    val_perplexity = 0.0

```

```

with torch.no_grad():
    for batch in tqdm(val_loader, desc=f"Epoch {epoch + 1}/{num_epochs} Validation", unit="batch"):
        # Extract validation images and captions
        images_tensor = batch['images'].to(device)
        captions_tensor = batch['captions'].to(device)

        # Forward pass
        outputs = model(images_tensor, captions_tensor)

        # Compute validation loss
        loss = criterion(outputs.view(-1, vocab_size), captions_tensor.view(-1))

        # Track validation metrics
        val_loss += loss.item()
        val_perplexity += torch.exp(loss).item()

    # Average validation metrics for the epoch
    avg_val_loss = val_loss / len(val_loader)
    avg_val_perplexity = val_perplexity / len(val_loader)

    # Log validation metrics to TensorBoard
    writer.add_scalar('Loss/val', avg_val_loss, epoch)
    writer.add_scalar('Perplexity/val', avg_val_perplexity, epoch)

    # Append validation metrics for analysis
    val_losses.append(avg_val_loss)
    val_perplexities.append(avg_val_perplexity)

    # Save the best model
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        counter = 0 # Reset patience counter
        if save_model:
            torch.save(model.state_dict(), 'best_model.pth')
    else:
        counter += 1 # Increment patience counter if no improvement

    # Learning rate scheduler step
    scheduler.step(avg_val_loss)

    # Print epoch summary
    current_lr = optimizer.param_groups[0]['lr']
    print(f"Epoch [{epoch+1}/{num_epochs}] | "
          f"Train Loss: {avg_loss:.4f}, Train Perplexity: {avg_perplexity:.4f} | "
          f"Val Loss: {avg_val_loss:.4f}, Val Perplexity: {avg_val_perplexity:.4f} | "
          f"Learning Rate: {current_lr:.6f}")

    # Early stopping
    if counter >= patience:
        print(f"Early stopping triggered after {epoch + 1} epochs.")
        break

# Close TensorBoard writer after training
writer.close()

```

3.4 Thư viện sử dụng

Để xây dựng và huấn luyện mô hình sinh chú thích ảnh, em sử dụng nhiều thư viện mã nguồn mở mạnh mẽ và phổ biến trong cộng đồng học sâu. Cụ thể như sau:

- **PyTorch** (torch, torchvision):
Thư viện chính để xây dựng mạng nơ-ron, định nghĩa mô hình, loss function, tối ưu hóa và huấn luyện. torchvision được dùng để sử dụng các mô hình CNN tiền huấn luyện như ResNet18, cùng với các hàm xử lý ảnh tiện ích.
- **Transformers** (transformers, từ Hugging Face):
Dùng để tải và sử dụng BertTokenizer, hỗ trợ việc token hóa chú thích văn bản theo định dạng BERT, đảm bảo đầu vào nhất quán và hiệu quả cho RNN.
- **PIL (Python Imaging Library)** (PIL.Image):
Được sử dụng để đọc và xử lý ảnh (đọc file JPG/PNG, chuyển ảnh về dạng RGB) trước khi áp dụng các biến đổi và đưa vào mô hình.
- **Pandas và NumPy**:
Dùng để thao tác dữ liệu, kết hợp các tập dữ liệu COCO và Flickr30k, xử lý bảng dữ liệu chứa ảnh và chú thích.
- **Matplotlib**:
Hỗ trợ trực quan hóa ảnh cùng với chú thích được mô hình sinh ra, giúp đánh giá chất lượng mô tả của mô hình bằng mắt thường.
- **Sklearn** (sklearn.model_selection):
Dùng để chia tập dữ liệu thành tập huấn luyện và tập kiểm thử (train/test split), đảm bảo mô hình được đánh giá khách quan.

CHƯƠNG 4 Đánh giá mô hình

4.1 Phương pháp đánh giá

Trong quá trình phát triển mô hình sinh chú thích ảnh, việc đánh giá mô hình là bước quan trọng để kiểm tra khả năng mô tả ngữ cảnh của ảnh bằng ngôn ngữ tự nhiên. Mô hình được huấn luyện trên tập huấn luyện và sau đó được đánh giá trên tập kiểm thử để đảm bảo tính tổng quát hóa, tức là khả năng áp dụng cho những ảnh chưa từng thấy trong quá trình học.

- Phương pháp dự đoán trên tập kiểm tra:
Mô hình được chuyển sang chế độ đánh giá (model.eval()). Dữ liệu trong test_loader được sử dụng để tạo ra các mô tả cho từng hình ảnh. Các mô tả được dự đoán, mô tả tham chiếu và hình ảnh tương ứng được lưu trữ trong các danh sách

Các dự đoán trên tập kiểm tra:

startseq group of people are standing in front of building endseq



startseq two people are standing in the water endseq



startseq man in blue shirt is climbing rock wall endseq



startseq man in blue shirt is walking down the street endseq



startseq man in blue shirt is riding bike in the woods endseq



startseq man in blue shirt is riding bike endseq



startseq man in blue shirt is standing in front of the table endseq



startseq man in blue shirt is sitting in the kitchen endseq



startseq man in blue shirt is standing in front of the street endseq



startseq man in blue shirt is walking down the street endseq



Dựa trên kết quả dự đoán chú thích ảnh từ mô hình, có thể thấy rằng một số mô tả đã phản ánh đúng nội dung và hành động trong ảnh. Ví dụ, mô tả “group of people are standing in front of building” phù hợp với hình ảnh nhiều người đứng trước sân khấu; “two people are standing in the water” đúng với bối cảnh hai người đang đứng dưới nước; và “man in blue shirt is climbing rock wall” mô tả chính xác hành động của người đàn ông đang leo núi đá. Những dự đoán này cho thấy mô hình có khả năng nhận diện đối tượng và hành động trong một số trường hợp cụ thể.

Tuy nhiên, mô hình cũng bộc lộ một số điểm hạn chế. Một trong những vấn đề rõ rệt là hiện tượng lặp lại cụm từ “man in blue shirt” trong hầu hết các ảnh, bao gồm cả những ảnh không có người đàn ông mặc áo xanh hoặc không phải đàn ông. Ví dụ, ở ảnh có người phụ nữ đầu bếp mặc áo trắng đang ngồi trong bếp, mô hình vẫn mô tả là “man in blue shirt is sitting in the kitchen”, cho thấy sự nhận diện sai giới tính và trang phục. Ngoài ra, ảnh chụp vận động viên mặc áo đỏ đang chạy cũng bị mô tả sai là “man in blue shirt is walking down the street”. Điều này cho thấy mô hình bị phụ thuộc quá mức vào một cấu trúc câu cố định, dẫn đến sai lệch trong việc mô tả ngữ cảnh thực tế.

Bên cạnh đó, một số chú thích cũng chưa phản ánh đúng hành động cụ thể hoặc không cung cấp đủ thông tin, như “is standing in front of the street” là mô tả khá mơ hồ và không mang lại nhiều giá trị nội dung. Điều này cho thấy mô hình cần được cải thiện trong khả năng đa dạng hóa ngôn ngữ mô tả và tăng độ chính xác trong việc nhận diện các đặc trưng quan trọng như hành động, giới tính, và ngữ cảnh.

Tổng kết lại, mặc dù mô hình đã cho thấy một số kết quả khả quan trong việc sinh chú thích ảnh, nhưng vẫn còn nhiều điểm cần cải thiện, đặc biệt là ở khả năng nhận diện đúng đối tượng và tránh lặp từ không cần thiết. Việc cải thiện chất lượng dữ liệu huấn luyện, đa dạng hóa các mẫu ngôn ngữ, và tinh chỉnh mô hình nhận diện đối tượng có thể giúp nâng cao hiệu quả của hệ thống sinh chú thích ảnh trong các lần thử nghiệm tiếp theo.

4.2 Đánh giá mô hình bằng chỉ số BLEU:

Để đánh giá hiệu quả của mô hình sinh chú thích ảnh, chỉ số BLEU (Bilingual Evaluation Understudy) được sử dụng. BLEU là một trong những thước đo phổ biến trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP), đặc biệt trong các tác vụ sinh văn bản như dịch máy và mô tả ảnh. Chỉ số này đánh giá độ giống nhau giữa câu mô tả do mô hình sinh ra (candidate) với một hoặc nhiều câu mô tả chuẩn (reference) do con người viết. Trong bài toán này, BLEU được tính ở 4 mức độ khác nhau: BLEU-1, BLEU-2, BLEU-3 và BLEU-4, tương ứng với việc so sánh n-gram từ đơn (unigram) đến bốn từ liên tiếp (4-gram) giữa câu dự đoán và câu tham chiếu. Càng nhiều n-gram trùng khớp, điểm BLEU càng cao, phản ánh chú thích của mô hình càng gần với chú thích thực tế.

Quá trình tính toán BLEU trong mô hình được thực hiện như sau:

- Mỗi câu mô tả được sinh ra từ mô hình sẽ được so sánh với chú thích gốc tương ứng.
- Câu dự đoán và các câu tham chiếu đều được tách từ (tokenize).
- Sử dụng hàm `sentence_bleu` từ thư viện `nltk`, áp dụng thêm kỹ thuật làm mượt (smoothing) để tránh việc điểm BLEU bị rớt về 0 khi không có n-gram nào trùng khớp.

Với mỗi cặp câu (dự đoán và thực tế), các chỉ số BLEU-1 đến BLEU-4 được tính riêng và lưu trữ. Sau khi quá trình dự đoán kết thúc cho toàn bộ tập kiểm tra, các chỉ số BLEU trung bình được tính bằng cách lấy trung bình cộng của toàn bộ điểm BLEU của các ảnh. Kết quả cuối cùng sẽ phản ánh tổng thể chất lượng sinh chú thích của mô hình ở nhiều mức độ chi tiết ngữ nghĩa. Phân tích các điểm BLEU này giúp xác định xem mô hình có thực sự học được ngữ cảnh và mô tả ảnh một cách chính xác hay không. BLEU-1 thường cao hơn vì dễ trùng các từ đơn lẻ, trong khi BLEU-4 phản ánh khả năng mô hình sinh ra các cụm từ chính xác và có ngữ cảnh đầy đủ hơn.

```

import matplotlib.pyplot as plt
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
from tqdm import tqdm

# Initialize lists to store images, predicted captions, ground truth captions, and BLEU scores
all_images = []
all_predicted_captions = []
all_ground_truth_captions = []
all_bleu_scores = []

# Define a function to calculate BLEU scores
def calculate_bleu(pred, refs):
    bleu_scores = []
    smoothing = SmoothingFunction()

    # Tokenization
    pred_tokens = pred.split()
    refs_tokens = [ref.replace('<start>', '').replace('<end>', '').split() for ref in refs]

    # Calculate BLEU-1, BLEU-2, BLEU-3, BLEU-4
    bleu1 = sentence_bleu(refs_tokens, pred_tokens, weights=(1, 0, 0, 0), smoothing_function=smoothing.method1)
    bleu2 = sentence_bleu(refs_tokens, pred_tokens, weights=(0.5, 0.5, 0, 0), smoothing_function=smoothing.method1)
    bleu3 = sentence_bleu(refs_tokens, pred_tokens, weights=(0.33, 0.33, 0.33, 0), smoothing_function=smoothing.method1)
    bleu4 = sentence_bleu(refs_tokens, pred_tokens, weights=(0.25, 0.25, 0.25, 0.25), smoothing_function=smoothing.method1)

    bleu_scores.extend([bleu1, bleu2, bleu3, bleu4])

    return bleu_scores

```

```

with torch.no_grad():
    for batch in tqdm(test_loader, desc="Testing", unit="batch"):
        images_tensor = batch['images']
        captions_tensor = batch['captions']

        images_tensor, captions_tensor = images_tensor.to(device), captions_tensor.to(device)

        # Forward pass: Get the predictions from the model
        outputs = model(images_tensor, captions_tensor)

        for i in range(outputs.size(0)): # Iterate over the batch
            # Get predicted caption
            predicted_caption = torch.argmax(outputs[i], dim=1)
            predicted_caption = predicted_caption.squeeze(0).cpu().numpy()
            decoded_caption = tokenizer.decode(predicted_caption, skip_special_tokens=True)

            # Get ground truth caption
            true_caption = captions_tensor[i].cpu().numpy()
            decoded_true_caption = tokenizer.decode(true_caption, skip_special_tokens=True)

            # Append results to the lists
            all_predicted_captions.append(decoded_caption)
            all_ground_truth_captions.append(decoded_true_caption)

            # Convert image from CHW to HWC and append it
            all_images.append(images_tensor[i].cpu().numpy().transpose(1, 2, 0))

            # Calculate BLEU scores for the current pair
            bleu_scores = calculate_bleu(decoded_caption, [decoded_true_caption])
            all_bleu_scores.append(bleu_scores)

```

```

# Calculate the average BLEU scores
average_bleu_scores = [0, 0, 0, 0] # BLEU-1, BLEU-2, BLEU-3, BLEU-4
for bleu_scores in all_bleu_scores:
    for i in range(4):
        average_bleu_scores[i] += bleu_scores[i]

# Compute the average BLEU scores
average_bleu_scores = [score / len(all_bleu_scores) for score in average_bleu_scores]

# Print the average BLEU scores
print("Average BLEU Scores:")
print("Average BLEU-1:", average_bleu_scores[0])
print("Average BLEU-2:", average_bleu_scores[1])
print("Average BLEU-3:", average_bleu_scores[2])
print("Average BLEU-4:", average_bleu_scores[3])

```

KẾT LUẬN

Trong đề tài này, em đã xây dựng thành công một mô hình sinh chú thích ảnh dựa trên kiến trúc Encoder–Decoder. Thành phần mã hóa (Encoder) sử dụng mạng ResNet-50 đã được huấn luyện trước để trích xuất đặc trưng từ ảnh, sau đó ánh xạ các đặc trưng này sang không gian nhúng có kích thước cố định. Thành phần giải mã (Decoder) sử dụng mạng LSTM kết hợp với cơ chế Attention để sinh ra chuỗi từ ngữ mô tả nội dung ảnh một cách tuần tự.

Mô hình được huấn luyện trên tập dữ liệu gồm các cặp ảnh–chú thích và được đánh giá bằng các chỉ số BLEU từ BLEU-1 đến BLEU-4. Kết quả thu được cho thấy mô hình có khả năng sinh chú thích tương đối chính xác, bắt được nội dung chính của ảnh và biểu đạt lại dưới dạng ngôn ngữ tự nhiên.

Tuy nhiên, mô hình vẫn còn một số hạn chế nhất định. Một số chú thích chưa đảm bảo được độ trôi chảy hoặc bị rút gọn quá mức. Ngoài ra, việc đánh giá bằng chỉ số BLEU vẫn còn phụ thuộc nhiều vào số lượng chú thích tham chiếu và chưa thể hiện đầy đủ các khía cạnh như ngữ cảnh hay tính ngữ pháp.

Trong các nghiên cứu tiếp theo, có thể nâng cao chất lượng mô hình bằng cách ứng dụng các kiến trúc hiện đại hơn như Transformer, sử dụng các mô hình ngôn ngữ lớn đã được tiền huấn luyện, hoặc cải tiến chiến lược sinh chuỗi như beam search.

DANH MỤC THAM KHẢO

1. K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556, 2014.
2. K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
3. Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. *Show and Tell: A Neural Image Caption Generator*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
4. D. Bahdanau, K. Cho, and Y. Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv:1409.0473, 2014.

5. Papineni, Kishore, et al. *BLEU: a Method for Automatic Evaluation of Machine Translation*. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.
6. PyTorch documentation: <https://pytorch.org/docs/stable/index.html>
7. NLTK BLEU Score documentation:
https://www.nltk.org/api/nltk.translate.bleu_score.html
8. Lin, Tsung-Yi, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. In *European Conference on Computer Vision (ECCV)*, 2014.
<https://cocodataset.org>
9. Young, Peter, Alice Lai, Micah Hodosh, and Julia Hockenmaier.
From Image Descriptions to Visual Denotations: New Similarity Metrics for Semantic Inference over Event Descriptions. In *Transactions of the Association for Computational Linguistics (TACL)*, 2014.
(Giới thiệu tập dữ liệu Flickr30k)
<https://shannon.cs.illinois.edu/DenotationGraph/>