

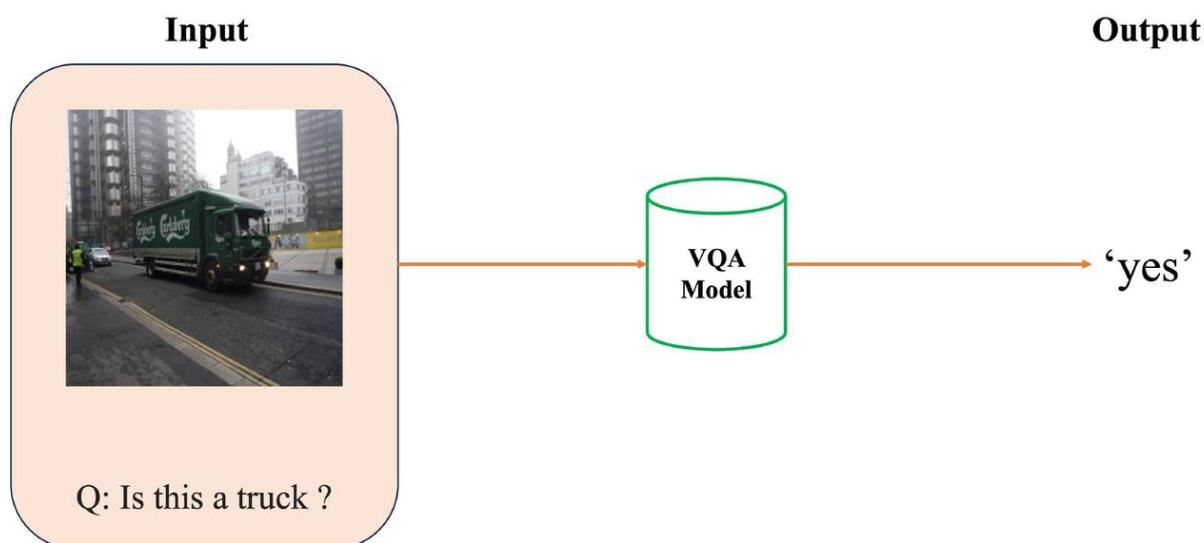
AI VIET NAM – AI COURSE 2024

Project: Visual Question Answering

Dinh-Thang Duong, Quang-Vinh Dinh

I. Giới thiệu

Visual Question Answering (VQA) là một bài toán phổ biến trong Machine Learning, ứng dụng các kỹ thuật liên quan đến hai lĩnh vực Computer Vision và Natural Language Processing để giải quyết bài toán này. Mục tiêu cốt lõi của VQA là phân tích một hình ảnh và trả lời câu hỏi về hình ảnh cho trước. Theo đó, bước đầu là phân tích thông tin đầu vào, bao gồm sử dụng các kỹ thuật xử lý hình ảnh và xử lý câu hỏi đặt ra bằng ngôn ngữ tự nhiên. Tiếp đến, mô hình VQA sẽ tổng hợp thông tin thu được từ phân tích hình ảnh và ngữ cảnh của câu hỏi để tạo ra một câu trả lời phù hợp. Vì vậy, một chương trình có độ chính xác cao cần xây dựng tốt cả hai thành phần này, đặt ra thách thức rất lớn trong việc giải quyết tốt bài toán hỏi đáp với ảnh.

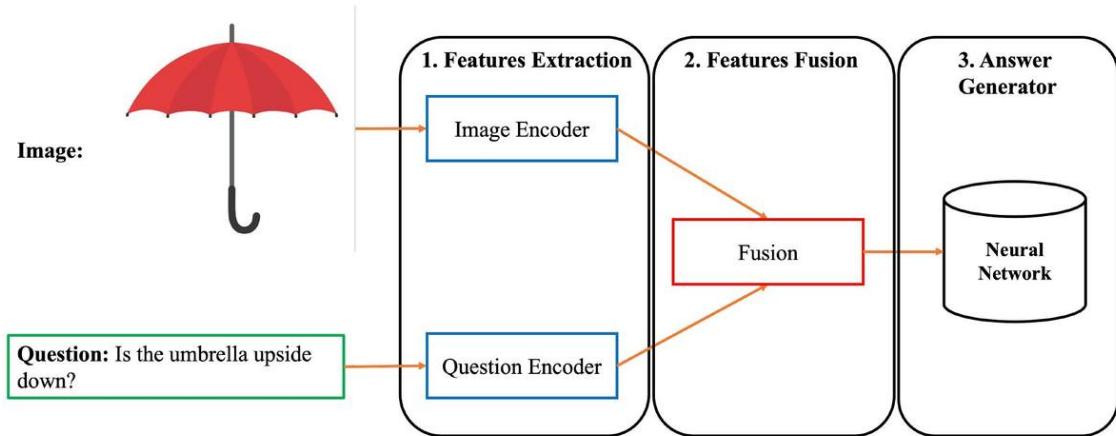


Hình 1: Minh họa về bài toán hỏi đáp trên ảnh - Visual Question Answering.

Theo đó, Input và Output của một mô hình VQA sẽ có dạng như sau:

- **Input:** Một cặp hình ảnh và câu hỏi bằng ngôn ngữ tự nhiên.
- **Output:** Câu trả lời cho câu hỏi về hình ảnh.

Ở project này, chúng ta sẽ cùng tìm hiểu và xây dựng một mô hình VQA với hướng tiếp cận cơ bản cũng như truyền thống của bài toán này. Theo đó, kiến trúc tổng quan của mô hình mà ta xây dựng như sau:



Hình 2: Minh họa kiến trúc của một mô hình VQA cơ bản.

Trong đó, ba thành phần chính gồm:

1. **Feature Extraction:** Trích xuất các vector đặc trưng phù hợp để đại diện cho dữ liệu ảnh và dữ liệu văn bản.
2. **Features Fusion:** Với hai đặc trưng từ hai nhánh dữ liệu, áp dụng kỹ thuật kết hợp cả hai loại thông tin này về một vector biểu diễn duy nhất.
3. **Answer Generator:** Dựa vector đặc trưng kết hợp vào mạng neural để dự đoán câu trả lời.

Dựa theo kiến trúc trên, ta sẽ cài đặt hai phiên bản gồm phiên bản dựa trên các mạng CNN+LSTM và phiên bản dựa trên các mạng thuộc họ transformer, cụ thể là ViT+RoBERTa. Ngoài ra, chúng ta cũng sẽ tìm hiểu hướng tiếp cận đang rất được quan tâm hiện nay, đó là sử dụng mô hình lớn. Cụ thể hơn, ta sẽ cài đặt mô hình lớn để có thể giải bài toán VQA trong project này.

II. Cài đặt chương trình

- **CNN + LSTM:** Trong phần này, chúng ta sẽ tiếp cận bài toán bằng cách phối hợp hai mô hình Deep Learning cơ bản dùng trong xử lý ảnh và văn bản là mạng CNN và LSTM.

1. **Tải bộ dữ liệu:** Cho bộ dữ liệu về Visual Question Answering có nội dung quan đến câu hỏi đáp dạng Yes/No, các bạn tải bộ dữ liệu **vqa_coco_dataset.zip** trong đường dẫn thuộc mục Datasets tại phần **IV**. Một số mẫu từ bộ dữ liệu khi được trực quan hóa sẽ có dạng như hình bên dưới:



Hình 3: Một vài mẫu dữ liệu trong bộ dữ liệu VQA dạng câu hỏi Yes/No.

2. **Import các thư viện cần thiết:** Chúng ta sẽ sử dụng thư viện PyTorch để xây dựng và huấn luyện mô hình deep learning. Thêm vào đó, vì làm việc liên quan đến dữ liệu ảnh và text, chúng ta sẽ sử dụng thư viện PIL cho ảnh và spacy, nltk cho text:

```

1 import torch
2 import torch.nn as nn
3 import torchtext
4 import os
5 import random
6 import numpy as np
7 import pandas as pd
8 import spacy
9 import timm
10 import matplotlib.pyplot as plt

```

```

11
12 from PIL import Image
13 from torch.utils.data import Dataset, DataLoader
14 from sklearn.model_selection import train_test_split
15 from torchtext.data.utils import get_tokenizer
16 from torchtext.vocab import build_vocab_from_iterator
17 from torchvision import transforms
18

```

3. **Cài đặt giá trị ngẫu nhiên cố định:** Để có thể tái tạo lại kết quả huấn luyện mô hình, ta sẽ cài đặt một giá trị ngẫu nhiên cố định cho các thư viện có liên quan đến việc tạo các giá trị ngẫu nhiên:

```

1 def set_seed(seed):
2     random.seed(seed)
3     np.random.seed(seed)
4     torch.manual_seed(seed)
5     torch.cuda.manual_seed(seed)
6     torch.cuda.manual_seed_all(seed)
7     torch.backends.cudnn.deterministic = True
8     torch.backends.cudnn.benchmark = False
9
10 seed = 59
11 set_seed(seed)

```

4. **Chia bộ dữ liệu train, val, test:** Vì bộ dữ liệu này đã được chia sẵn thành train, val, test, ta chỉ cần đọc dữ liệu lên từ file .txt cho trước từ bộ dữ liệu như sau:

```

1 # Load train data
2 train_data = []
3 train_set_path = './vaq2.0.TrainImages.txt'
4
5 with open(train_set_path, "r") as f:
6     lines = f.readlines()
7     for line in lines:
8         temp = line.split('\t')
9         qa = temp[1].split('?')
10
11         if len(qa) == 3:
12             answer = qa[2].strip()
13         else:
14             answer = qa[1].strip()
15
16         data_sample = {
17             'image_path': temp[0][-2],
18             'question': qa[0] + '?',
19             'answer': answer
20         }
21         train_data.append(data_sample)
22
23 # Load val data
24 val_data = []
25 val_set_path = './vaq2.0.DevImages.txt'
26
27 with open(val_set_path, "r") as f:
28     lines = f.readlines()

```

```

29     for line in lines:
30         temp = line.split('\t')
31         qa = temp[1].split('?')
32
33         if len(qa) == 3:
34             answer = qa[2].strip()
35         else:
36             answer = qa[1].strip()
37
38         data_sample = {
39             'image_path': temp[0][:-2],
40             'question': qa[0] + '?',
41             'answer': answer
42         }
43         val_data.append(data_sample)
44
45 # Load test data
46 test_data = []
47 test_set_path = './vaq2.0.TestImages.txt'
48
49 with open(test_set_path, "r") as f:
50     lines = f.readlines()
51     for line in lines:
52         temp = line.split('\t')
53         qa = temp[1].split('?')
54
55         if len(qa) == 3:
56             answer = qa[2].strip()
57         else:
58             answer = qa[1].strip()
59
60         data_sample = {
61             'image_path': temp[0][:-2],
62             'question': qa[0] + '?',
63             'answer': answer
64         }
65         test_data.append(data_sample)
66

```

5. **Xây dựng bộ từ vựng:** Chúng ta cần tiền xử lý text bằng cách biến đổi câu hỏi đầu vào thành các token bằng thư viện spacy và xây dựng bộ từ vựng cho model:

```

1 eng = spacy.load("en_core_web_sm")
2
3 def get_tokens(data_iter):
4     for sample in data_iter:
5         question = sample['question']
6         yield [token.text for token in eng.tokenizer(question)]
7
8 vocab = build_vocab_from_iterator(
9     get_tokens(train_data),
10    min_freq=2,
11    specials=['<pad>', '<sos>', '<eos>', '<unk>'],
12    special_first=True
13 )
14 vocab.set_default_index(vocab['<unk>'])

```

6. **Xây dựng dictionary mapping classes:** Để thuận tiện trong việc chuyển đổi tên class (trong trường hợp này gồm 2 class là "yes" và "no"), ta tạo dictionary dùng để chuyển tên class thành mã số tương ứng và ngược lại. Cách làm như sau:

```

1 classes = set([sample['answer'] for sample in train_data])
2 label2idx = {
3     cls_name: idx for idx, cls_name in enumerate(classes)
4 }
5 idx2label = {
6     idx: cls_name for idx, cls_name in enumerate(classes)
7 }
8

```

7. **Xây dựng hàm tokenize:** Sau khi đã có bộ từ vựng cho model, ta xây dựng một hàm tokenize để biến câu hỏi đầu vào thành danh sách các token tương ứng trong bộ từ vựng:

```

1 def tokenize(question, max_seq_len):
2     tokens = [token.text for token in eng.tokenizer(question)]
3     sequence = [vocab[token] for token in tokens]
4     if len(sequence) < max_seq_len:
5         sequence += [vocab['<pad>']] * (max_seq_len - len(sequence))
6     else:
7         sequence = sequence[:max_seq_len]
8
9     return sequence
10

```

8. **Xây dựng class PyTorch dataset:** Chúng ta xây dựng class datasets cho bộ dữ liệu VQA như sau:

```

1 class VQADataset(Dataset):
2     def __init__(
3         self,
4         data,
5         label2idx,
6         max_seq_len=20,
7         transform=None,
8         img_dir='val2014-resised/',
9     ):
10         self.transform = transform
11         self.data = data
12         self.max_seq_len = max_seq_len
13         self.img_dir = img_dir
14         self.label2idx = label2idx
15
16     def __len__(self):
17         return len(self.data)
18
19     def __getitem__(self, index):
20         img_path = os.path.join(self.img_dir, self.data[index][
21             'image_path'])
22         img = Image.open(img_path).convert('RGB')
23         if self.transform:
24             img = self.transform(img)

```

```
25     question = self.data[index]['question']
26     question = tokenize(question, self.max_seq_len)
27     question = torch.tensor(question, dtype=torch.long)
28
29     label = self.data[index]['answer']
30     label = label2idx[label]
31     label = torch.tensor(label, dtype=torch.long)
32
33     return img, question, label
```

9. **Xây dựng PyTorch transform:** Để dữ liệu ảnh đầu vào được đồng bộ về kích thước cũng như thực hiện một số các phép tăng cường dữ liệu ảnh lên bộ dữ liệu, chúng ta sẽ khai báo PyTorch transforms như sau:

```

1 data_transform = {
2     'train': transforms.Compose([
3         transforms.Resize(size=(224, 224)),
4         transforms.CenterCrop(size=180),
5         transforms.ColorJitter(brightness=0.1, contrast=0.1,
6             saturation=0.1),
7         transforms.RandomHorizontalFlip(),
8         transforms.GaussianBlur(3),
9         transforms.ToTensor(),
10        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224,
11            0.225)),
12    ]),
13    'val': transforms.Compose([
14        transforms.Resize(size=(224, 224)),
15        transforms.ToTensor(),
16        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224,
17            0.225)),
18    ])
19 }

```

10. Khai báo datasets object cho ba bộ train, val, test:

```
1 train_dataset = VQADataset(  
2     train_data,  
3     label2idx=label2idx,  
4     transform=data_transform['train'])  
5 )  
6 val_dataset = VQADataset(  
7     val_data,  
8     label2idx=label2idx,  
9     transform=data_transform['val'])  
10 )  
11 test_dataset = VQADataset(  
12     test_data,  
13     label2idx=label2idx,  
14     transform=data_transform['val'])  
15 )
```

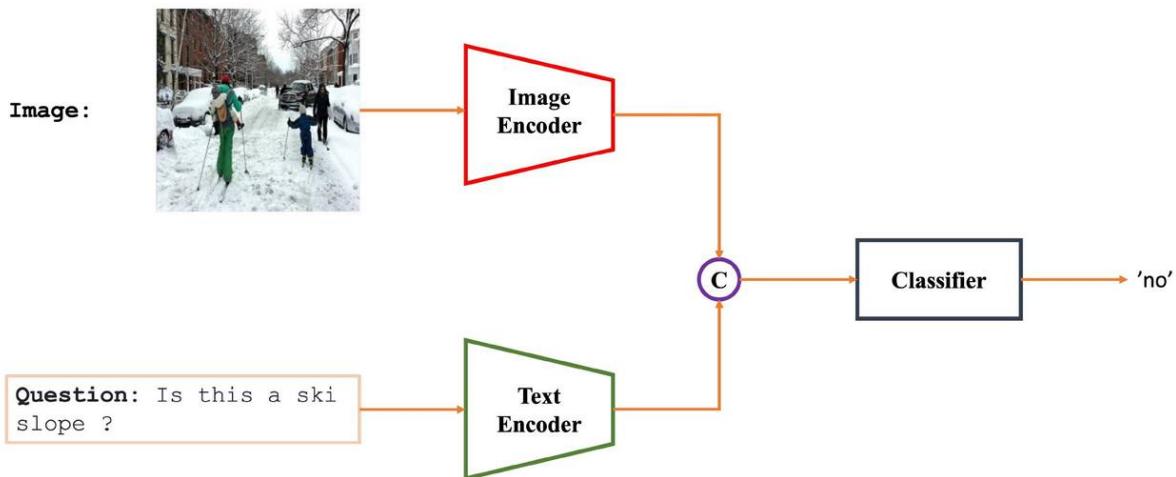
11. **Khai báo DataLoader:** Với ba object datasets trên, ta khai báo giá trị batch size và tao dataloader như sau:

```

1 train_batch_size = 256
2 test_batch_size = 32
3
4 train_loader = DataLoader(
5     train_dataset,
6     batch_size=train_batch_size,
7     shuffle=True
8 )
9 val_loader = DataLoader(
10    val_dataset,
11    batch_size=test_batch_size,
12    shuffle=False
13 )
14 test_loader = DataLoader(
15    test_dataset,
16    batch_size=test_batch_size,
17    shuffle=False
18 )

```

12. **Xây dựng model:** Trong phần này, ta sẽ dùng kiến trúc ResNet cho phần xử lý ảnh và kiến trúc BiLSTM cho phần xử lý text. Hướng tiếp cận này có thể được mô tả qua ảnh sau:



Hình 4: Ảnh mô tả pipeline cho hướng tiếp cận sử dụng mô hình CNN để trích xuất đặc trưng ảnh và mô hình Bi-LSTM để trích xuất đặc trưng văn bản.

Cụ thể, đối với ResNet, ta dùng thư viện [timm](#) để load kiến trúc ResNet18. Code của class model có nội dung như sau:

```

1 class VQAModel(nn.Module):
2     def __init__(self,
3      n_classes,
4      img_model_name,
5      embedding_dim,
6      n_layers=2,
7      hidden_size=256,
8

```

```

9         drop_p=0.2
10    ):
11        super(VQAModel, self).__init__()
12        self.image_encoder = timm.create_model(
13            img_model_name,
14            pretrained=True,
15            num_classes=hidden_size
16        )
17
18        for param in self.image_encoder.parameters():
19            param.requires_grad = True
20
21        self.embedding = nn.Embedding(len(vocab), embeddding_dim)
22        self.lstm1 = nn.LSTM(
23            input_size=embeddding_dim,
24            hidden_size=hidden_size,
25            num_layers=n_layers,
26            batch_first=True,
27            bidirectional=True,
28            dropout=drop_p
29        )
30
31        self.fc1 = nn.Linear(hidden_size * 3, hidden_size)
32        self.dropout = nn.Dropout(drop_p)
33        self.gelu = nn.GELU()
34        self.fc2 = nn.Linear(hidden_size, n_classes)
35
36    def forward(self, img, text):
37        img_features = self.image_encoder(img)
38
39        text_emb = self.embedding(text)
40        lstm_out, _ = self.lstm1(text_emb)
41
42        lstm_out = lstm_out[:, -1, :]
43
44        combined = torch.cat((img_features, lstm_out), dim=1)
45        x = self.fc1(combined)
46        x = self.gelu(x)
47        x = self.dropout(x)
48        x = self.fc2(x)
49
50    return x
51

```

Sau khi định nghĩa class, ta tiến hành khai báo model như sau:

```

1 n_classes = len(classes)
2 img_model_name = 'resnet18'
3 hidden_size = 256
4 n_layers = 2
5 embeddding_dim = 128
6 drop_p = 0.2
7 device = 'cuda' if torch.cuda.is_available() else 'cpu'
8
9 model = VQAModel(
10     n_classes=n_classes,

```

```

11     img_model_name=img_model_name ,
12     embeddding_dim=embeddding_dim ,
13     n_layers=n_layers ,
14     hidden_size=hidden_size ,
15     drop_p=drop_p
16 ).to(device)
17

```

Lưu ý rằng, trong trường hợp của bài toán VQA, việc thay đổi các giá trị tham số mô hình sẽ ảnh hưởng rất lớn đến kết quả của mô hình. Các giá trị gán mặc định trong code đã được tinh chỉnh cho phù hợp. Vì vậy, nếu muốn thay đổi, các bạn cần để ý để điều chỉnh cho phù hợp. Điều này cũng áp dụng tương tự cho các tham số thuộc phần optimizer.

13. **Xây dựng hàm train và evaluate:** Để huấn luyện mô hình, ta cần xây dựng hàm huấn luyện và đánh giá mô hình như sau:

```

1 def evaluate(model, dataloader, criterion, device):
2     model.eval()
3     correct = 0
4     total = 0
5     losses = []
6     with torch.no_grad():
7         for image, question, labels in dataloader:
8             image, question, labels = image.to(device), question.to(
device), labels.to(device)
9             outputs = model(image, question)
10            loss = criterion(outputs, labels)
11            losses.append(loss.item())
12            _, predicted = torch.max(outputs.data, 1)
13            total += labels.size(0)
14            correct += (predicted == labels).sum().item()
15
16    loss = sum(losses) / len(losses)
17    acc = correct / total
18
19    return loss, acc
20
21 def fit(
22     model,
23     train_loader,
24     val_loader,
25     criterion,
26     optimizer,
27     scheduler,
28     device,
29     epochs
30 ):
31     train_losses = []
32     val_losses = []
33
34     for epoch in range(epochs):
35         batch_train_losses = []
36
37         model.train()

```

```

38     for idx, (images, questions, labels) in enumerate(
39         train_loader):
40         images = images.to(device)
41         questions = questions.to(device)
42         labels = labels.to(device)
43
44         optimizer.zero_grad()
45         outputs = model(images, questions)
46         loss = criterion(outputs, labels)
47         loss.backward()
48         optimizer.step()
49
50         batch_train_losses.append(loss.item())
51
52     train_loss = sum(batch_train_losses) / len(batch_train_losses)
53
54     val_loss, val_acc = evaluate(
55         model, val_loader,
56         criterion, device
57     )
58     val_losses.append(val_loss)
59
60     print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal
61     loss: {val_loss:.4f}\tVal Acc: {val_acc}')
62
63     scheduler.step()
64
65     return train_losses, val_losses

```

14. Khai báo hàm loss, optimizer và scheduler:

```

1 lr = 1e-3
2 epochs = 50
3
4 scheduler_step_size = epochs * 0.8
5 criterion = nn.CrossEntropyLoss()
6
7 optimizer = torch.optim.Adam(
8     model.parameters(),
9     lr=lr
10 )
11 scheduler = torch.optim.lr_scheduler.StepLR(
12     optimizer,
13     step_size=scheduler_step_size,
14     gamma=0.1
15 )
16

```

15. Training: Tổng hợp tất cả các thành phần trên, ta gọi hàm fit() để bắt đầu quá trình huấn luyện mô hình VQA:

```

1 train_losses, val_losses = fit(
2     model,

```

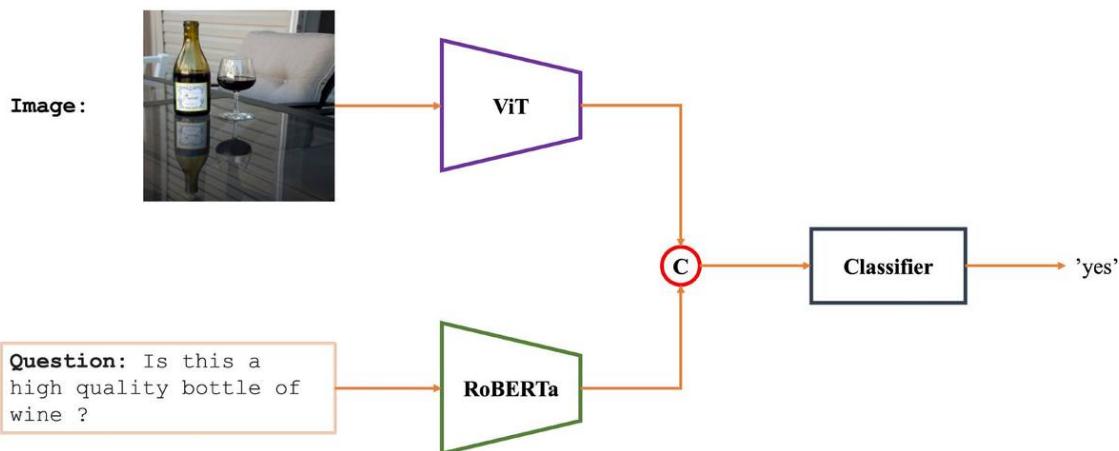
```
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     scheduler,
8     device,
9     epochs
10 )
11
```

16. **Evaluation:** Với mô hình đã huấn luyện, ta đưa vào đánh giá trên hai tập val và test như sau:

```
1 val_loss, val_acc = evaluate(
2     model,
3     val_loader,
4     criterion,
5     device
6 )
7 test_loss, test_acc = evaluate(
8     model,
9     test_loader,
10    criterion,
11    device
12 )
13
14 print('Evaluation on val/test dataset')
15 print('Val accuracy: ', val_acc)
16 print('Test accuracy: ', test_acc)
17
18 ## Evaluation results from author
19 # Val accuracy: 0.4989754098360656
20 # Test accuracy: 0.5084075173095944
21
```

Có thể thấy rằng, với hướng tiếp cận cơ bản trên, mô hình VQA chúng ta xây dựng không đạt được kết quả khả quan. Vì vậy ở phần sau, ta sẽ thử sử dụng các mô hình tốt hơn để cải thiện kết quả bài toán này.

- **VisionTransformers + RoBERTa:** Trong phần này, chúng ta sẽ tiếp cận bài toán bằng cách phối hợp hai mô hình Deep Learning thuộc họ mô hình Transformer, một kiến trúc mạng cực kỳ mạnh mẽ đang được sử dụng rộng rãi tại thời điểm này. Cụ thể, với ảnh đầu vào, ta dùng VisionTransformer (ViT). Đối với câu hỏi, ta dùng RoBERTa. Từ đó, trong mô hình VQA, dựa trên kết quả từ hai mô hình, ta sẽ kết hợp chúng để dự đoán câu trả lời. Hướng tiếp cận này có thể được mô tả như hình sau:



Hình 5: Ảnh mô tả pipeline cho hướng tiếp cận sử dụng các mô hình thuộc họ Transformer, cụ thể trong trường hợp này là ViT và RoBERTa.

Chúng ta sẽ triển khai hướng tiếp cận này thông qua các bước thực hiện dưới đây:

1. **Import các thư viện cần thiết:** Đối với transformer, ta sẽ sử dụng thư viện HuggingFace để gọi các mô hình và một số hàm xử lý cần thiết:

```

1 import torch
2 import torch.nn as nn
3 import os
4 import random
5 import numpy as np
6 import pandas as pd
7 import matplotlib.pyplot as plt
8
9 from PIL import Image
10 from torch.utils.data import Dataset, DataLoader
11 from torchvision import transforms
12 from transformers import ViTModel, ViTImageProcessor
13 from transformers import AutoTokenizer, RobertaModel
14

```

2. **Cài đặt giá trị ngẫu nhiên cố định:**

```

1 def set_seed(seed):
2     random.seed(seed)
3     np.random.seed(seed)
4     torch.manual_seed(seed)
5     torch.cuda.manual_seed(seed)

```

```

6     torch.cuda.manual_seed_all(seed)
7     torch.backends.cudnn.deterministic = True
8     torch.backends.cudnn.benchmark = False
9
10 seed = 59
11 set_seed(seed)

```

3. Chia bộ dữ liệu train, val, test:

```

1 # Load train data
2 train_data = []
3 train_set_path = './vaq2.0.TrainImages.txt'
4
5 with open(train_set_path, "r") as f:
6     lines = f.readlines()
7     for line in lines:
8         temp = line.split('\t')
9         qa = temp[1].split('?')
10
11     if len(qa) == 3:
12         answer = qa[2].strip()
13     else:
14         answer = qa[1].strip()
15
16     data_sample = {
17         'image_path': temp[0][-2],
18         'question': qa[0] + '?',
19         'answer': answer
20     }
21     train_data.append(data_sample)
22
23 # Load val data
24 val_data = []
25 val_set_path = './vaq2.0.DevImages.txt'
26
27 with open(val_set_path, "r") as f:
28     lines = f.readlines()
29     for line in lines:
30         temp = line.split('\t')
31         qa = temp[1].split('?')
32
33     if len(qa) == 3:
34         answer = qa[2].strip()
35     else:
36         answer = qa[1].strip()
37
38     data_sample = {
39         'image_path': temp[0][-2],
40         'question': qa[0] + '?',
41         'answer': answer
42     }
43     val_data.append(data_sample)
44
45 # Load test data
46 test_data = []
47 test_set_path = './vaq2.0.TestImages.txt'

```

```

48
49 with open(test_set_path, "r") as f:
50     lines = f.readlines()
51     for line in lines:
52         temp = line.split('\t')
53         qa = temp[1].split('?')
54
55         if len(qa) == 3:
56             answer = qa[2].strip()
57         else:
58             answer = qa[1].strip()
59
60         data_sample = {
61             'image_path': temp[0][-2],
62             'question': qa[0] + '?',
63             'answer': answer
64         }
65         test_data.append(data_sample)
66

```

4. Xây dựng dictionary mapping classes:

```

1 classes = set([sample['answer'] for sample in train_data])
2 label2idx = {
3     cls_name: idx for idx, cls_name in enumerate(classes)
4 }
5 idx2label = {
6     idx: cls_name for idx, cls_name in enumerate(classes)
7 }
8

```

5. Xây dựng class PyTorch dataset:

Với việc sử dụng hai mô hình liên quan đến transformers, chúng ta sẽ có một chút sự thay đổi trong việc triển khai class dataset như sau:

```

1 class VQADataset(Dataset):
2     def __init__(
3         self,
4         data,
5         label2idx,
6         img_feature_extractor,
7         text_tokenizer,
8         device,
9         transforms=None,
10         img_dir='val2014-resized'
11     ):
12         self.data = data
13         self.img_dir = img_dir
14         self.label2idx = label2idx
15         self.img_feature_extractor = img_feature_extractor
16         self.text_tokenizer = text_tokenizer
17         self.device = device
18         self.transforms = transforms
19
20     def __len__(self):
21         return len(self.data)

```

```

22
23     def __getitem__(self, index):
24         img_path = os.path.join(self.img_dir, self.data[index]['
25             image_path'])
26         img = Image.open(img_path).convert('RGB')
27
28         if self.transforms:
29             img = self.transforms(img)
30
31         if self.img_feature_extractor:
32             img = self.img_feature_extractor(images=img,
33             return_tensors="pt")
34             img = {k: v.to(self.device).squeeze(0) for k, v in img.
35             items()}
36
37             question = self.data[index]['question']
38             if self.text_tokenizer:
39                 question = self.text_tokenizer(
40                     question,
41                     padding="max_length",
42                     max_length=20,
43                     truncation=True,
44                     return_tensors="pt"
45                 )
46                 question = {k: v.to(self.device).squeeze(0) for k, v in
47                 question.items()}
48
49             label = self.data[index]['answer']
50             label = torch.tensor(
51                 self.label2idx[label],
52                 dtype=torch.long
53             ).to(self.device)
54
55             sample = {
56                 'image': img,
57                 'question': question,
58                 'label': label
59             }
59
60             return sample

```

Ở đây, ta bổ sung vào hai thành phần mới gồm `img_feature_extractor` và `text_tokenizer`, đây là hai hàm dùng để tiền xử lý dữ liệu đầu vào dành riêng cho VisionTransformers và RoBERTa. Vì kết quả của hai hàm này là tensor, chúng ta cũng cần truyền vào tham số `device` để chuyển đổi tensor về format tính toán của máy.

6. Xây dựng PyTorch transform:

```

1 data_transform = transforms.Compose([
2     transforms.Resize(size=(224, 224)),
3     transforms.CenterCrop(size=180),
4     transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation
5     =0.1),
6     transforms.RandomHorizontalFlip(),

```

```

6     transforms.GaussianBlur(3) ,
7 ])

```

7. Khai báo datasets object cho ba bộ train, val, test:

```

1 img_feature_extractor = ViTImageProcessor.from_pretrained("google/vit
2 -base-patch16-224")
3 text_tokenizer = AutoTokenizer.from_pretrained("roberta-base")
4 device = 'cuda' if torch.cuda.is_available() else 'cpu'
5
6 train_dataset = VQADataset(
7     train_data,
8     label2idx=label2idx,
9     img_feature_extractor=img_feature_extractor,
10    text_tokenizer=text_tokenizer,
11    device=device,
12    transforms=data_transform
13 )
14 val_dataset = VQADataset(
15     val_data,
16     label2idx=label2idx,
17     img_feature_extractor=img_feature_extractor,
18     text_tokenizer=text_tokenizer,
19     device=device
20 )
21 test_dataset = VQADataset(
22     test_data,
23     label2idx=label2idx,
24     img_feature_extractor=img_feature_extractor,
25     text_tokenizer=text_tokenizer,
26     device=device
27 )

```

8. Khai báo DataLoader:

```

1 train_batch_size = 256
2 test_batch_size = 32
3
4 train_loader = DataLoader(
5     train_dataset,
6     batch_size=train_batch_size,
7     shuffle=True
8 )
9 val_loader = DataLoader(
10    val_dataset,
11    batch_size=test_batch_size,
12    shuffle=False
13 )
14 test_loader = DataLoader(
15    test_dataset,
16    batch_size=test_batch_size,
17    shuffle=False
18 )

```

9. Xây dựng model:

Ta sẽ chia mô hình thành 3 phần gồm TextEncoder, VisualEncoder

và Classifier. Sau đó, ta kết hợp ba thành phần này lại để trở thành một mô hình hoàn chỉnh. Code triển khai như sau:

– **TextEncoder:**

```

1 class TextEncoder(nn.Module):
2     def __init__(self):
3         super(TextEncoder, self).__init__()
4         self.model = RobertaModel.from_pretrained("roberta-base")
5
6     def forward(self, inputs):
7         outputs = self.model(**inputs)
8
9     return outputs.pooler_output
10

```

– **VisualEncoder:**

```

1 class VisualEncoder(nn.Module):
2     def __init__(self):
3         super(VisualEncoder, self).__init__()
4         self.model = ViTModel.from_pretrained("google/vit-base-
5 patch16-224")
6
7     def forward(self, inputs):
8         outputs = self.model(**inputs)
9
10    return outputs.pooler_output
11

```

– **Classifier:**

```

1 class Classifier(nn.Module):
2     def __init__(
3         self,
4         hidden_size=512,
5         dropout_prob=0.2,
6         n_classes=2
7     ):
8         super(Classifier, self).__init__()
9         self.fc1 = nn.Linear(768 * 2, hidden_size)
10        self.dropout = nn.Dropout(dropout_prob)
11        self.gelu = nn.GELU()
12        self.fc2 = nn.Linear(hidden_size, n_classes)
13
14    def forward(self, x):
15        x = self.fc1(x)
16        x = self.gelu(x)
17        x = self.dropout(x)
18        x = self.fc2(x)
19
20    return x
21

```

– **VQAModel:** Tổng hợp lại 3 thành phần trên, ta được mô hình VQA hoàn chỉnh như sau:

```

1 class VQAModel(nn.Module):

```

```

2     def __init__(self,
3         visual_encoder,
4         text_encoder,
5         classifier
6     ):
7         super(VQAModel, self).__init__()
8         self.visual_encoder = visual_encoder
9         self.text_encoder = text_encoder
10        self.classifier = classifier
11
12
13    def forward(self, image, answer):
14        text_out = self.text_encoder(answer)
15        image_out = self.visual_encoder(image)
16
17        x = torch.cat((image_out, text_out), dim=1)
18        x = self.classifier(x)
19
20    return x
21
22
23    def freeze(self, visual=True, textual=True, clas=False):
24        if visual:
25            for n,p in self.visual_encoder.named_parameters():
26                p.requires_grad = False
27        if textual:
28            for n,p in self.text_encoder.named_parameters():
29                p.requires_grad = False
30        if clas:
31            for n,p in self.classifier.named_parameters():
32                p.requires_grad = False
33

```

Các bạn có thể thấy trong phần code này, chúng ta có triển khai thêm hàm `freeze()`. Hàm này có chức năng dùng để đóng băng các tham số của những pre-trained model, cụ thể ở đây là một trong ba thành phần chính của mô hình. Trong bài này, chúng ta sẽ chỉ cập nhật trọng số cho phần Classifier, các phần khác chúng ta sẽ đóng băng lại.

Sau khi định nghĩa xong, ta khai báo mô hình VQA:

```

1 n_classes = len(classes)
2 hidden_size = 256
3 dropout_prob = 0.2
4
5 text_encoder = TextEncoder().to(device)
6 visual_encoder = VisualEncoder().to(device)
7 classifier = Classifier(
8     hidden_size=hidden_size,
9     dropout_prob=dropout_prob,
10    n_classes=n_classes
11 ).to(device)
12
13 model = VQAModel(
14     visual_encoder=visual_encoder,
15     text_encoder=text_encoder,

```

```

16     classifier=classifier
17 ).to(device)
18 model.freeze()
19

```

10. **Xây dựng hàm train và evaluate:** Ta xây dựng hàm train và evaluate. Ở phần này, nội dung code của hai hàm sẽ có đôi chút sự khác biệt trong việc lấy dữ liệu từ dataloader vì chúng ta đã thay đổi cấu trúc data lúc định nghĩa PyTorch dataset:

```

1 def evaluate(model, dataloader, criterion):
2     model.eval()
3     correct = 0
4     total = 0
5     losses = []
6     with torch.no_grad():
7         for idx, inputs in enumerate(dataloader):
8             images = inputs['image']
9             questions = inputs['question']
10            labels = inputs['label']
11            outputs = model(images, questions)
12            loss = criterion(outputs, labels)
13            losses.append(loss.item())
14            _, predicted = torch.max(outputs.data, 1)
15            total += labels.size(0)
16            correct += (predicted == labels).sum().item()
17
18    loss = sum(losses) / len(losses)
19    acc = correct / total
20
21    return loss, acc
22
23 def fit(
24     model,
25     train_loader,
26     val_loader,
27     criterion,
28     optimizer,
29     scheduler,
30     epochs
31 ):
32     train_losses = []
33     val_losses = []
34
35     for epoch in range(epochs):
36         batch_train_losses = []
37
38         model.train()
39         for idx, inputs in enumerate(train_loader):
40             images = inputs['image']
41             questions = inputs['question']
42             labels = inputs['label']
43
44             optimizer.zero_grad()
45             outputs = model(images, questions)
46             loss = criterion(outputs, labels)

```

```
47     loss.backward()
48     optimizer.step()
49
50     batch_train_losses.append(loss.item())
51
52     train_loss = sum(batch_train_losses) / len(batch_train_losses)
53 )
54     train_losses.append(train_loss)
55
56     val_loss, val_acc = evaluate(
57         model, val_loader,
58         criterion
59     )
60     val_losses.append(val_loss)
61
62     print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal
63 loss: {val_loss:.4f}\tVal Acc: {val_acc}')
64
65     scheduler.step()
66
67     return train_losses, val_losses
```

11. Khai báo hàm loss, optimizer và scheduler:

```

1 lr = 1e-3
2 epochs = 50
3 scheduler_step_size = epochs * 0.8
4 criterion = nn.CrossEntropyLoss()
5
6 optimizer = torch.optim.Adam(
7     model.parameters(),
8     lr=lr
9 )
10 scheduler = torch.optim.lr_scheduler.StepLR(
11     optimizer,
12     step_size=scheduler_step_size,
13     gamma=0.1
14 )
15

```

12. Training: Với tất cả các thành phần có được, ta tiến hành huấn luyện mô hình:

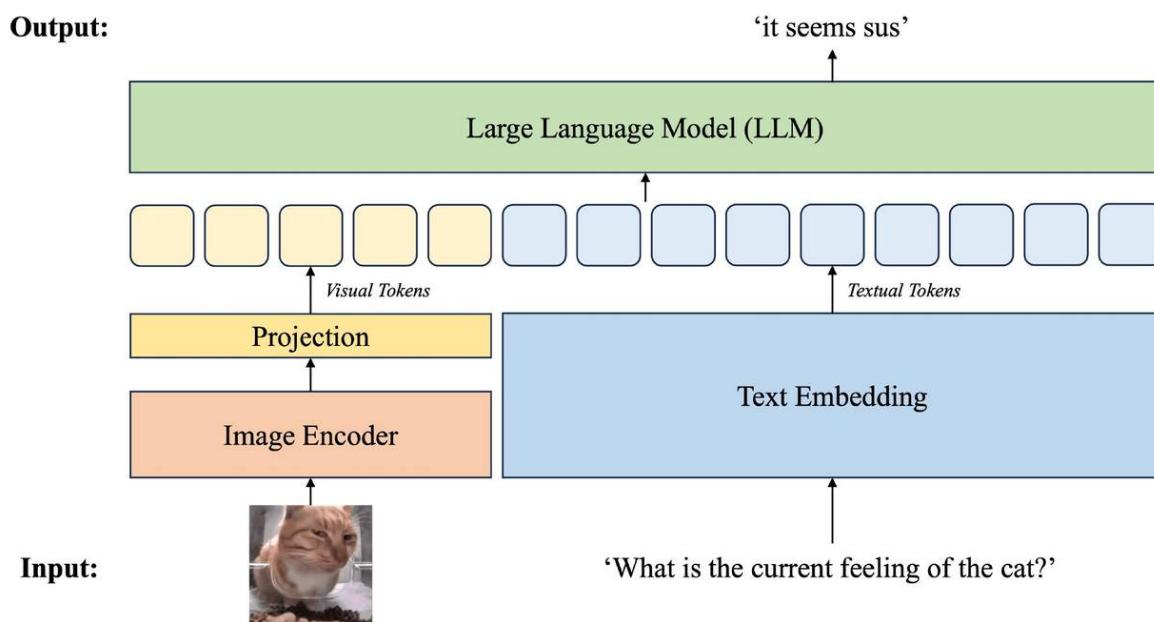
```
1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     scheduler,
8     epochs
9 )
10
```

13. **Evaluation:** Cuối cùng, ta đánh giá mô hình đã huấn luyện được trên hai tập val và test:

```
1 val_loss, val_acc = evaluate(
2     model,
3     val_loader,
4     criterion
5 )
6 test_loss, test_acc = evaluate(
7     model,
8     test_loader,
9     criterion
10 )
11
12 print('Evaluation on val/test dataset')
13 print('Val accuracy: ', val_acc)
14 print('Test accuracy: ', test_acc)
15
16 ## Evaluation results from author
17 # Val accuracy: 0.6664959016393442
18 # Test accuracy: 0.6533135509396637
19
```

Như vậy, ta có thể thấy một sự cải thiện rõ rệt của mô hình khi sử dụng các mô hình thuộc họ transformer để làm encoder cho hai kênh dữ liệu. Dù rằng chưa đạt được kết quả tốt nhất, song điều này cũng thể hiện được phần nào tiềm năng và hiệu quả mà các mô hình transformer mang lại trên bài toán VQA.

- **Vision-Language Models (VLMs):** Mô hình thị giác ngôn ngữ - Vision Language Models (VLMs) là thuật ngữ chỉ loại mô hình deep learning có khả năng xử lý đồng thời cả dữ liệu ảnh và văn bản. Hiện nay, các mô hình này đang rất được quan tâm bởi sự phát triển của các mô hình ngôn ngữ lớn - Large Language Models (LLMs). Theo đó, dựa trên ý tưởng tận dụng tri thức khổng lồ trong LLMs, các nhóm nghiên cứu này đã và đang phát triển, xây dựng một mô hình Large Vision Language Models (LVLMs) thông qua việc kết nối thông tin thị giác từ một Vision Encoder vào một LLM đủ tốt. Với cách thức này, các mô hình dạng LVLMs được kỳ vọng có thể giải quyết hầu hết tất cả mọi bài toán không chỉ liên quan đến dữ liệu văn bản mà còn về dữ liệu ảnh như Object Detection, Image Captioning...



Hình 6: Một kiến trúc cơ bản của Vision Language Models (VLMs) với hướng tiếp cận kết hợp với mô hình ngôn ngữ lớn (LLMs).

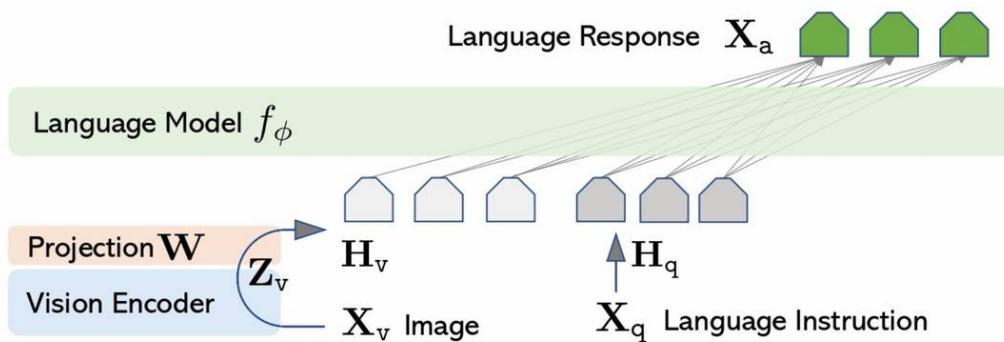
Để hiểu hơn về khả năng của VLM, ở phần này chúng ta sẽ tìm hiểu cách sử dụng một mô hình VLM để giải bài toán VQA. Code cài đặt như sau:

1. **Import các thư viện cần thiết:** Tương tự phần transformer, chúng ta cũng sẽ gọi các mô hình thị giác ngôn ngữ lớn trong thư viện HuggingFace:

```

1 import torch
2 import os
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6
7 from transformers import LlavaForConditionalGeneration
8 from transformers import AutoProcessor
9 from transformers import BitsAndBytesConfig
10 from transformers import GenerationConfig

```



Hình 7: Kiến trúc của mô hình LLaVA. Nguồn: [link](#).

4. **Xây dựng hàm tạo prompt:** Để sử dụng các mô hình lớn nói chung, chúng ta cần thiết kế và đưa vào mô hình một mô tả văn bản gọi là “prompt”. Có rất nhiều cách để thiết kế một mẫu prompt. Ở project VQA này, ta sẽ sử dụng một mẫu prompt đơn giản như sau:

```

1 def create_prompt(question):
2     prompt = f"""### INSTRUCTION:
3 Your task is to answer the question based on the given image. You can
4 only answer 'yes' or 'no'.
5 ### USER: <image>
6 {question}
7 ### ASSISTANT:"""
8     return prompt

```

5. **Khai báo các tham số tạo sinh:** Việc tạo sinh của một mô hình lớn bị ảnh hưởng bởi rất nhiều các tham số khác nhau. Trong bài này, ta sẽ cài đặt một số tham số như sau:

```

1 generation_config = GenerationConfig(
2     max_new_tokens=10,
3     do_sample=True,
4     temperature=0.1,
5     top_p=0.95,
6     top_k=50,
7     eos_token_id=model.config.eos_token_id,
8     pad_token=model.config.pad_token_id,
9 )

```

6. **Thực hiện dự đoán:** Cuối cùng, ta sẽ gọi mô hình để thực hiện dự đoán cho một mẫu dữ liệu trong bộ test của chúng ta bằng đoạn code sau:

```

1 idx = 0
2 question = test_data[idx]['question']
3 image_name = test_data[idx]['image_path']
4 image_path = os.path.join('val2014-resized', image_name)
5 label = test_data[idx]['answer']
6 image = Image.open(image_path)
7
8 prompt = create_prompt(question)
9 inputs = processor(prompt,

```

```

10             image ,
11             padding=True ,
12             return_tensors="pt").to(device)
13
14 output = model.generate(**inputs ,
15                         generation_config=generation_config)
16 generated_text = processor.decode(output[0] ,
17                                     skip_special_tokens=True)
18
19 plt.imshow(image)
20 plt.axis("off")
21 plt.show()
22 print(f"Question: {question}")
23 print(f"Label: {label}")
24 print(f"Prediction: {generated_text.split('### ASSISTANT: ')[-1]}")

```

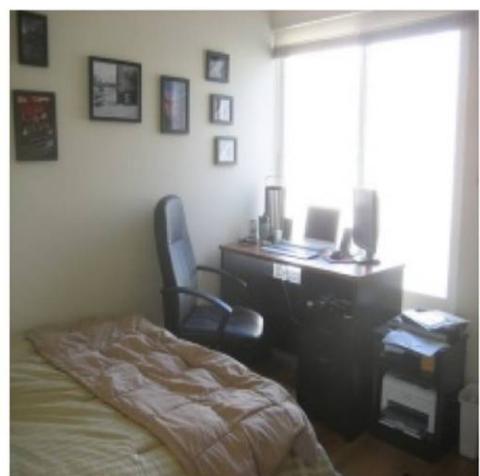


Question: Are there any trees visible ?

Groundtruth: no

Predicted: no

Trained ViT+RoBERTa Model.



Question: Are there any boxes in the room ?

Groundtruth: no

Prediction: No

Pre-trained LLaVA-7B Model.

Hình 8: Kết quả dự đoán của mô hình ViT+RoBERTa sau khi huấn luyện vs mô hình pre-trained LLaVA.

Mặc dù không cần phải trải qua thêm bất cứ bước training trên bộ dữ liệu mới hoặc cài đặt gì khác, song mô hình pre-trained LLaVA vẫn hoàn toàn có thể dự đoán chính xác một số mẫu dữ liệu trên bộ test. Điều này cho ta thấy tiềm năng và sức mạnh của các mô hình ngôn ngữ thị giác không chỉ trong bài toán VQA mà còn liên quan đến các ứng dụng khác trong trí tuệ nhân tạo.

III. Câu hỏi trắc nghiệm

1. Mục tiêu của bài toán Visual Question Answering là gì?
 - (a) Để tăng kích cỡ của tấm ảnh.
 - (b) Để trả lời câu hỏi dựa trên hình ảnh.
 - (c) Để tạo sinh ra hình ảnh dựa trên mô tả.
 - (d) Để tạo ra mô hình 3D từ hình ảnh 2D.
2. Trong VQA, mô hình thường nhận đầu vào là gì?
 - (a) Chỉ là hình ảnh.
 - (b) Chỉ là câu hỏi văn bản.
 - (c) Hình ảnh và câu hỏi văn bản.
 - (d) Âm thanh.
3. Visual Question Answering là sự kết hợp giữa 2 lĩnh vực nào sau đây?
 - (a) Robotics và Xử lý ngôn ngữ tự nhiên.
 - (b) Thị giác máy tính và học máy.
 - (c) Thị giác máy tính và xử lý ngôn ngữ tự nhiên.
 - (d) Học máy và robotics.
4. Trong VQA, câu trả lời có thể là gì?
 - (a) Chỉ là “có” hoặc “không”.
 - (b) Chỉ là một số nguyên.
 - (c) Có thể là một câu trả lời văn bản hoặc câu trả lời “có” hoặc “không”.
 - (d) Chỉ là một màu sắc.
5. Thành phần nào sau đây là một trong những thành phần chính của một mô hình VQA?
 - (a) Nhận diện giọng nói.
 - (b) Trích xuất đặc trưng từ hình ảnh.
 - (c) Dịch câu hỏi ngôn ngữ tự nhiên.
 - (d) Xử lý âm thanh.
6. Một số thách thức trong VQA bao gồm:
 - (a) Hiểu biết ngôn ngữ tự nhiên.
 - (b) Nhận dạng đối tượng trong hình ảnh.
 - (c) Không có thách thức nào.
 - (d) Cả a và b đều đúng.
7. Một hệ thống VQA thường xử lý câu hỏi ngôn ngữ tự nhiên như thế nào?

- (a) Bằng cách chuyển đổi câu hỏi đó thành hình ảnh.
- (b) Bằng cách áp dụng sentiment analysis cho tấm ảnh đó.
- (c) Dịch câu hỏi đó qua ngôn ngữ khác.
- (d) Bằng cách trích xuất đặc trưng câu hỏi đó bằng các kỹ thuật NLP.

8. Đoạn code sau đây dùng để làm gì?

```

1 eng = spacy.load("en_core_web_sm")
2
3 def get_tokens(data_iter):
4     for sample in data_iter:
5         question = sample['question']
6         yield [token.text for token in eng.tokenizer(question)]
7

```

- (a) Để load mô hình tiếng anh Spacy.
- (b) Tạo ra một danh sách các từ dựa trên danh sách token.
- (c) Thêm ký tự đặc biệt vào các câu hỏi.
- (d) Chia văn bản tiếng Anh thành những token.

9. Mục đích của đoạn code sau là gì?

```

1 vocab = build_vocab_from_iterator(
2     get_tokens(train_data),
3     min_freq=2,
4     specials=[ '<pad>', '<sos>', '<eos>', '<unk>' ],
5     special_first=True
6 )
7 vocab.set_default_index(vocab['<unk>'])
8

```

- (a) Xây dựng từ vựng từ tập train, bao gồm các token đặc biệt và token mặc định <unk> cho các từ chưa biết.
- (b) Dịch tập data huấn luyện sang một ngôn ngữ khác bằng cách sử dụng các token đặc biệt.
- (c) Sắp xếp các từ trong tập huấn luyện dựa trên tần suất xuất hiện của các từ.
- (d) Dào tạo một mô hình mới để hiểu các mẫu ngôn ngữ trong tập data huấn luyện.

10. Mô hình học máy nào thường được sử dụng cho bài toán VQA?

- (a) Mô hình dựa trên luật có sẵn.
- (b) Các mô hình CNN.
- (c) Mô hình linear regression.
- (d) Mô hình Decision Tree.

11. Thách thức trong bài toán VQA là gì?

- (a) Tăng độ phân giải của hình ảnh.
- (b) Sự mơ hồ trong câu hỏi ngôn ngữ tự nhiên.

- (c) Xử lý âm thanh.
(d) Tối ưu hóa phần cứng.
12. Output của mô hình VQA là:
(a) Bản báo cáo chi tiết.
(b) Con số cụ thể.
(c) Câu trả lời cho câu hỏi về hình ảnh.
(d) Một đồ thị.
13. Hệ thống VQA được ứng dụng trong việc:
(a) Chơi các video game.
(b) Hỗ trợ người dùng khiếm thị trong việc hiểu môi trường xung quanh.
(c) Lọc email rác.
(d) Xử lý âm thanh.
14. Dòng code sau đây có tác dụng gì?
1 image_encoder = timm.create_model(image_model, pretrained=True,
2 num_classes=hidden_dim)
2
(a) Tải pretrained hình ảnh từ thư viện timm.
(b) Chuyển đổi dữ liệu hình ảnh màu sang hình ảnh xám.
(c) Tăng độ phân giải của hình ảnh.
(d) Nén hình ảnh để xử lý nhanh hơn.
15. Output của image_encoder trong đoạn code sau là gì?
1 image_encoder = timm.create_model(image_model, pretrained=True,
2 num_classes=hidden_dim)
2
(a) Ảnh được biến đổi từ tập dữ liệu.
(b) Một vector embedding với số chiều là hidden_dim.
(c) Xác suất tấm ảnh đầu vào thuộc về từng class.
(d) Chiều của tấm ảnh đầu vào.
16. Điều KHÔNG làm ảnh hưởng đến độ chính xác của một mô hình VQA?
(a) Tốc độ Internet.
(b) Chất lượng của tập dữ liệu đầu vào.
(c) Độ nhiễu trong các tấm ảnh đầu vào.
(d) Độ phức tạp trong câu hỏi từ ngôn ngữ tự nhiên.

IV. Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
4. **Rubric:**

Mục	Kiến Thức	Dánh Giá
II.1.	<ul style="list-style-type: none"> - Kiến thức về việc giải quyết bài toán VQA. - Cách kết hợp mô hình CNN và LSTM để giải quyết bài toán VQA. 	<ul style="list-style-type: none"> - Nắm được các bước cơ bản giải quyết bài toán VQA. - Có thể cài đặt và huấn luyện mô hình giải quyết bài toán VQA sử dụng CNN và LSTM trong PyTorch.
II.2.	<ul style="list-style-type: none"> - Kiến thức về một số mô hình transformers. - Cách sử dụng mô hình transformers. - Kiến thức về việc sử dụng các mô hình thuộc họ transformers để giải quyết bài toán VQA. 	<ul style="list-style-type: none"> - Có thể cài đặt và huấn luyện mô hình giải quyết bài toán VQA sử dụng VisionTransformers và RoBERTa trong PyTorch.
II.3.	<ul style="list-style-type: none"> - Kiến thức về mô hình ngôn ngữ thị giác lớn (LVLMs). - Cách sử dụng mô hình LVLMs bằng thư viện HuggingFace. - Kiến thức về việc sử dụng mô hình pretrained LVLMs để giải bài toán VQA. 	<ul style="list-style-type: none"> - Có thể sử dụng mô hình thị giác ngôn ngữ lớn LLaVA để thực hiện bài toán VQA.

- *Hết* -