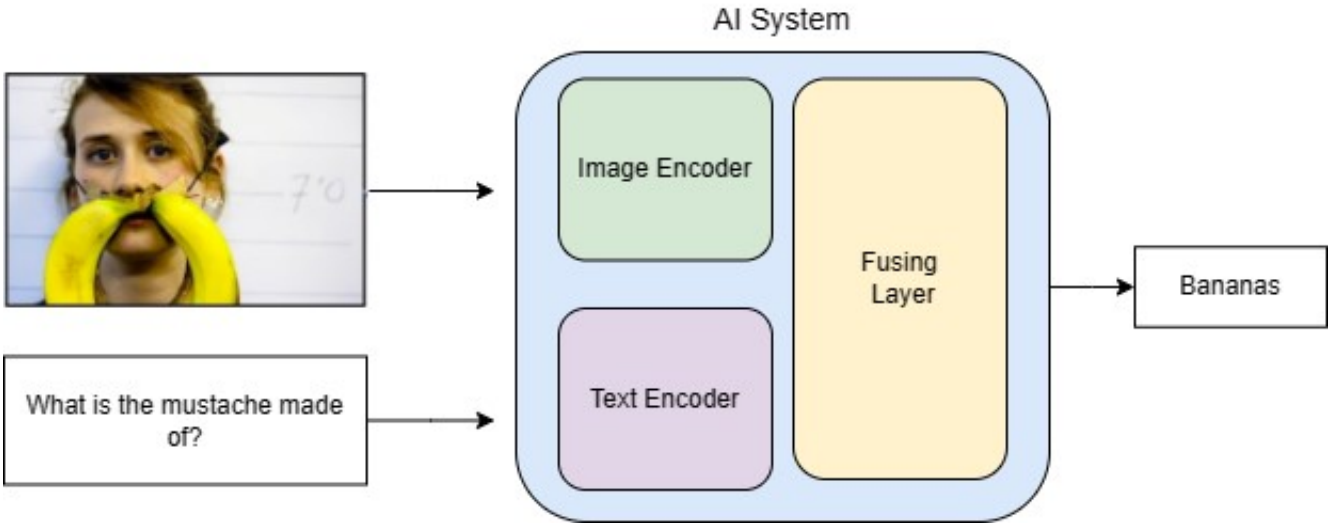
AI VIET NAM – COURSE 2023

# Visual Question Answering - Project

Ngày 10 tháng 12 năm 2023

## Phần I: Giới thiệu

**Visual Question Answering (VQA)** là một bài toán phổ biến trong Machine Learning, ứng dụng các kĩ thuật liên quan từ hai lĩnh vực Computer Vision và Natural Language Processing. Khái niệm cốt lõi của bài toán này là phân tích một hình ảnh và trả lời câu hỏi về hình ảnh đó. Bước đầu là phân tính thông tin đầu vào, bao gồm sử dụng các kĩ thuật xử lý hình ảnh và xử lý câu hỏi đặt ra bằng ngôn ngữ tự nhiên. Sau đó, hệ thống VQA sẽ 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.




Trong project này, chúng ta sẽ xây dựng một chương trình VQA sử dụng mô hình CNN cho hình ảnh và LSTM cho xử lý ngôn ngữ. Input và output của chương trình như sau:

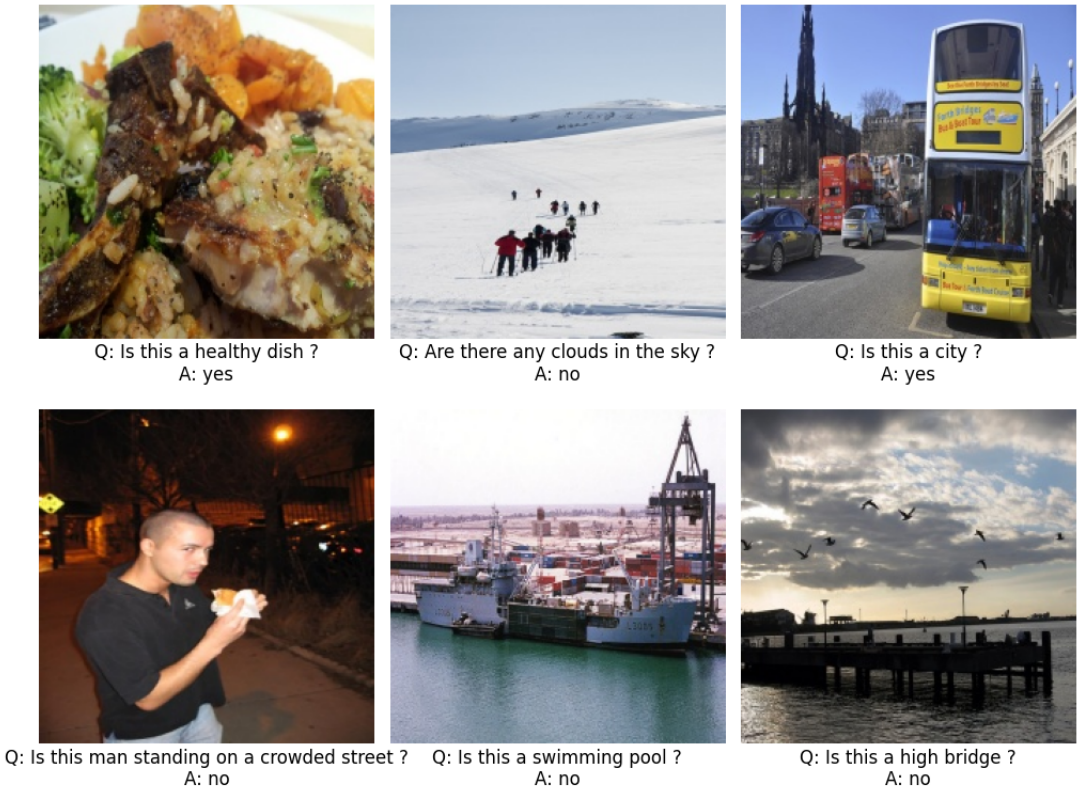
- **Input:** Một cặp hình ảnh và câu hỏi bằng ngôn ngữ tự nhiên.
- **Output:** Câu trả lời cho câu hỏi về hình ảnh.

# Phần II: Cài đặt chương trình

## A. Phần lập 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. **Download dataset:** 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 này tại đây. Một số mẫu từ bộ dữ liệu khi được trực quan hóa sẽ có dạng như hình bên dưới:




Hình 1: 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:

```
import torch
import torch.nn as nn
import os
import pandas as pd
import matplotlib.pyplot as plt

from PIL import Image
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import torchtext
from torchtext.data.utils import get_tokenizer
```

```
from torchtext.vocab import build_vocab_from_iterator
import spacy

```

3. **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:

```
# Load train data
train_data = []
train_set_path = './vaq2.0.TrainImages.txt'

with open(train_set_path, "r") as f:
    lines = f.readlines()
    for line in lines:
        temp = line.split('\t')
        qa = temp[1].split('?')

        if len(qa) == 3:
            answer = qa[2].strip()
        else:
            answer = qa[1].strip()

        data_sample = {
            'image_path': temp[0][:-2],
            'question': qa[0] + '?',
            'answer': answer
        }
        train_data.append(data_sample)

# Load val data
val_data = []
val_set_path = './vaq2.0.DevImages.txt'

with open(val_set_path, "r") as f:
    lines = f.readlines()
    for line in lines:
        temp = line.split('\t')
        qa = temp[1].split('?')

        if len(qa) == 3:
            answer = qa[2].strip()
        else:
            answer = qa[1].strip()

        data_sample = {
            'image_path': temp[0][:-2],
            'question': qa[0] + '?',
            'answer': answer
        }
        val_data.append(data_sample)

# Load test data
test_data = []
test_set_path = './vaq2.0.TestImages.txt'

with open(test_set_path, "r") as f:
    lines = f.readlines()
    for line in lines:
        temp = line.split('\t')
        qa = temp[1].split('?')

```

```
        if len(qa) == 3:
            answer = qa[2].strip()
        else:
            answer = qa[1].strip()

        data_sample = {
            'image_path': temp[0][:-2],
            'question': qa[0] + '?',
            'answer': answer
        }
        test_data.append(data_sample)

```

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

```
eng = spacy.load("en_core_web_sm") # Load the English model to tokenize
    English text

def get_tokens(data_iter):
    for sample in data_iter:
        question = sample['question']
        yield [token.text for token in eng.tokenizer(question)]


vocab = build_vocab_from_iterator(
    get_tokens(train_data),
    min_freq=2,
    specials= ['<pad>', '<sos>', '<eos>', '<unk>'],
    special_first=True
)
vocab.set_default_index(vocab['<unk>'])

```

5. **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:

```
classes = set([sample['answer'] for sample in train_data])
classes_to_idx = {
    cls_name: idx for idx, cls_name in enumerate(classes)
}
idx_to_classes = {
    idx: cls_name for idx, cls_name in enumerate(classes)
}

```

6. **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:

```
def tokenize(question, max_sequence_length):
tokens = [token.text for token in eng.tokenizer(question)]
sequence = [vocab[token] for token in tokens]
if len(sequence) < max_sequence_length:
    sequence += [vocab['<pad>']] * (max_sequence_length - len(sequence))
else:
    sequence = sequence[:max_sequence_length]

return sequence

```

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

```
class VQADataset(Dataset):
    def __init__(
        self,
        data,
        classes_to_idx,
        max_seq_len=30,
        transform=None,
        root_dir='/content/val2014-resised/'
    ):
        self.transform = transform
        self.data = data
        self.max_seq_len = max_seq_len
        self.root_dir = root_dir
        self.classes_to_idx = classes_to_idx

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

    def __getitem__(self, index):
        img_path = os.path.join(self.root_dir, self.data[index]['image_path'
    ])
        img = Image.open(img_path).convert('RGB')
        if self.transform:
            img = self.transform(img)

        question = self.data[index]['question']
        question = tokenize(question, self.max_seq_len)
        question = torch.tensor(question, dtype=torch.long)

        label = self.data[index]['answer']
        label = classes_to_idx[label]
        label = torch.tensor(label, dtype=torch.long)

        return img, question, label
```

8. **Xây dựng hàm tiền xử lý ảnh (transforms):** Để dữ liệu ảnh đầu vào được đồng bộ về kích thước cũng như đơn giản hóa, chúng ta sẽ xây dựng hàm tiền xử lý ảnh như sau:

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
])

```

Các kỹ thuật được áp dụng: resize ảnh, đổi về tensor và chuẩn hóa giá trị pixel.

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

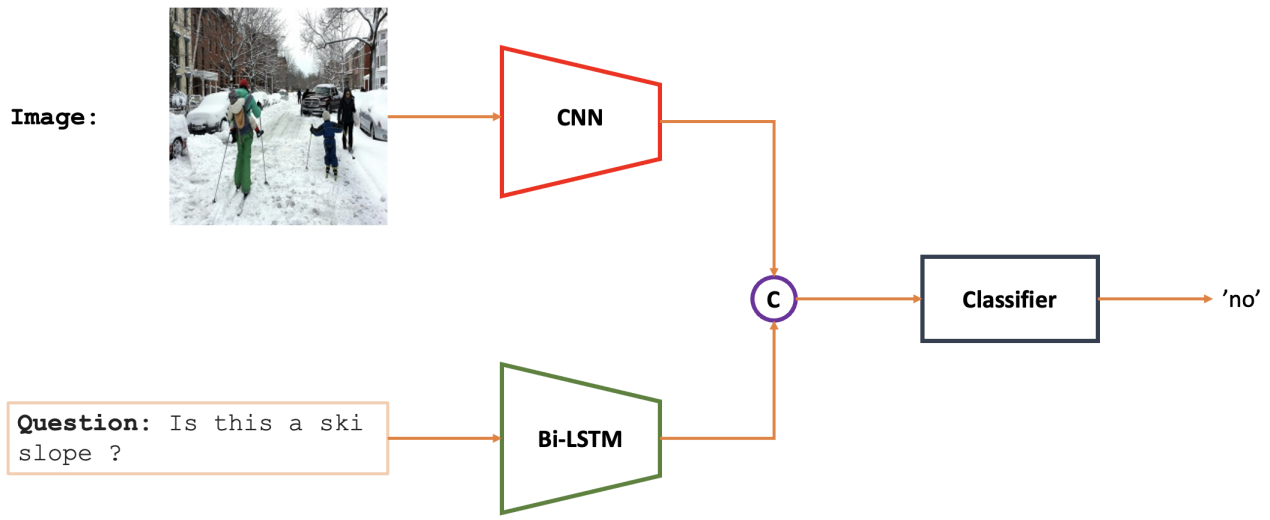
```
train_dataset = VQADataset(
    train_data,
    classes_to_idx=classes_to_idx,
    transform=transform
)
val_dataset = VQADataset(
    val_data,
    classes_to_idx=classes_to_idx,
    transform=transform
)
```

```
test_dataset = VQADataset(
    test_data,
    classes_to_idx=classes_to_idx,
    transform=transform
)
```

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

```
train_batch_size = 128
test_batch_size = 32

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batch_size,
    shuffle=True
)
val_loader = DataLoader(
    val_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
test_loader = DataLoader(
    test_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
```

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




Hình 2: 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 ResNet50. Code của class model có nội dung như sau:

```
class VQAModel(nn.Module):
    def __init__(
        self,
```

```
        n_classes,
        img_model_name='resnet50',
        embeddding_dim=300,
        n_layers=2,
        hidden_size=128,
        dropout_prob=0.2
    ):
        super(VQAModel, self).__init__()
        self.image_encoder = timm.create_model(
            img_model_name,
            pretrained=True,
            num_classes=hidden_size
        )

        self.embedding = nn.Embedding(len(vocab), embeddding_dim)
        self.lstm = nn.LSTM(
            input_size=embeddding_dim,
            hidden_size=hidden_size,
            num_layers=n_layers,
            batch_first=True,
            bidirectional=True
        )
        self.layernorm = nn.LayerNorm(hidden_size * 2)
        self.fc1 = nn.Linear(hidden_size * 3, 256)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout_prob)
        self.fc2 = nn.Linear(256, n_classes)

    def forward(self, img, text):
        img_features = self.image_encoder(img)

        text_emb = self.embedding(text)
        lstm_out, _ = self.lstm(text_emb)

        lstm_out = lstm_out[:, -1, :]
        lstm_out = self.layernorm(lstm_out)

        combined = torch.cat((img_features, lstm_out), dim=1)
        x = self.fc1(combined)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)

        return x

```

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

```
n_classes = len(classes)
img_model_name = 'resnet50'
hidden_size = 128
n_layers = 1
embeddding_dim = 128
dropout_prob = 0.2
device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = VQAModel(
    n_classes=n_classes,
    img_model_name=img_model_name,
    embeddding_dim=embeddding_dim,
    n_layers=n_layers,
```

```
    hidden_size=hidden_size,
    dropout_prob=dropout_prob
).to(device)

```

12. **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:

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for image, question, labels in dataloader:
            image, question, labels = image.to(device), question.to(device), labels.to(device)
            outputs = model(image, question)
            loss = criterion(outputs, labels)
            losses.append(loss.item())
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    loss = sum(losses) / len(losses)
    acc = correct / total

    return loss, acc

def fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    scheduler,
    device,
    epochs
):
    train_losses = []
    val_losses = []

    for epoch in range(epochs):
        batch_train_losses = []

        model.train()
        for idx, (images, questions, labels) in enumerate(train_loader):
            images = images.to(device)
            questions = questions.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
            outputs = model(images, questions)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            batch_train_losses.append(loss.item())

        train_loss = sum(batch_train_losses) / len(batch_train_losses)
        train_losses.append(train_loss)
```

```
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 loss: {
    val_loss:.4f}\tVal Acc: {val_acc}')
61
62          scheduler.step()
63
64      return train_losses, val_losses
65
```

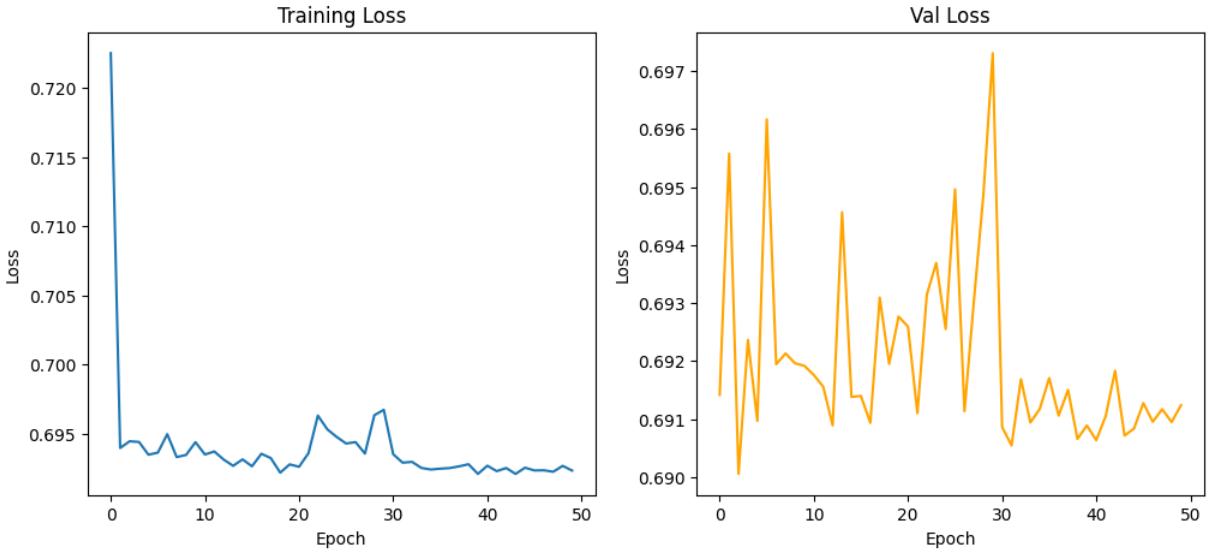
13. **Khai báo hàm loss, optimizer và scheduler:**

```
lr = 1e-2
epochs = 50
weight_decay = 1e-5
scheduler_step_size = epochs * 0.6
criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(
    model.parameters(),
    lr=lr,
    weight_decay=weight_decay
)
scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer,
    step_size=scheduler_step_size,
    gamma=0.1
)

```

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

```
train_losses, val_losses = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    scheduler,
    device,
    epochs
)

```

Khi quá trình huấn luyện kết thúc, ta có thể trực quan kết quả loss của mô hình qua từng epoch như sau:



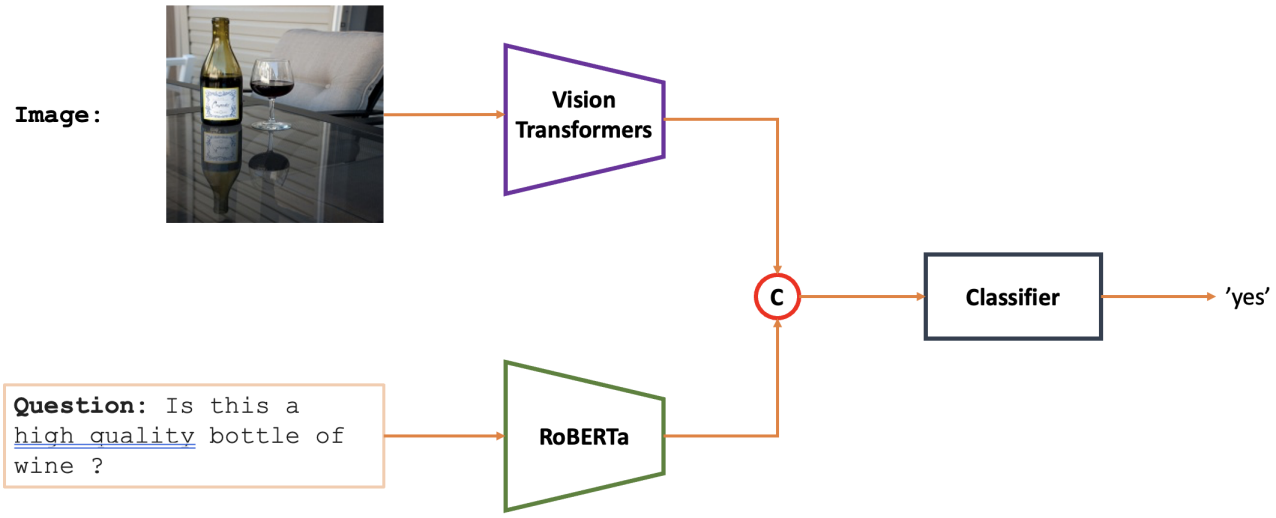

Hình 3: Kết quả huấn luyện của mô hình với ResNet+Bi-LSTM

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

```
val_loss, val_acc = evaluate(
    model,
    val_loader,
    criterion,
    device
)
test_loss, test_acc = evaluate(
    model,
    test_loader,
    criterion,
    device
)

print('Evaluation on val/test dataset')
print('Val accuracy: ', val_acc)
print('Test accuracy: ', test_acc)

```

Có thể thấy, với hướng tiếp cận cơ bản trên, mô hình VQA ta xây dựng không đạt được kết quả khả quan. Vì vậy ở phần sau, ta sẽ 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 transformers, một kiến trúc mạng cực kì mạnh mẽ đang được sử dụng rộng rãi tại thời điểm này. Cụ thể, với ảnh đầu vào, ta dùng VisionTransformers, với câu hỏi, ta dùng RoBERTa. Với 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 phỏng như hình sau:




Hình 4: Ảnh mô tả pipeline cho hướng tiếp cận sử dụng các mô hình thuộc họ transformers

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

1. **Import các thư viện cần thiết:**

```python
import torch
import torch.nn as nn
import os
import numpy as np
import pandas as pd
import timm
import matplotlib.pyplot as plt

from PIL import Image
from torch.utils.data import Dataset, DataLoader
from transformers import ViTModel, ViTImageProcessor
from transformers import AutoTokenizer, RobertaModel

```

2. **Chia bộ dữ liệu train, val, test:** Tương tự phần trên, ta tạo ba bộ dữ liệu train, val, test:

```python
# Load train data
train_data = []
train_set_path = './vaq2.0.TrainImages.txt'

with open(train_set_path, "r") as f:
    lines = f.readlines()
    for line in lines:
        temp = line.split('\t')
        qa = temp[1].split('?')

        if len(qa) == 3:
            answer = qa[2].strip()
        else:
            answer = qa[1].strip()

        data_sample = {
            'image_path': temp[0][:-2],
```

```
            'question': qa[0] + '?',
            'answer': answer
        }
        train_data.append(data_sample)

# Load val data
val_data = []
val_set_path = './vaq2.0.DevImages.txt'

with open(val_set_path, "r") as f:
    lines = f.readlines()
    for line in lines:
        temp = line.split('\t')
        qa = temp[1].split('?')

        if len(qa) == 3:
            answer = qa[2].strip()
        else:
            answer = qa[1].strip()

        data_sample = {
            'image_path': temp[0][:-2],
            'question': qa[0] + '?',
            'answer': answer
        }
        val_data.append(data_sample)

# Load test data
test_data = []
test_set_path = './vaq2.0.TestImages.txt'

with open(test_set_path, "r") as f:
    lines = f.readlines()
    for line in lines:
        temp = line.split('\t')
        qa = temp[1].split('?')

        if len(qa) == 3:
            answer = qa[2].strip()
        else:
            answer = qa[1].strip()

        data_sample = {
            'image_path': temp[0][:-2],
            'question': qa[0] + '?',
            'answer': answer
        }
        test_data.append(data_sample)

```

3. **Xây dựng dictionary mapping classes:** Tương tự như phần trên, ta khởi tạo dictionary dùng để chuyển đổi tên class thành mã số tương ứng và ngược lại:

```
classes = set([sample['answer'] for sample in train_data])
classes_to_idx = {
    cls_name: idx for idx, cls_name in enumerate(classes)
}
idx_to_classes = {
    idx: cls_name for idx, cls_name in enumerate(classes)
}

```

4. **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:

```
class VQADataset(Dataset):
    def __init__(
        self,
        data,
        classes_to_idx,
        img_feature_extractor,
        text_tokenizer,
        device,
        root_dir='/content/val2014-resised/'
    ):
        self.data = data
        self.root_dir = root_dir
        self.classes_to_idx = classes_to_idx
        self.img_feature_extractor = img_feature_extractor
        self.text_tokenizer = text_tokenizer
        self.device = device

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

    def __getitem__(self, index):
        img_path = os.path.join(self.root_dir, self.data[index]['image_path'])
        img = Image.open(img_path).convert('RGB')

        if self.img_feature_extractor:
            img = self.img_feature_extractor(images=img, return_tensors="pt")
            img = {k: v.to(self.device).squeeze(0) for k, v in img.items()}

        question = self.data[index]['question']
        if self.text_tokenizer:
            question = self.text_tokenizer(
                question,
                padding="max_length",
                max_length=20,
                truncation=True,
                return_tensors="pt"
            )
            question = {k: v.to(self.device).squeeze(0) for k, v in question.items()}


        label = self.data[index]['answer']
        label = torch.tensor(
            classes_to_idx[label],
            dtype=torch.long
        ).to(device)

        sample = {
            'image': img,
            'question': question,
            'label': label
        }

        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, vì vậy chúng ta cũng cần truyền vào tham số `device` để chuyển đổi tensor về format tính toán của máy.

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

```
img_feature_extractor = ViTImageProcessor.from_pretrained("google/vit-base-
    patch16-224")
text_tokenizer = AutoTokenizer.from_pretrained("roberta-base")
device = 'cuda' if torch.cuda.is_available() else 'cpu'

train_dataset = VQADataset(
    train_data,
    classes_to_idx=classes_to_idx,
    img_feature_extractor=img_feature_extractor,
    text_tokenizer=text_tokenizer,
    label_encoder=label_encoder,
    device=device
)
val_dataset = VQADataset(
    val_data,
    classes_to_idx=classes_to_idx,
    img_feature_extractor=img_feature_extractor,
    text_tokenizer=text_tokenizer,
    label_encoder=label_encoder,
    device=device
)
test_dataset = VQADataset(
    test_data,
    classes_to_idx=classes_to_idx,
    img_feature_extractor=img_feature_extractor,
    text_tokenizer=text_tokenizer,
    label_encoder=label_encoder,
    device=device
)

```

6. **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:**

```
class TextEncoder(nn.Module):
    def __init__(self):
        super(TextEncoder, self).__init__()
        self.model = RobertaModel.from_pretrained("roberta-base")

    def forward(self, inputs):
        outputs = self.model(**inputs)

        return outputs.pooler_output

```

   – **VisualEncoder:**

```
class VisualEncoder(nn.Module):
    def __init__(self):
        super(VisualEncoder, self).__init__()
        self.model = ViTModel.from_pretrained("google/vit-base-patch16-224
    ")

```

```
    def forward(self, inputs):
        outputs = self.model(**inputs)

        return outputs.pooler_output

```

– **Classifier:**

```
class Classifier(nn.Module):
    def __init__(
        self,
        input_size=768*2,
        hidden_size=512,
        n_layers=1,
        dropout_prob=0.2,
        n_classes=2
    ):
        super(Classifier,self).__init__()
        self.lstm = nn.LSTM(
            input_size,
            hidden_size,
            num_layers=n_layers,
            batch_first=True,
            bidirectional=True
        )
        self.dropout = nn.Dropout(dropout_prob)
        self.fc1 = nn.Linear(hidden_size*2, n_classes)

    def forward(self,x):
        x, _ = self.lstm(x)
        x = self.dropout(x)
        x = self.fc1(x)

        return x

```

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

```
class VQAModel(nn.Module):
    def __init__(
        self,
        visual_encoder,
        text_encoder,
        classifier
    ):
        super(VQAModel, self).__init__()
        self.visual_encoder = visual_encoder
        self.text_encoder = text_encoder
        self.classifier = classifier

    def forward(self, image, answer):
        text_out = self.text_encoder(answer)
        image_out = self.visual_encoder(image)
        x = torch.cat((text_out, image_out), dim=1)
        x = self.classifier(x)

        return x

    def freeze(self, visual=True, textual=True, clas=False):
        if visual:
            for n,p in self.visual_encoder.named_parameters():
```

```
                p.requires_grad = False
        if textual:
            for n,p in self.text_encoder.named_parameters():
                p.requires_grad = False
        if clas:
            for n,p in self.classifier.named_parameters():
                p.requires_grad = False

```

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 pretrained 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:

```
n_classes = len(classes)
hidden_size = 1024
n_layers = 1
dropout_prob = 0.2

text_encoder = TextEncoder().to(device)
visual_encoder = VisualEncoder().to(device)
classifier = Classifier(
    hidden_size=hidden_size,
    n_layers=n_layers,
    dropout_prob=dropout_prob,
    n_classes=n_classes
).to(device)

model = VQAModel(
    visual_encoder=visual_encoder,
    text_encoder=text_encoder,
    classifier=classifier
).to(device)
model.freeze()

```

7. **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:

```
def evaluate(model, dataloader, criterion):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for idx, inputs in enumerate(dataloader):
            images = inputs['image']
            questions = inputs['question']
            labels = inputs['label']
            outputs = model(images, questions)
            loss = criterion(outputs, labels)
            losses.append(loss.item())
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    loss = sum(losses) / len(losses)
    acc = correct / total
```

```
    return loss, acc

def fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    scheduler,
    epochs
):
    train_losses = []
    val_losses = []

    for epoch in range(epochs):
        batch_train_losses = []

        model.train()
        for idx, inputs in enumerate(train_loader):
            images = inputs['image']
            questions = inputs['question']
            labels = inputs['label']

            optimizer.zero_grad()
            outputs = model(images, questions)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            batch_train_losses.append(loss.item())

        train_loss = sum(batch_train_losses) / len(batch_train_losses)
        train_losses.append(train_loss)

        val_loss, val_acc = evaluate(
            model, val_loader,
            criterion
        )
        val_losses.append(val_loss)

        print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}\tVal Acc: {val_acc}')

        scheduler.step()

    return train_losses, val_losses

```

8. **Khai báo hàm loss, optimizer và scheduler:**

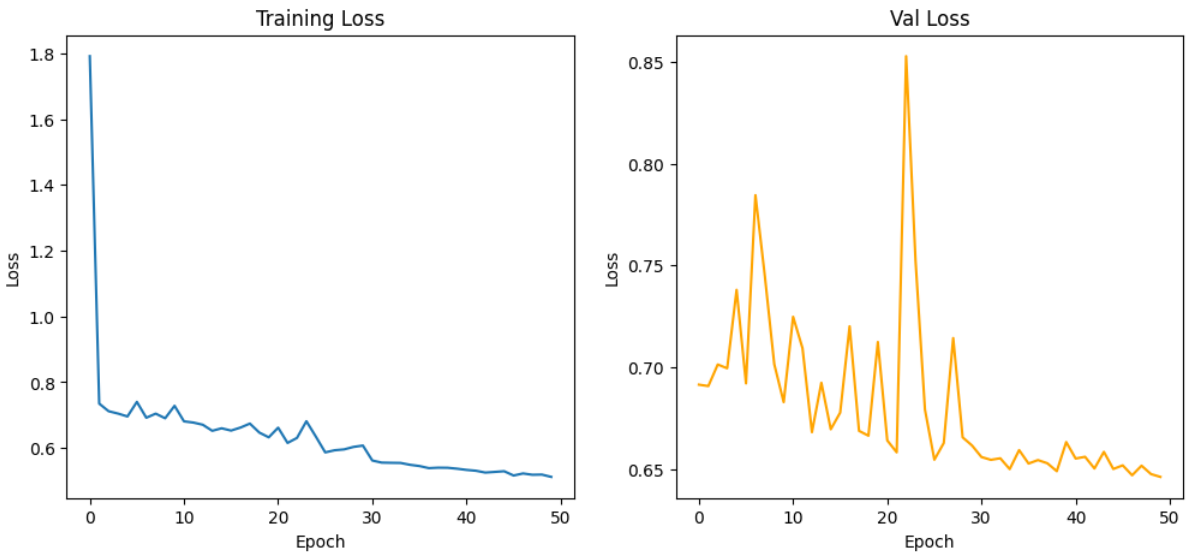
```
lr = 1e-2
epochs = 50
scheduler_step_size = epochs * 0.6
criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(
    model.parameters(),
    lr=lr
)
scheduler = torch.optim.lr_scheduler.StepLR(
```

```
    optimizer,
    step_size=scheduler_step_size,
    gamma=0.1
)

```

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

```
train_losses, val_losses = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    scheduler,
    epochs
)

```

Sau khi quá trình huấn luyện hoàn tất, ta có thể trực quan hóa kết quả loss của mô hình trên hai tập train và val qua từng epoch:




Hình 5: Kết quả huấn luyện của mô hình với ViT+RoBERTa

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

```
val_loss, val_acc = evaluate(
    model,
    val_loader,
    criterion
)
test_loss, test_acc = evaluate(
    model,
    test_loader,
    criterion
)

print('Evaluation on val/test dataset')
print('Val accuracy: ', val_acc)
print('Test accuracy: ', test_acc)

```

# Phần 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 dự trên hình ảnh.
   (*c*) Để tạo sinh ra hình ảnh dựa trên câu 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)

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 kĩ thuật NLP.

8. Đoạn code sau đây dùng để làm gì?

```
eng = spacy.load("en_core_web_sm")

def get_tokens(data_iter):
    for sample in data_iter:
        question = sample['question']
        yield [token.text for token in eng.tokenizer(question)]

```

(*a*) Để load mô hình tiếng anh Spacy.

(*b*) Tạo ra một danh sách những từ từ những 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?

```
vocab = build_vocab_from_iterator(
    get_tokens(train_data),
    min_freq=2,
    specials= ['<pad>', '<sos>', '<eos>', '<unk>'],
    special_first=True
)
vocab.set_default_index(vocab['<unk>'])

```

(*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*) Đào 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?

```
image_encoder = timm.create_model(image_model, pretrained=True, num_classes=
    hidden_dim))

```

    (*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?

```
image_encoder = timm.create_model(image_model, pretrained=True, num_classes=
    hidden_dim))

```

    (*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.

17. Công nghệ nào là cần thiết để giải thích thành phần văn bản trong VQA?
    (*a*) Blockchain
    (*b*) Xử lý Ngôn Ngữ Tự Nhiên (NLP)
    (*c*) Tính Toán Lượng Tử
    (*d*) Mật mã học

***- Hết -***