AI VIETNAM
All-in-One Course
(TA Session)

Visual Question Answering

Project

Dinh-Thang Duong – TA

Year 2023

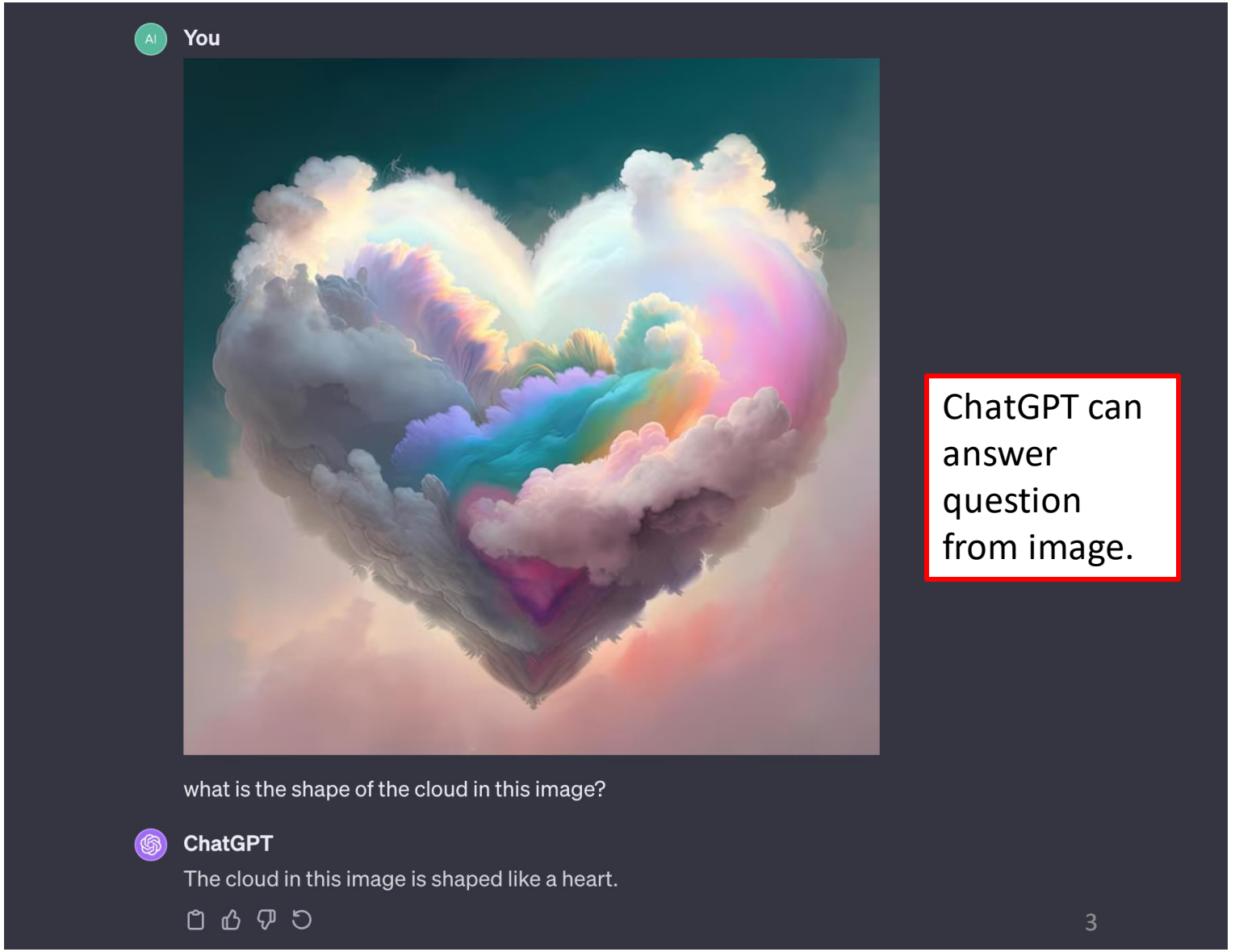
Outline

- Introduction
- CNN+LSTM Approach
- ViT+RoBERTa Approach
- Question

Introduction

❖ Getting Started

Q: What is the shape of the cloud in this image?



ChatGPT can answer question from image.

Introduction

❖ Getting Started

Who is wearing glasses?

man

woman

Where is the child sitting?

fridge

arms

Is the umbrella upside down?

yes

no

How many children are in the bed?

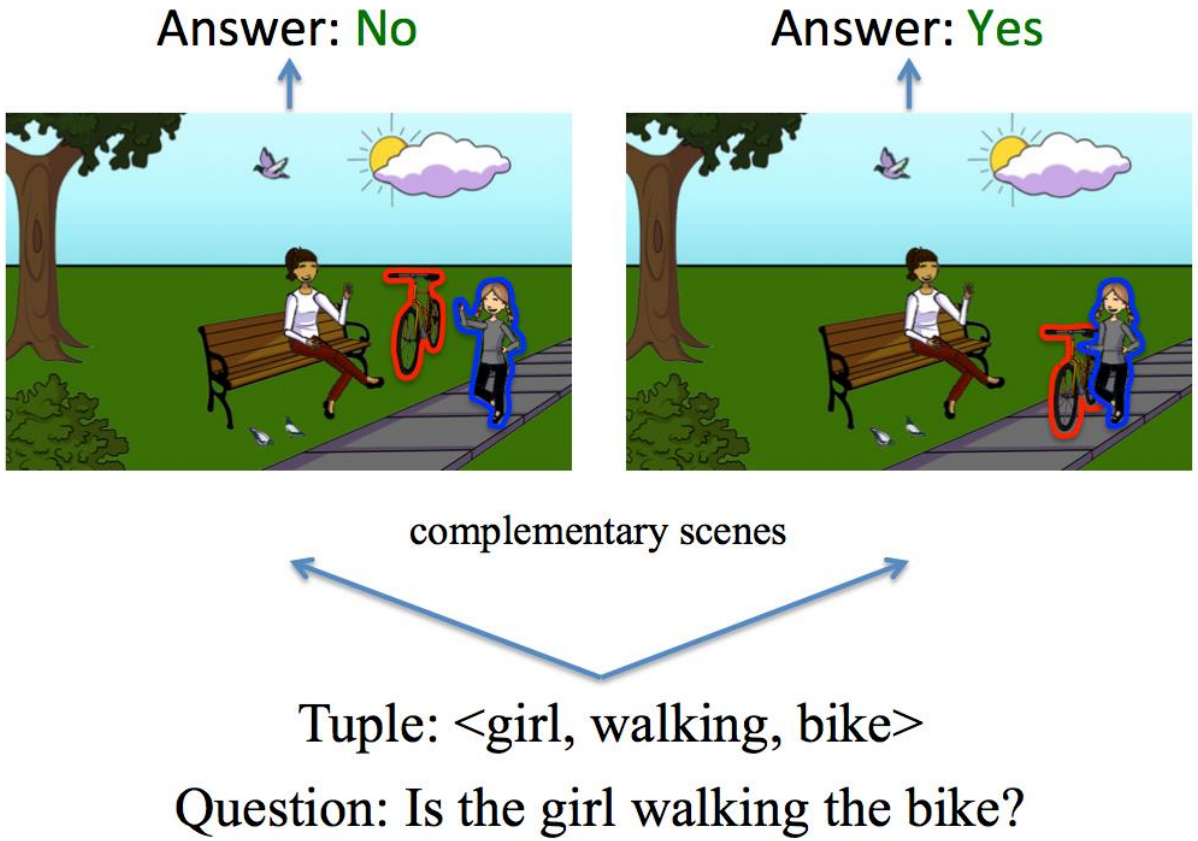
2

1

Many questions can be asked within an image. How can we made a ML model do this task?

Introduction

❖ What is VQA?



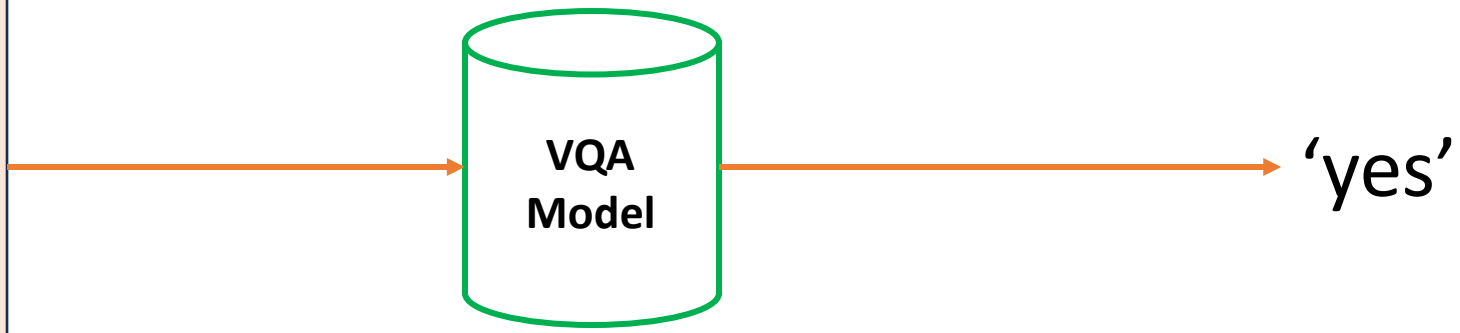
Visual Question Answering (VQA): A task in Machine Learning that aims to answer a question related to a given image. This is a very challenging problem since it requires a model to appropriately extract features from both image and text for prediction.

Introduction

❖ VQA Input/Output

Input

Q: Is this a truck ?



Output

'yes'

Introduction

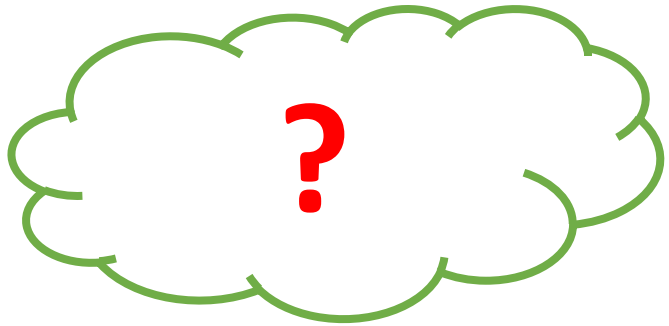
❖ VQA Approach

Image:

Question: Is the umbrella upside down?

Fixed possible answers list:
- 1
- 2
- 3
- yes
- no
- red
- yellow
- green
- nothing
- circle
- round

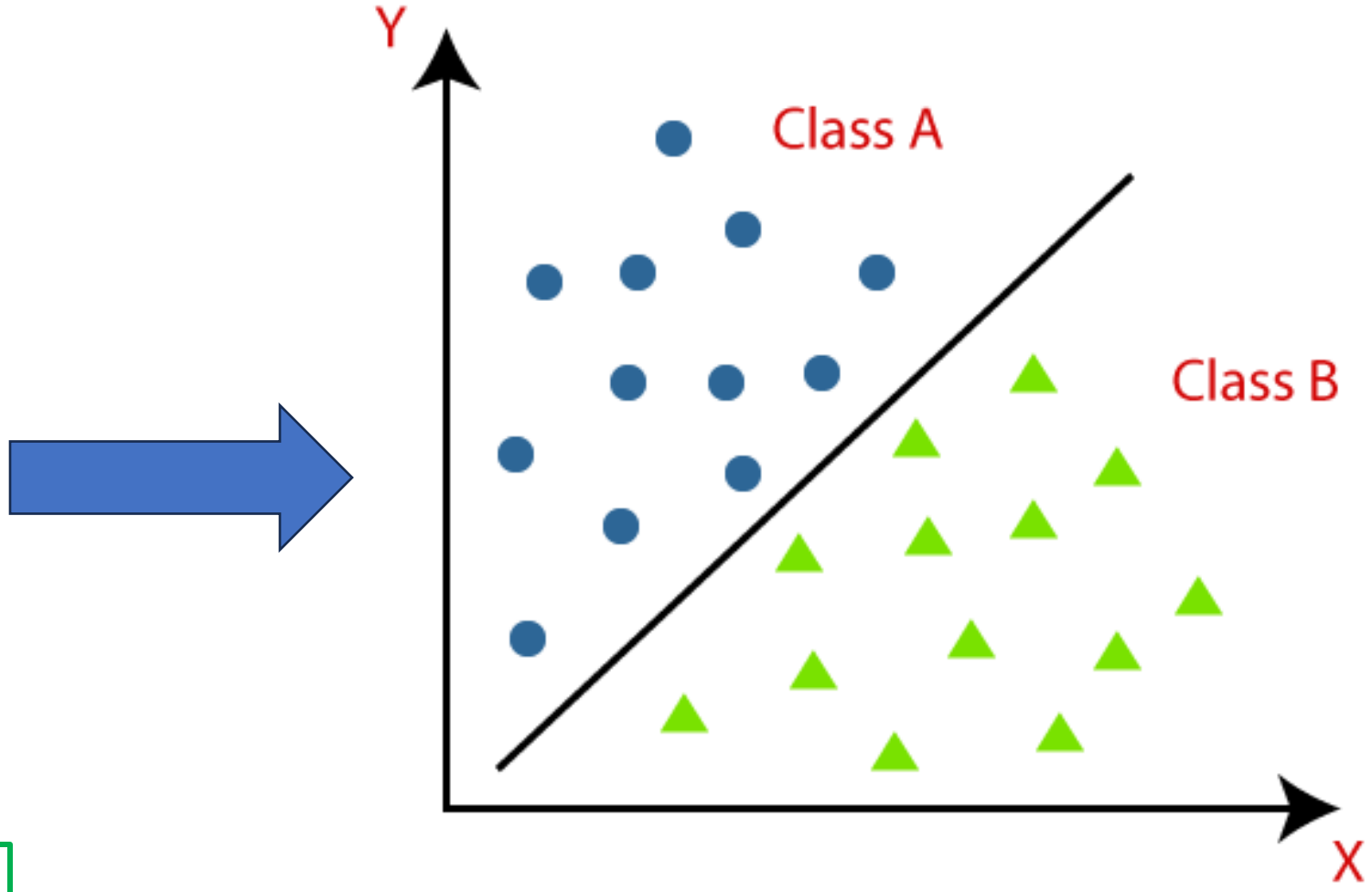
…



Introduction

❖ VQA Approach

Fixed possible answers list:
- 1
- 2
- 3
- yes
- no
- red
- yellow
- green
- nothing
- circle
- round
...



Since we have a list of fixed possible answers...

Introduction

❖ VQA Approach

Image:

Feature Extraction

Fusion

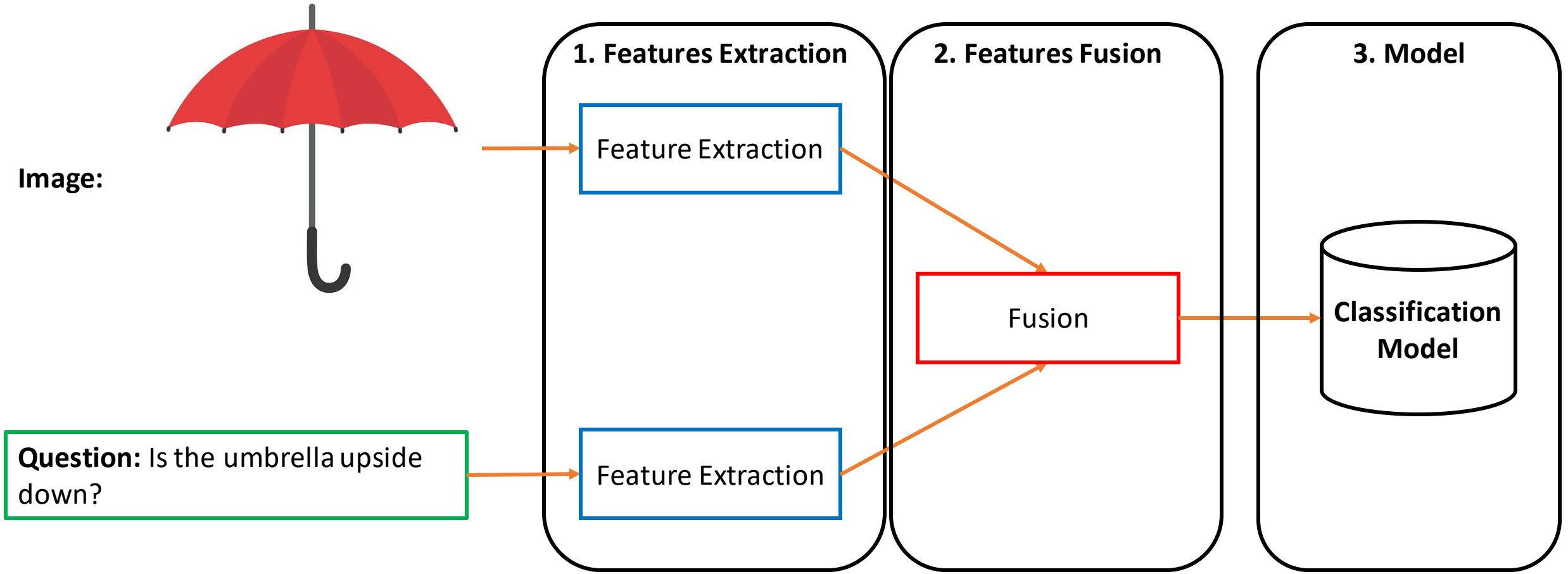
Classification Model

Question: Is the umbrella upside down?

Feature Extraction

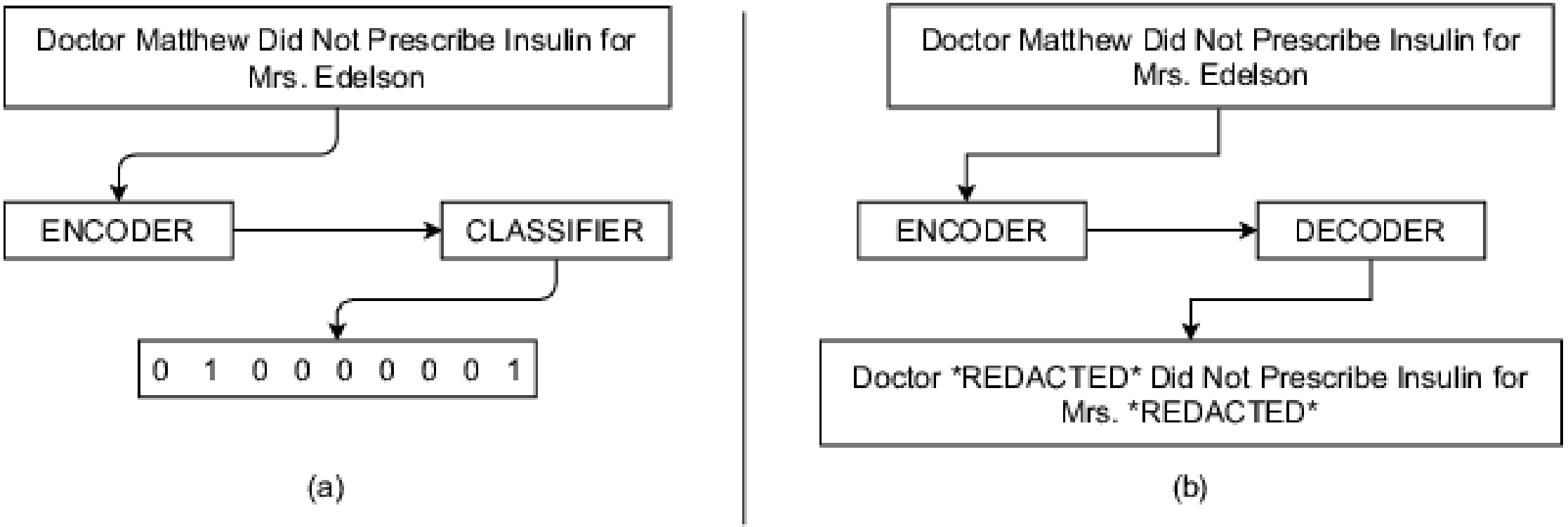
Introduction

❖ VQA Approach



Introduction

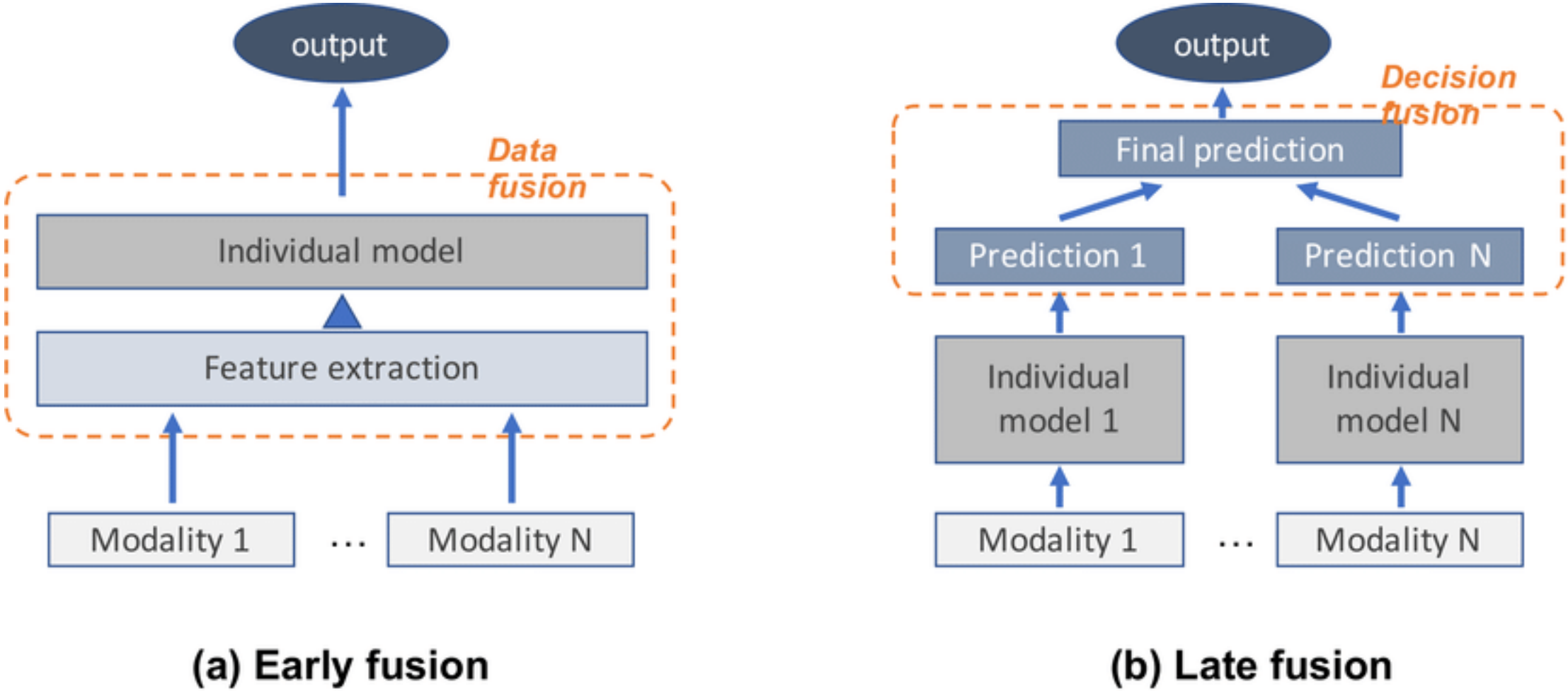
❖ VQA Approach: Type of model



Consider VQA task as Classification or Seq-to-Seq problem?

Introduction

❖ VQA Approach: Features Fusion



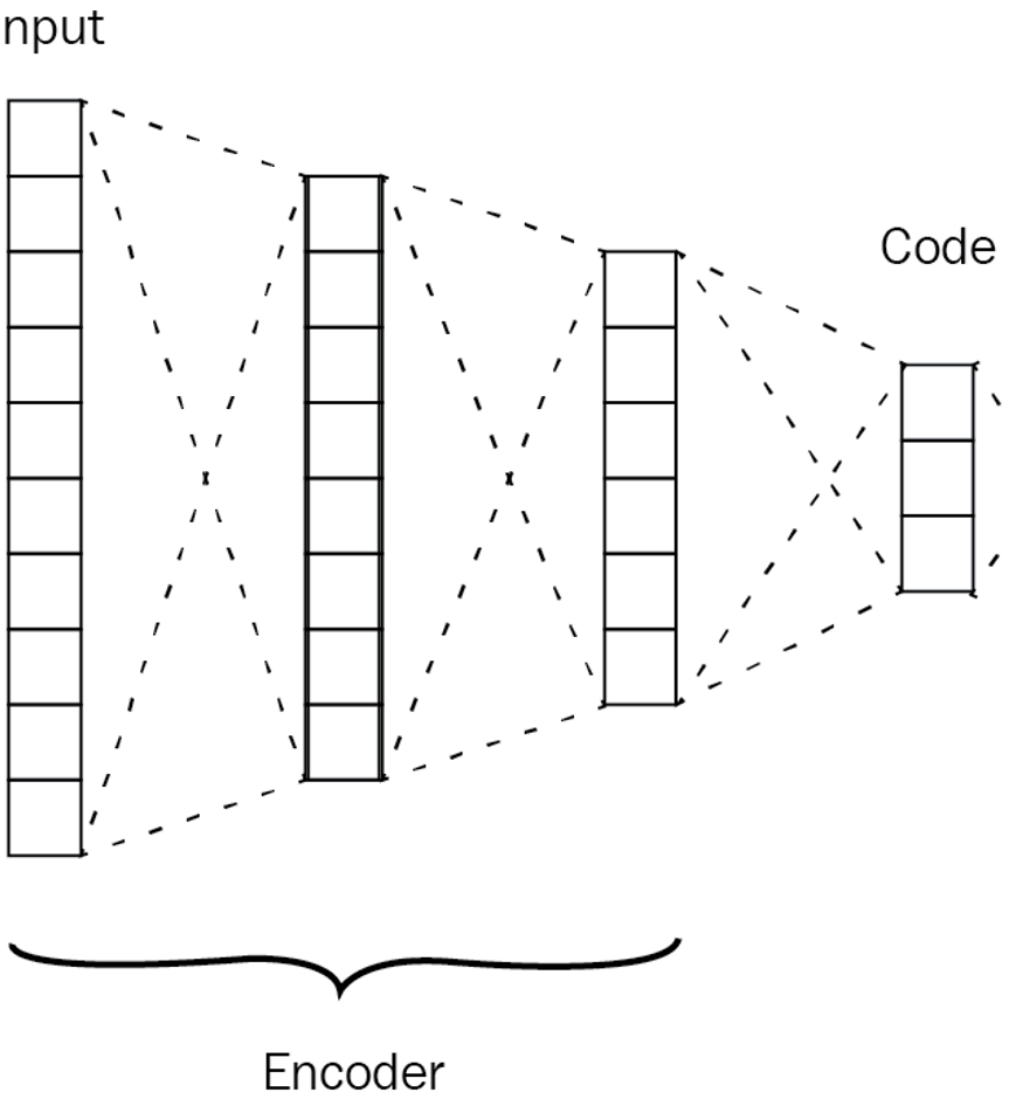
(a) Early fusion

(b) Late fusion

Introduction

❖ VQA Approach: Features Extractions

Image:



Question: Is the umbrella upside down?

How should we encode visual and textual data?

Introduction

❖ Challenges

Image:

Question: Is the umbrella upside down?

How to implement these effectively?

1. Features Extraction

Feature Extraction

Feature Extraction

2. Features Fusion

Fusion

3. Model

Classification Model

Introduction

❖ Challenges

Table 1. Overview of VQA datasets described in this paper.

Dataset	# Images	# Questions	Question Type(s)	Venue	Model(s)	Accuracy
DAQUAR [20]	1449	12468	Object Identitfication	NIPS 2014	AutoSeg [5]	13.75%
VQA [2]	204721	614163	Combining vision, language and common-sense	ICCV 2015	CNN + LSTM	54.06%
Visual Madlibs [36]	10738	360001	Fill in the blanks	ICCV 2015	nCCA (bbox)	47.9%
Visual7W [39]	47300	2201154	7Ws, locating objects	CVPR 2016	LSTM + Attention	55.6%
CLEVR [12]	100000	853554	Synthetic question generation using relations	CVPR 2017	CNN + LSTM + Spatial Relationship	93%
Tally-QA [1]	165000	306907	Counting objects on varying complexities	AAAI 2019	RCN Network	71.8%
KVQA [26]	24602	183007	Questions based on Knowledge Graphs	AAAI 2019	MemNet	59.2%

Different types of question

https://arxiv.org/pdf/1909.01860v2.pdf

Introduction

❖ Challenges

Fig. 2. Samples from VQA dataset [2].

Correct Answer, Plausible Answer, Popular Answers and Random Answers

Introduction

❖ Challenges

This place is a(n) **road**.
When I look at this picture, I feel **free**.
The most interesting aspect of this picture is **the motorcycles**.
One or two seconds before this picture was taken, **they stopped to chat and decided where to go**.
One or two seconds after this picture was taken, **the bikers ride down the road**.

The people are **sitting at** the dining table.

The people are **playing with** the teddy bears.

Person B is **wearing a grey T-shirt**.
Person B is **talking to two other people**.
Person B is **next to an elephant**.
Person C is **a blonde woman**.
Person C is **petting an elephant**.
Person C is **at a zoo**.

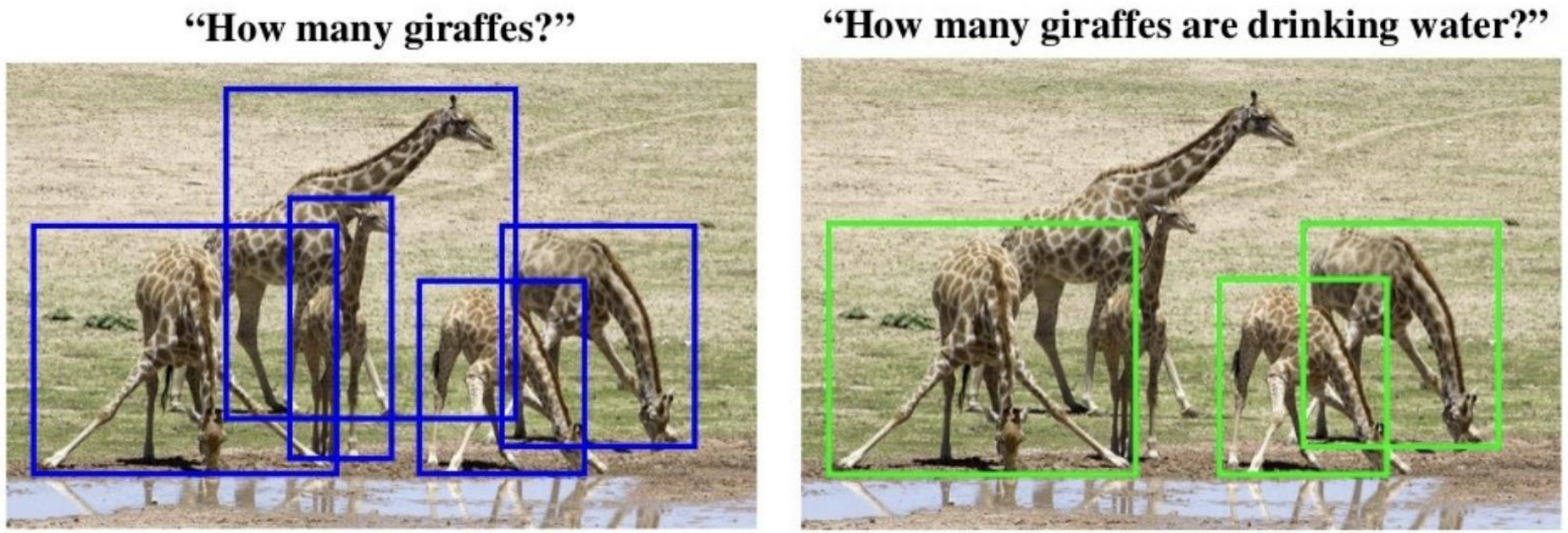
The cake is **iced in chocolate**.
The cake is **on the table**.
People could **eat** the cake.
The table is **covered with a white cloth**.
The table is **in a restaurant**.
People could **have cake at** the table.

Fig. 3. Samples from Madlibs dataset [36].

Fill in the blank questions

Introduction

❖ Challenges



(a) Simple counting question

(b) Complex counting question

Fig. 4. Samples from Tally-QA dataset [1].

Object Counting questions

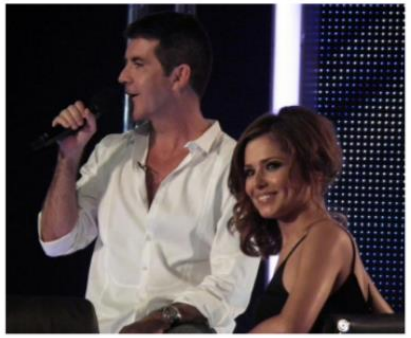
Introduction

❖ Challenges

(a) ***Wikipedia caption*:** Khan with United States Secretary of State Hillary Clinton in 2009.

Q: Who is to the left of Hillary Clinton? (*spatial*)
A: **Aamir Khan**

Q: Do all the people in the image have a common occupation? (*multi-entity, intersection, 1-hop, Boolean*)
A: **No**



(b) ***Wikipedia caption*:** Cheryl alongside Simon Cowell on The X Factor, London, June 2010.

Q: What is the age gap between the two people in the image? (*multi-entity, subtraction, 1-hop*)
A: **24 years**

Q: How many people in this image were born in United Kingdom? (*1-hop, multi-entity, counting*)
A: **2**

(c) ***Wikipedia caption*:** BRICS leaders at the G-20 summit in Brisbane, Australia, 15 November 2014

Q: Were all the people in the image born in the same country? (*Boolean, multi-entity, intersection*)
A: **No**

Q: Who is the founder of the political party to which person second from left belongs to? (*spatial, multi-hop*)
A: **Syama Prasad Mookerjee**

(d) ***Wikipedia caption*:** Serena Williams and Venus Williams, Australian Open 2009.

Q: Who among the people in the image is the eldest? (multi-entity, comparison)
A: **Person in the left**

Q: Who among the people in the image were born after the end of World War II? (multi-entity, multi-relation, comparison)
A: **Both**

Fig. 5. Samples from KVQA dataset [26].

Knowledge-based questions

Introduction

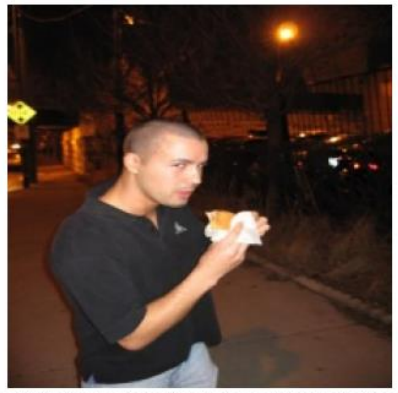
❖ Project Description

Description: Given a dataset about Yes/No questions with images (download here), build a model to answer a given question related to the image.

Q: Is this a healthy dish ?
A: yes

Q: Are there any clouds in the sky ?
A: no

Q: Is this a city ?
A: yes



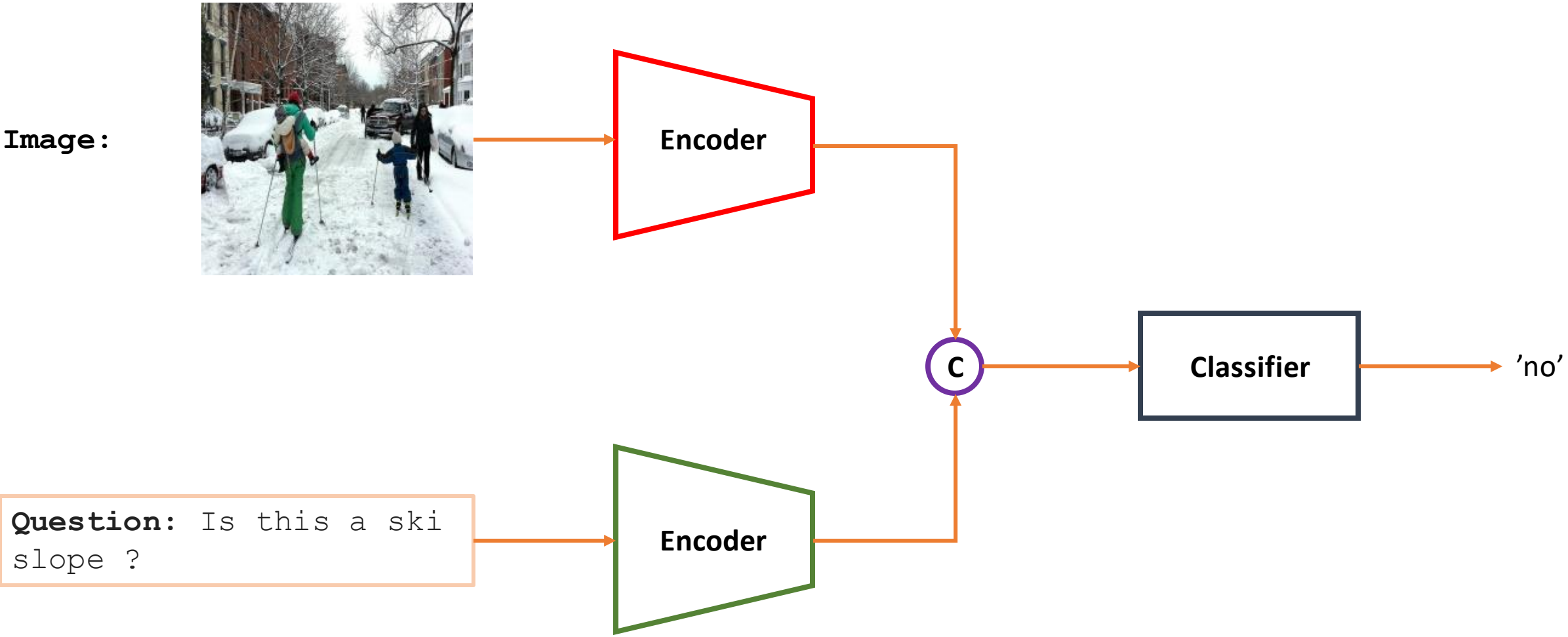
Q: Is this man standing on a crowded street ?
A: no

Q: Is this a swimming pool ?
A: no

Q: Is this a high bridge ?
A: no

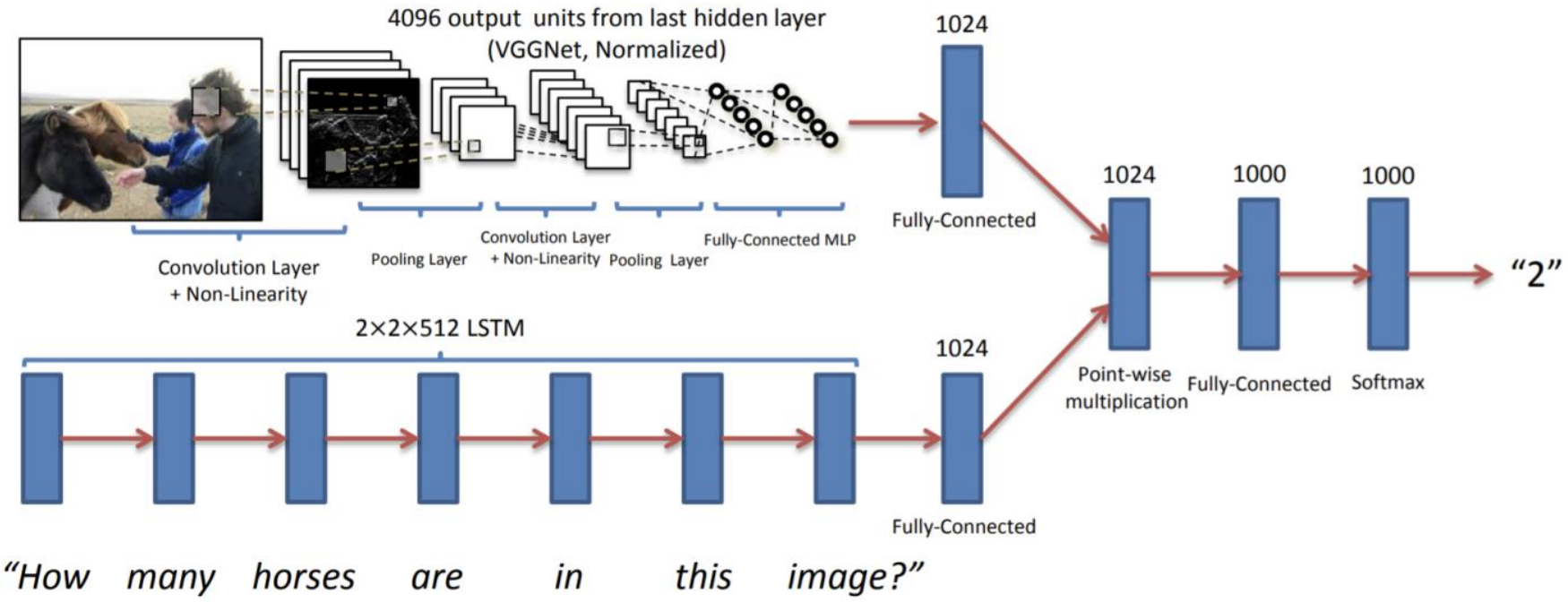
Introduction

❖ Pipeline



CNN+LSTM Approach

❖ Introduction

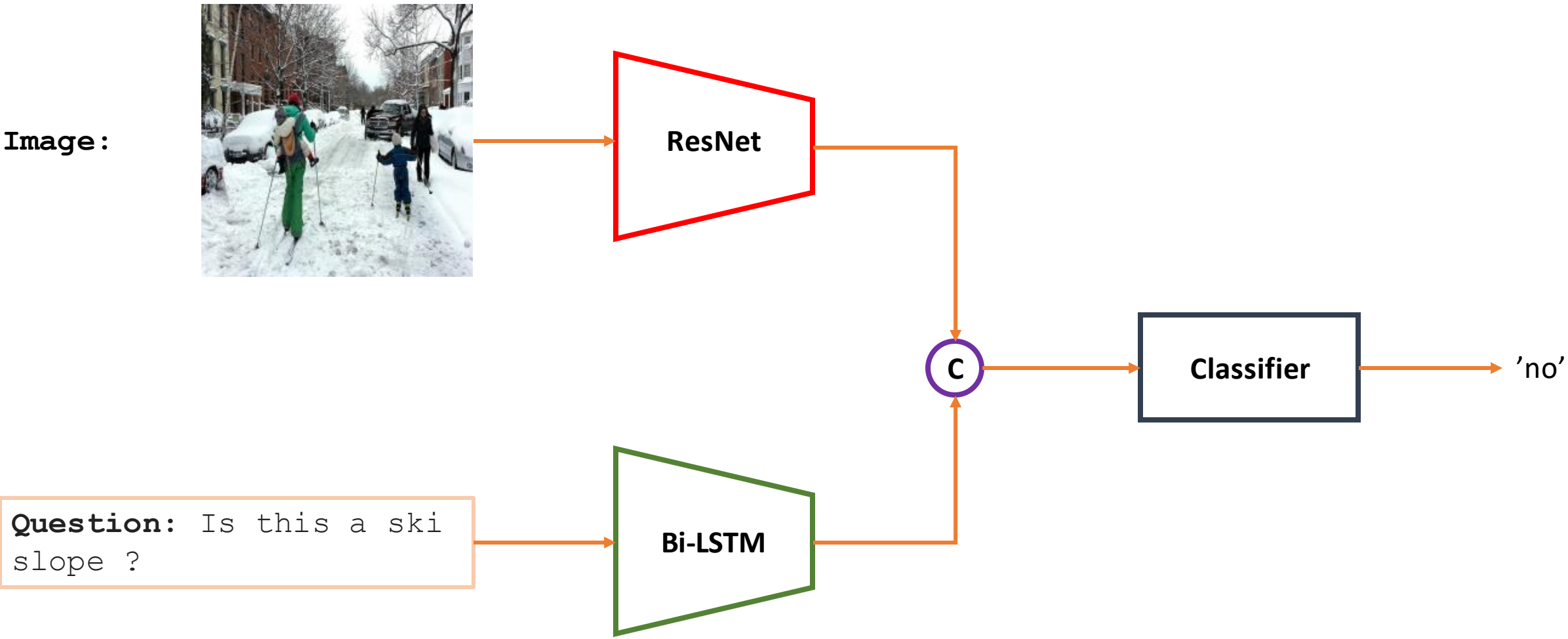


Vanilla VQA: Considered as a benchmark for deep learning methods, the vanilla VQA model uses CNN for feature extraction and LSTM or Recurrent networks for language processing.

Vanilla VQA: https://openaccess.thecvf.com/content_iccv_2015/papers/Antol_VQA_Visual_Question_ICCV_2015_paper.pdf

CNN+LSTM Approach

❖ Pipeline



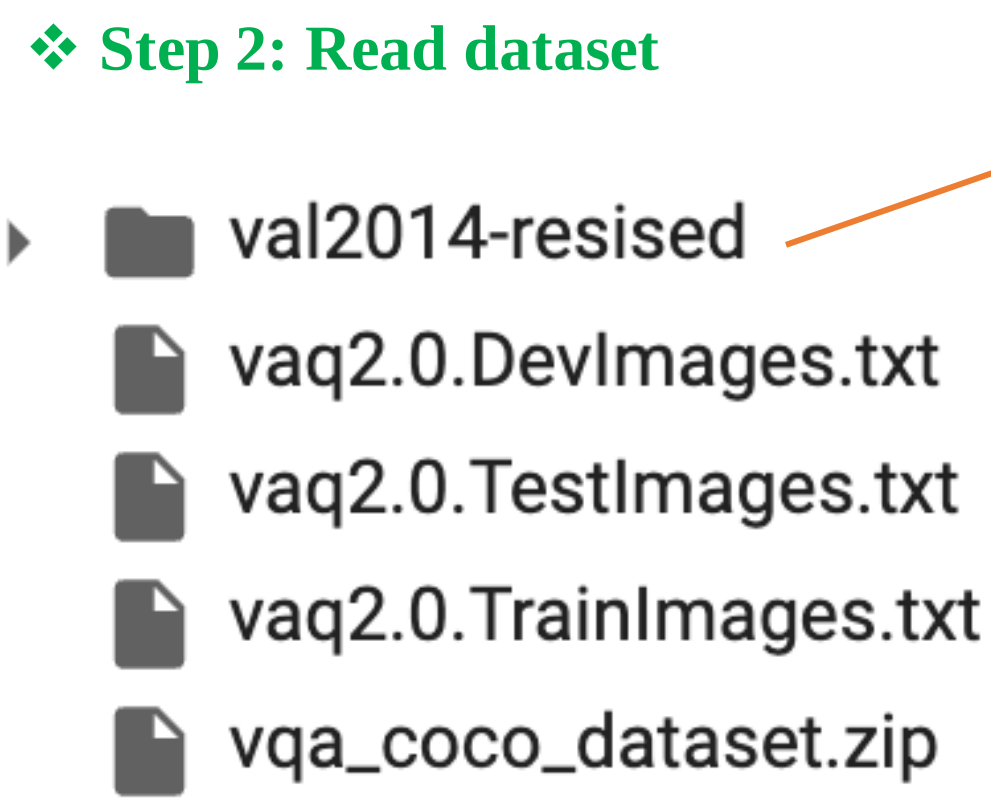
CNN+LSTM Approach

❖ Step 1: Import libraries

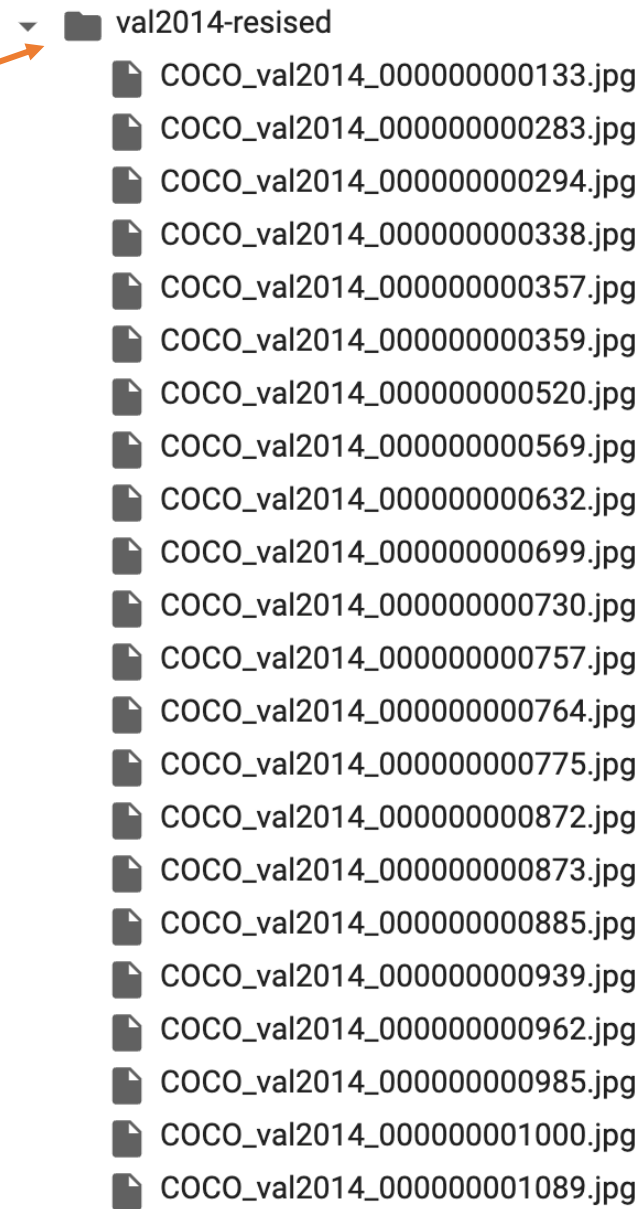
```
import torch
import torch.nn as nn
import torchtext
import os
import numpy as np
import pandas as pd
import spacy
import timm
import matplotlib.pyplot as plt

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

CNN+LSTM Approach

❖ Step 2: Read dataset

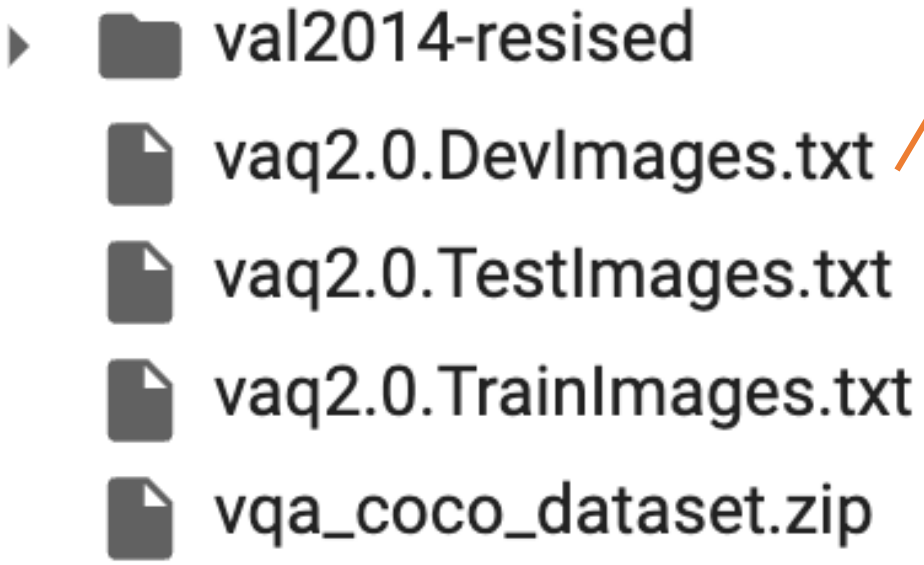


Dataset: images folder



CNN+LSTM Approach

❖ Step 2: Read dataset



Dataset: File label structure

vaq2.0.DevImages.txt

```
1 COCO_val2014_000000262175.jpg#0 Is this a designer tie ? no
2 COCO_val2014_000000393284.jpg#1 Is this man snowboarding ? yes
3 COCO_val2014_000000000133.jpg#0 Is this a child room ? yes
4 COCO_val2014_000000000133.jpg#0 Could this be child room ? yes
5 COCO_val2014_000000240323.jpg#0 Is this a vegetarian meal ? no
6 COCO_val2014_000000524450.jpg#1 Could this be tour bus ? yes
7 COCO_val2014_000000262386.jpg#1 Is this a metal toilet ? no
8 COCO_val2014_000000262386.jpg#2 Is this a private bathroom ? yes
9 COCO_val2014_000000000294.jpg#0 Is this a home kitchen ? yes
10 COCO_val2014_000000000294.jpg#1 Is this man cooking at home ? yes
11 COCO_val2014_000000393578.jpg#0 Is this a train ? no
12 COCO_val2014_000000262608.jpg#1 Could this be sandwich with a bun ? no
13 COCO_val2014_000000262608.jpg#2 Could this be healthy dish ? yes
14 COCO_val2014_000000262651.jpg#1 Is this a leisure boat ? no
15 COCO_val2014_000000262682.jpg#0 Is this a color-coordinated room ? yes
16 COCO_val2014_000000262682.jpg#0 Could this be color-coordinated room ? yes
17 COCO_val2014_000000000569.jpg#0 Is this a color photograph ? yes
18 COCO_val2014_000000087489.jpg#1 Could this be bus seen in America ? yes
19 COCO_val2014_000000524992.jpg#0 Is this a bus depot ? yes
20 COCO_val2014_000000262895.jpg#1 Is this man happy ? no
```

CNN+LSTM Approach

❖ Step 2: Read dataset

vaq2.0.DevImages.txt

```
COCO_val2014_000000262175.jpg#0 Is this a designer tie ? no
COCO_val2014_000000393284.jpg#1 Is this man snowboarding ? yes
COCO_val2014_000000000133.jpg#0 Is this a child room ? yes
COCO_val2014_000000000133.jpg#0 Could this be child room ? yes
COCO_val2014_000000240323.jpg#0 Is this a vegetarian meal ? no
COCO_val2014_000000524450.jpg#1 Could this be tour bus ? yes
COCO_val2014_000000262386.jpg#1 Is this a metal toilet ? no
COCO_val2014_000000262386.jpg#2 Is this a private bathroom ? yes
COCO_val2014_000000000294.jpg#0 Is this a home kitchen ? yes
COCO_val2014_000000000294.jpg#1 Is this man cooking at home ? yes
COCO_val2014_000000393578.jpg#0 Is this a train ? no
COCO_val2014_000000262608.jpg#1 Could this be sandwich with a bun ? no
COCO_val2014_000000262608.jpg#2 Could this be healthy dish ? yes
COCO_val2014_000000262651.jpg#1 Is this a leisure boat ? no
COCO_val2014_000000262682.jpg#0 Is this a color-coordinated room ? yes
COCO_val2014_000000262682.jpg#0 Could this be color-coordinated room ? yes
COCO_val2014_000000000569.jpg#0 Is this a color photograph ? yes
COCO_val2014_000000087489.jpg#1 Could this be bus seen in America ? yes
COCO_val2014_000000524992.jpg#0 Is this a bus depot ? yes
COCO_val2014_000000262895.jpg#1 Is this man happy ? no
```

In order to create data sample, we need to extract:

1. Image Path
2. Question
3. Answer

CNN+LSTM Approach

❖ Step 2: Read dataset

vaq2.0.DevImages.txt

```
COCO_val2014_000000262175.jpg#0 Is this a designer tie ? no
COCO_val2014_000000393284.jpg#1 Is this man snowboarding ? yes
COCO_val2014_000000000133.jpg#0 Is this a child room ? yes
COCO_val2014_000000000133.jpg#0 Could this be child room ? yes
COCO_val2014_000000240323.jpg#0 Is this a vegetarian meal ? no
COCO_val2014_000000524450.jpg#1 Could this be tour bus ? yes
COCO_val2014_000000262386.jpg#1 Is this a metal toilet ? no
COCO_val2014_000000262386.jpg#2 Is this a private bathroom ? yes
COCO_val2014_000000000294.jpg#0 Is this a home kitchen ? yes
COCO_val2014_000000000294.jpg#1 Is this man cooking at home ? yes
COCO_val2014_000000393578.jpg#0 Is this a train ? no
COCO_val2014_000000262608.jpg#1 Could this be sandwich with a bun ? no
COCO_val2014_000000262608.jpg#2 Could this be healthy dish ? yes
COCO_val2014_000000262651.jpg#1 Is this a leisure boat ? no
COCO_val2014_000000262682.jpg#0 Is this a color-coordinated room ? yes
COCO_val2014_000000262682.jpg#0 Could this be color-coordinated room ? yes
COCO_val2014_000000000569.jpg#0 Is this a color photograph ? yes
COCO_val2014_000000087489.jpg#1 Could this be bus seen in America ? yes
COCO_val2014_000000524992.jpg#0 Is this a bus depot ? yes
COCO_val2014_000000262895.jpg#1 Is this man happy ? no
```

Line format: IMG_PATH\tQUESTION ? ANSWER

```python
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)
```

CNN+LSTM Approach

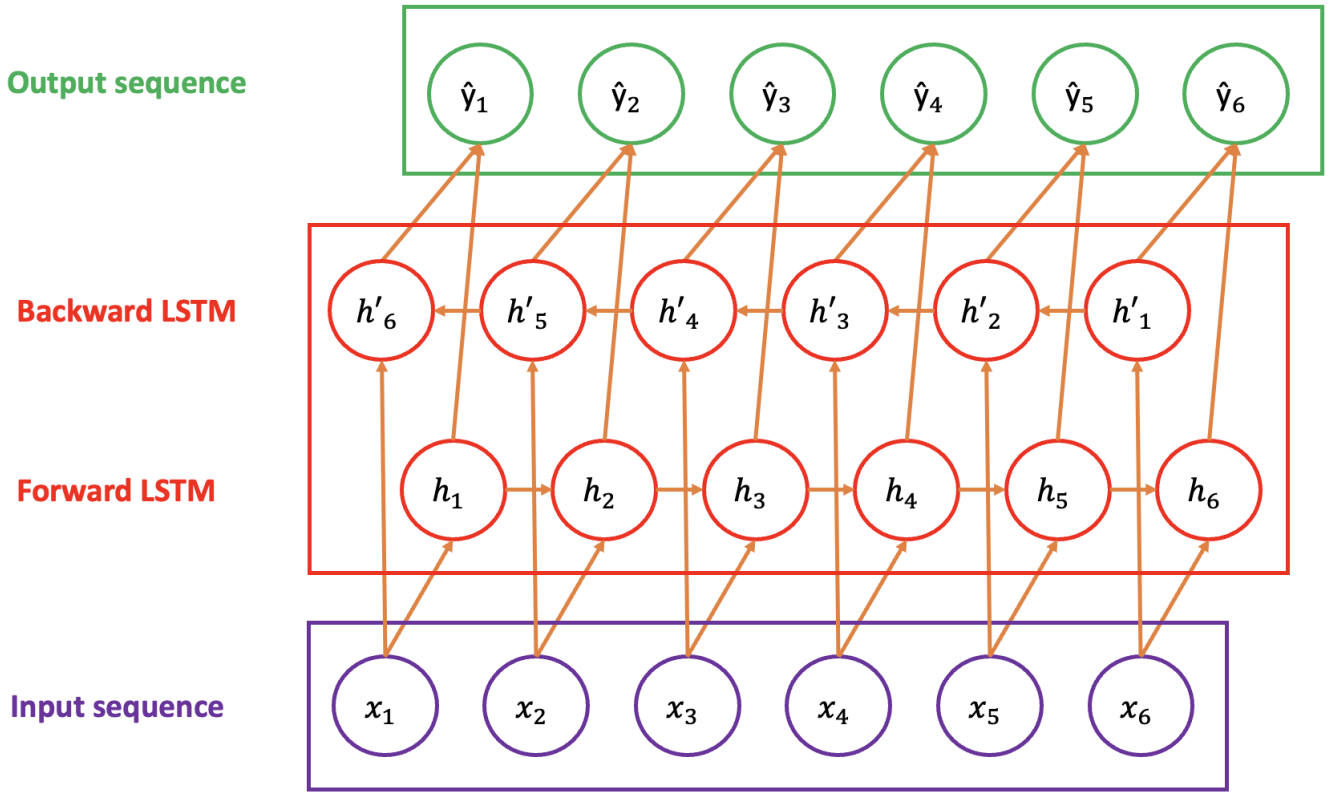
❖ Step 2: Read dataset

```
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)
```

```
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)
```

CNN+LSTM Approach

❖ Step 3: Data Preprocessing



Q: Is this a truck ?

↓

Text Tokenizer

↓

[6, 5, 7, 16, 21, 75, 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

We need to encode question into tokens

CNN+LSTM Approach

❖ Step 3: Data Preprocessing

```
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)]


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>'])
```

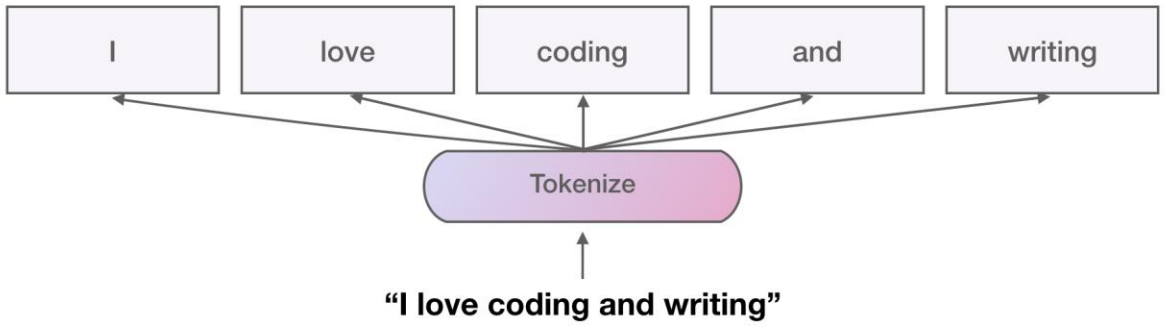
```
len(vocab)
```

```
1678
```

Using build_vocab_from_iterator() function to build a vocabulary

CNN+LSTM Approach

❖ Step 3: Data Preprocessing




```
example_question = "Is this a picture of an apple?"
max_seq_len = 20

tokenize(example_question, max_seq_len)
```

```
[6, 5, 7, 16, 21, 75, 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

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

    return sequence
```

Examination of tokenize() function

CNN+LSTM Approach

❖ Step 3: Data Preprocessing

Classname	ID
no	0
yes	1

```
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)
}
print(idx_to_classes)
```

```
{0: 'yes', 1: 'no'}
```

Create a classname, id mapping dictionary

CNN+LSTM Approach

❖ Step 4: Create pytorch datasets

```
class VQADataset(Dataset):
    def __init__(
        self,
        data,
        classes_to_idx,
        max_seq_len=20,
        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
```

CNN+LSTM Approach

❖ Step 5: Create dataloader

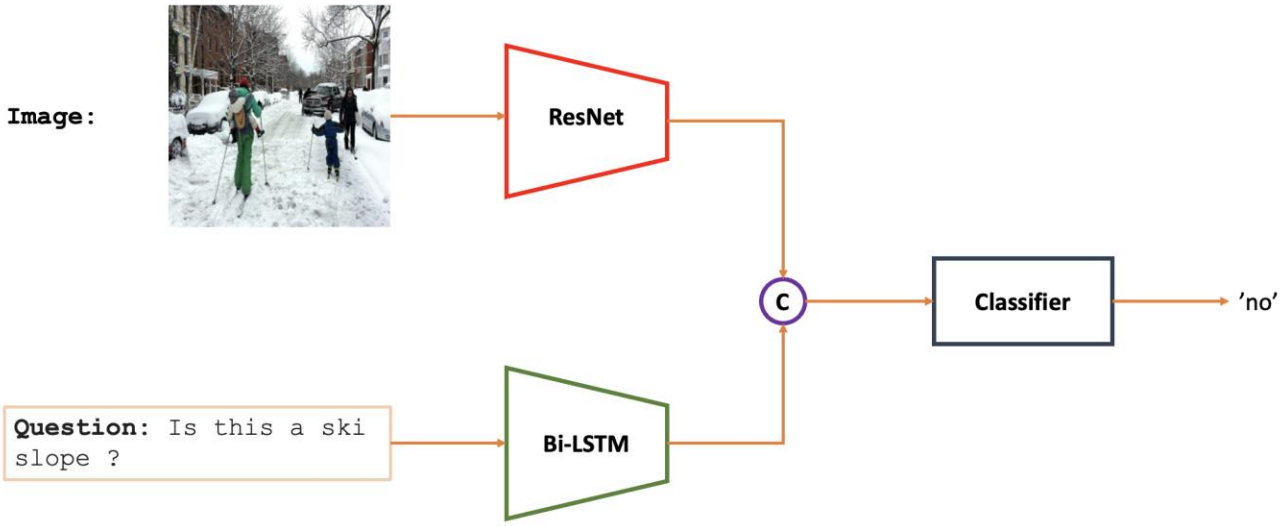
```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(
        (0.485, 0.456, 0.406),
        (0.229, 0.224, 0.225)
    ),
])
```

```
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
)
```

```
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
)
```

CNN+LSTM Approach

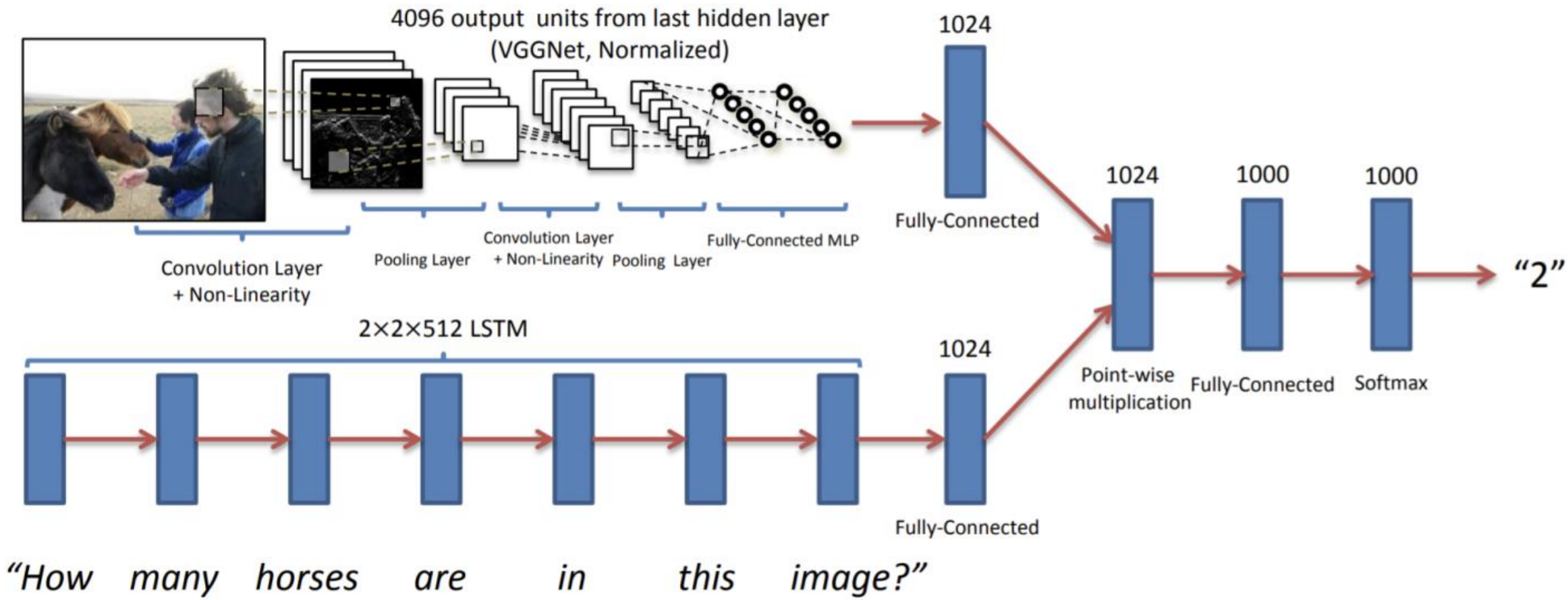
❖ Step 6: Create model




```python
class VQAModel(nn.Module):
    def __init__(
        self,
        n_classes,
        img_model_name='resnet50',
        embeddding_dim=300,
        n_layers=2,
        hidden_size=512,
        dropout_prob=0.2
    ):
        super(VQAModel, self).__init__()
        self.image_encoder = timm.create_model(
            img_model_name,
            pretrained=True,
            num_classes=hidden_size
        )
        self.img_fc = nn.Linear(hidden_size, 1024)
        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.text_fc = nn.Linear(hidden_size * 2, 1024)
        self.fc1 = nn.Linear(1024, 1000)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout_prob)
        self.fc2 = nn.Linear(1000, n_classes)
```

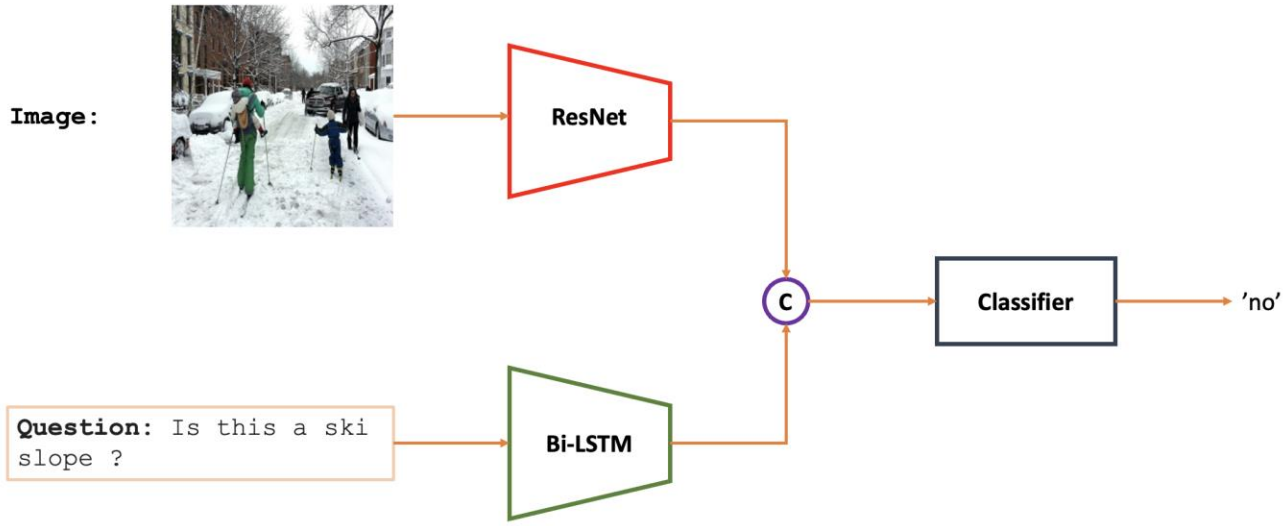
CNN+LSTM Approach

❖ Step 6: Create model



CNN+LSTM Approach

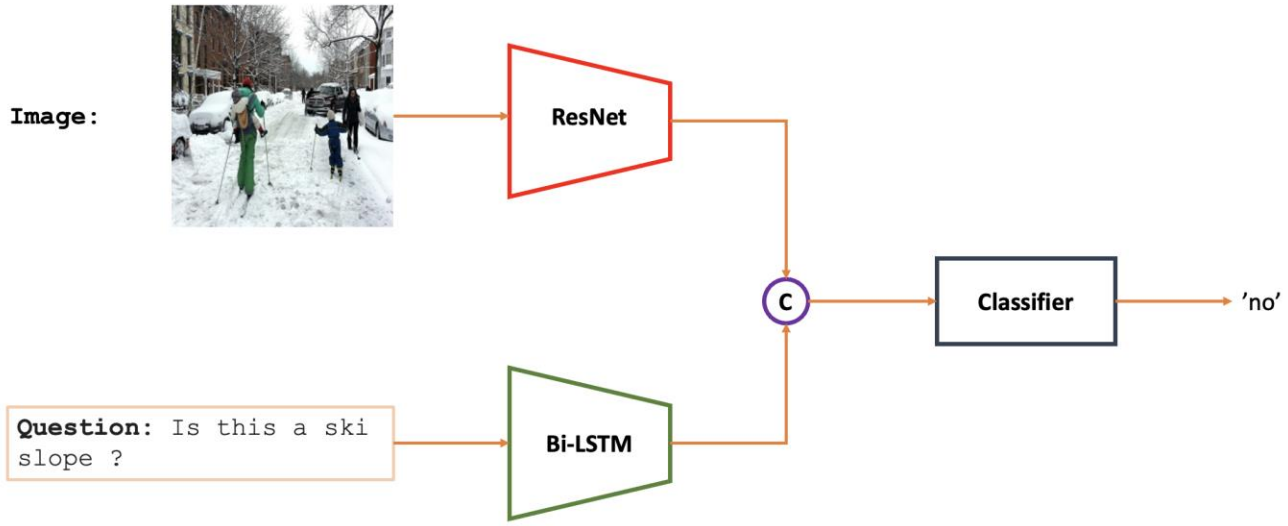
❖ Step 6: Create model




```python
def forward(self, img, text):
    img_features = self.image_encoder(img)
    img_features = self.img_fc(img_features)
    text_emb = self.embedding(text)
    lstm_out, _ = self.lstm(text_emb)
    lstm_out = lstm_out[:, -1, :]
    lstm_out = self.text_fc(lstm_out)

    x = img_features * lstm_out
    x = self.fc1(x)
    x = self.relu(x)
    x = self.dropout(x)
    x = self.fc2(x)

    return x
```

CNN+LSTM Approach

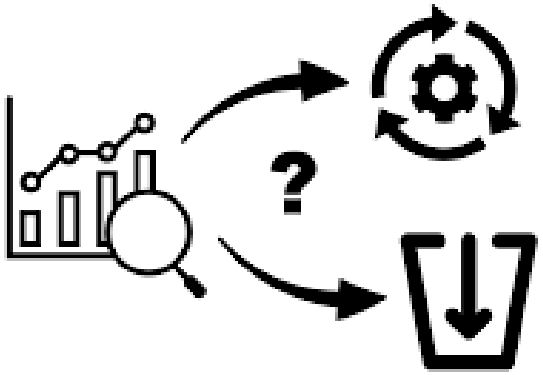
❖ Step 6: Create model

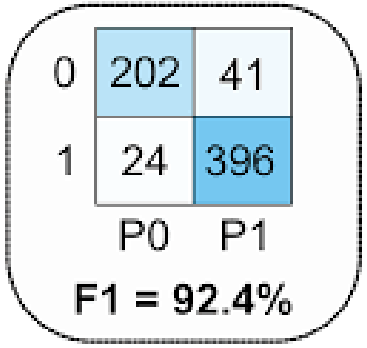



```
n_classes = len(classes)
img_model_name = 'resnet50'
hidden_size = 512
n_layers = 2
embeddding_dim = 64
dropout_prob = 0.2
device = 'cuda' if torch.cuda.is_available() else 'cpu'

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)
```

CNN+LSTM Approach

❖ Step 7: Create evaluate model






```python
def evaluate(model, dataloader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for image, question, labels in dataloader:
            image = image.to(device)
            question = question.to(device)
            labels = 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
```

CNN+LSTM Approach

❖ Step 8: Training

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

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

        train_correct = 0
        train_total = 0

        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)
            _, predicted = torch.max(outputs.data, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()
            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)

        train_acc = train_correct / train_total
        train_accs.append(train_acc)

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

        print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tTrain acc:

        scheduler.step()

    return train_losses, val_losses, train_accs, val_accs
```

CNN+LSTM Approach

❖ Step 8: Training

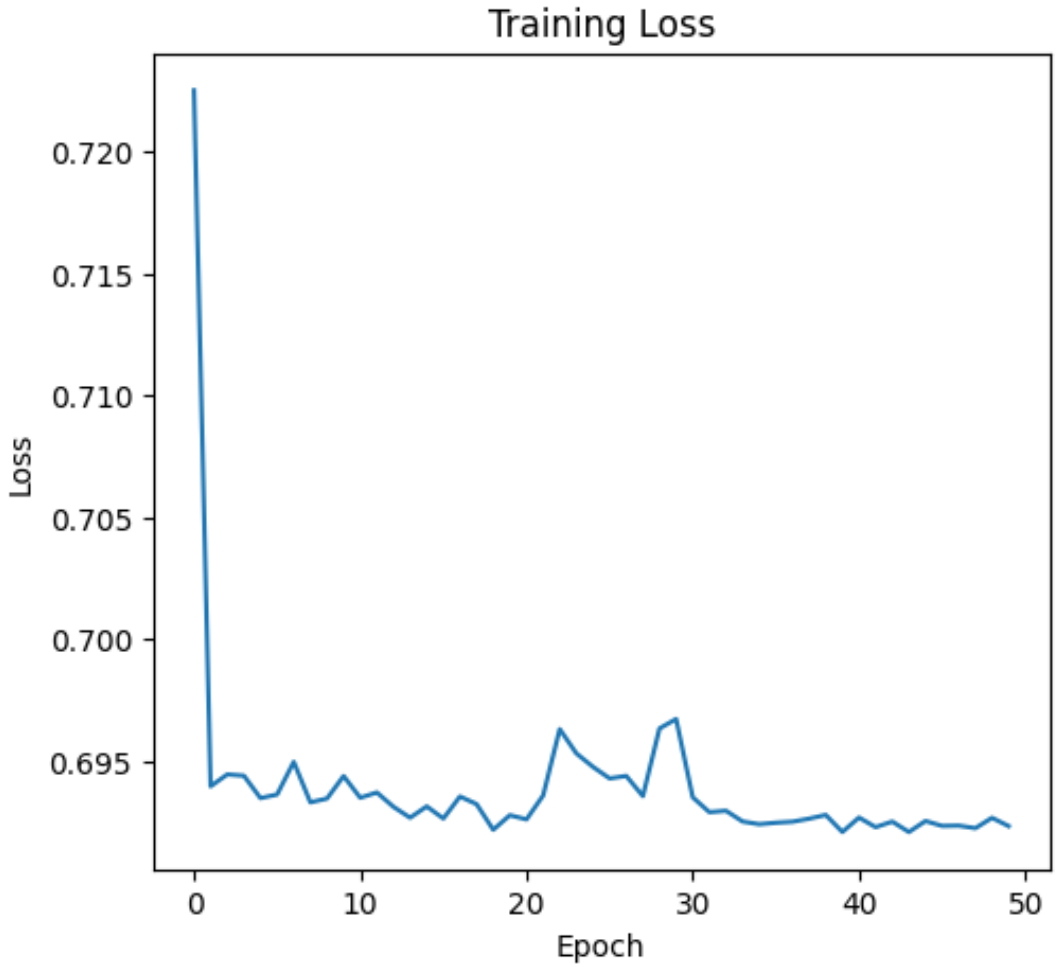
```
lr = 1e-2
epochs = 50

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
)
```

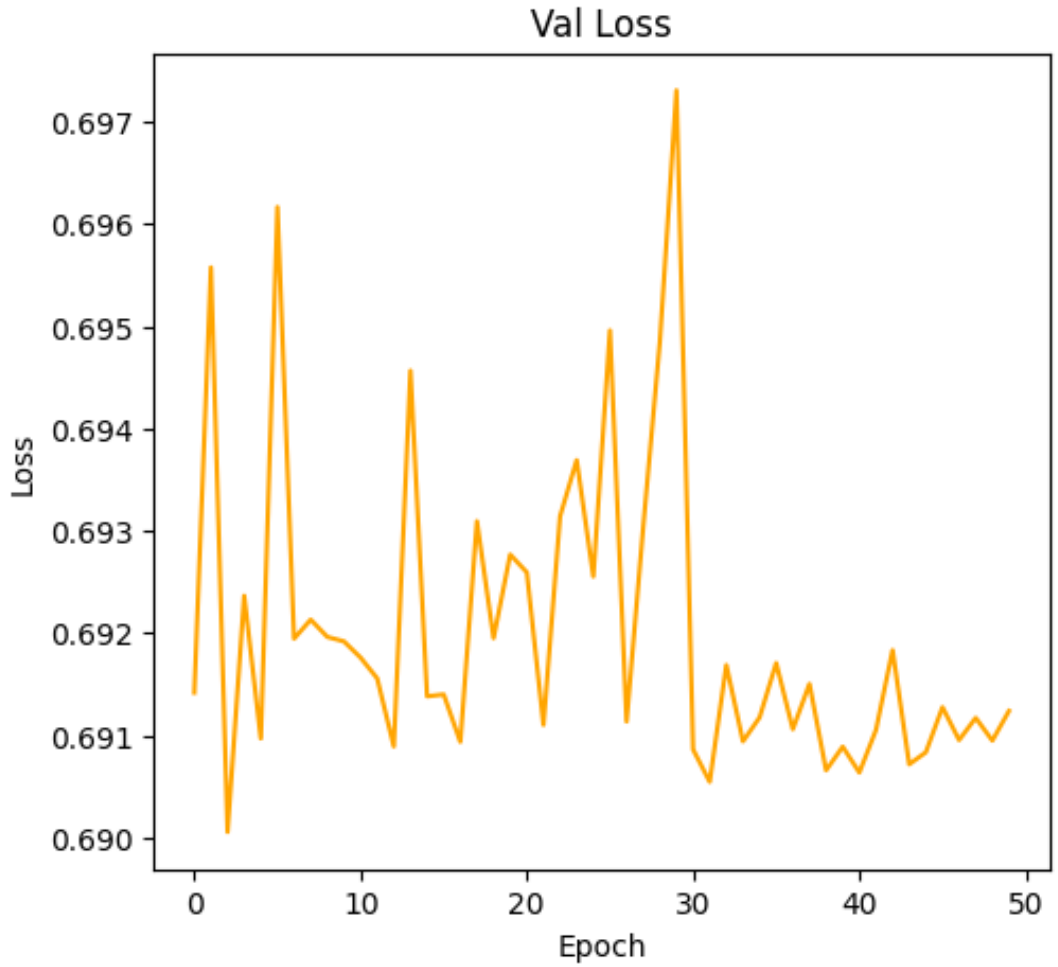
```
train_losses, val_losses = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    scheduler,
    device,
    epochs
)
```

```
EPOCH 1:        Train loss: 0.7225      Val loss: 0.6914
EPOCH 2:        Train loss: 0.6940      Val loss: 0.6956
EPOCH 3:        Train loss: 0.6944      Val loss: 0.6901
EPOCH 4:        Train loss: 0.6944      Val loss: 0.6924
EPOCH 5:        Train loss: 0.6935      Val loss: 0.6910
EPOCH 6:        Train loss: 0.6936      Val loss: 0.6962
EPOCH 7:        Train loss: 0.6950      Val loss: 0.6919
EPOCH 8:        Train loss: 0.6933      Val loss: 0.6921
EPOCH 9:        Train loss: 0.6935      Val loss: 0.6920
EPOCH 10:       Train loss: 0.6944      Val loss: 0.6919
EPOCH 11:       Train loss: 0.6935      Val loss: 0.6918
EPOCH 12:       Train loss: 0.6937      Val loss: 0.6916
EPOCH 13:       Train loss: 0.6931      Val loss: 0.6909
EPOCH 14:       Train loss: 0.6927      Val loss: 0.6946
EPOCH 15:       Train loss: 0.6931      Val loss: 0.6914
EPOCH 16:       Train loss: 0.6926      Val loss: 0.6914
EPOCH 17:       Train loss: 0.6935      Val loss: 0.6909
EPOCH 18:       Train loss: 0.6932      Val loss: 0.6931
EPOCH 19:       Train loss: 0.6922      Val loss: 0.6920
EPOCH 20:       Train loss: 0.6928      Val loss: 0.6928
```

CNN+LSTM Approach

❖ Step 8: Training





CNN+LSTM Approach

❖ Step 9: Evaluation

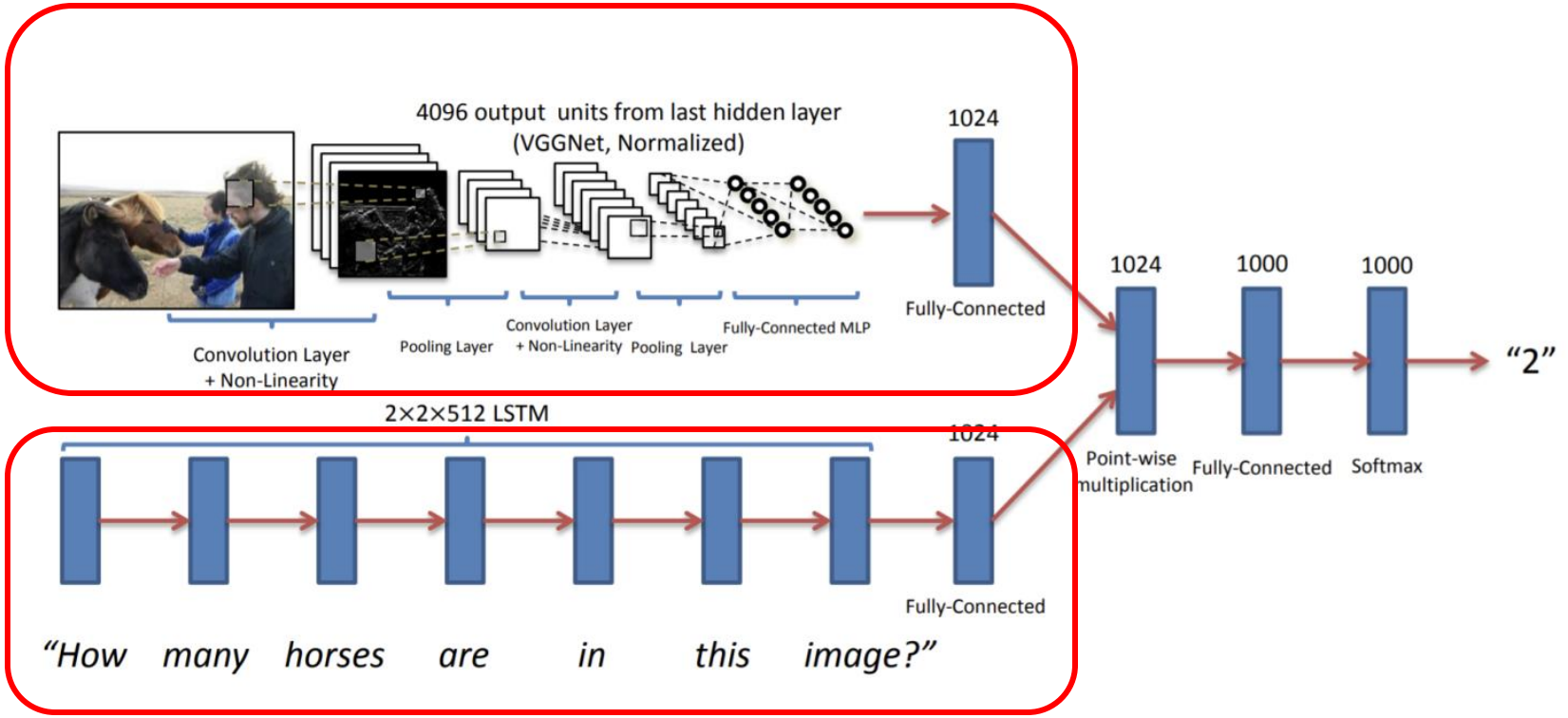
```
val_loss, val_acc = evaluate(
    model,
    val_loader,
    criterion,
    device
)
test_loss, test_acc = evaluate(
    model,
    test_loader,
    criterion,
    device
)

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

```
Evaluation on val/test dataset
Val accuracy:  0.5358606557377049
Test accuracy:  0.5489614243323442
```

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for image, question, labels in dataloader:
            image = image.to(device)
            question = question.to(device)
            labels = 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
```

ViT+RoBERTa Approach

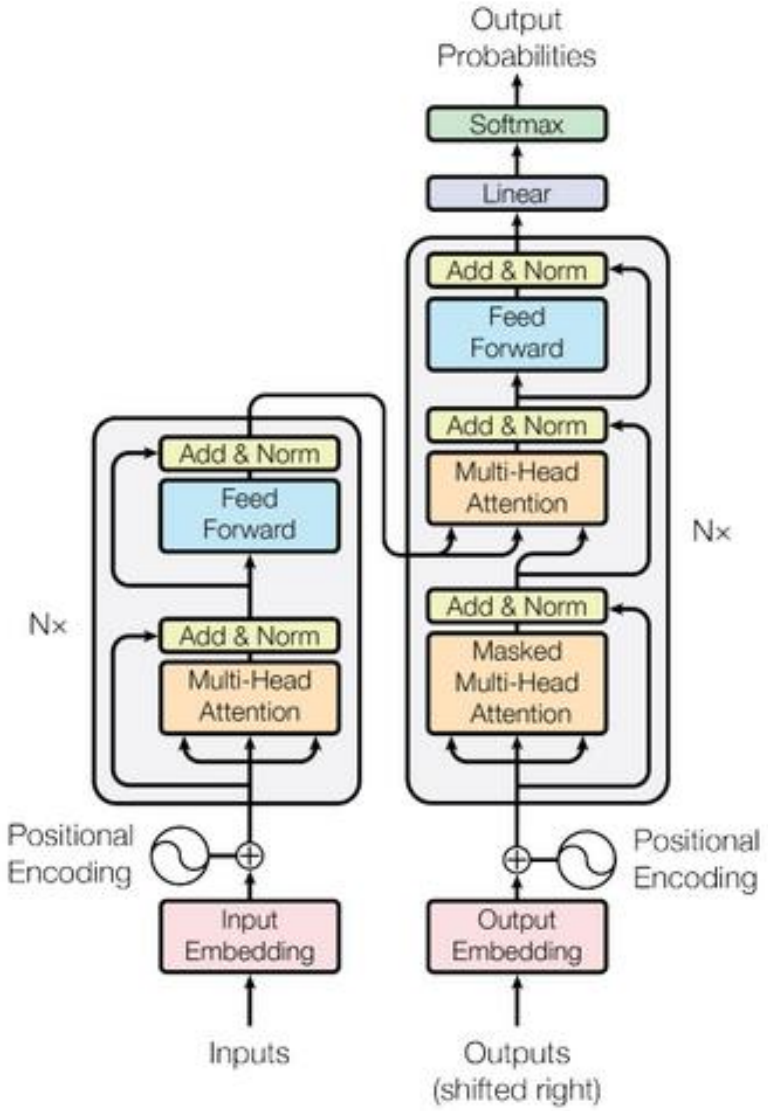
❖ Introduction



How about replace CNN and LSTM with a better model?

ViT+RoBERTa Approach

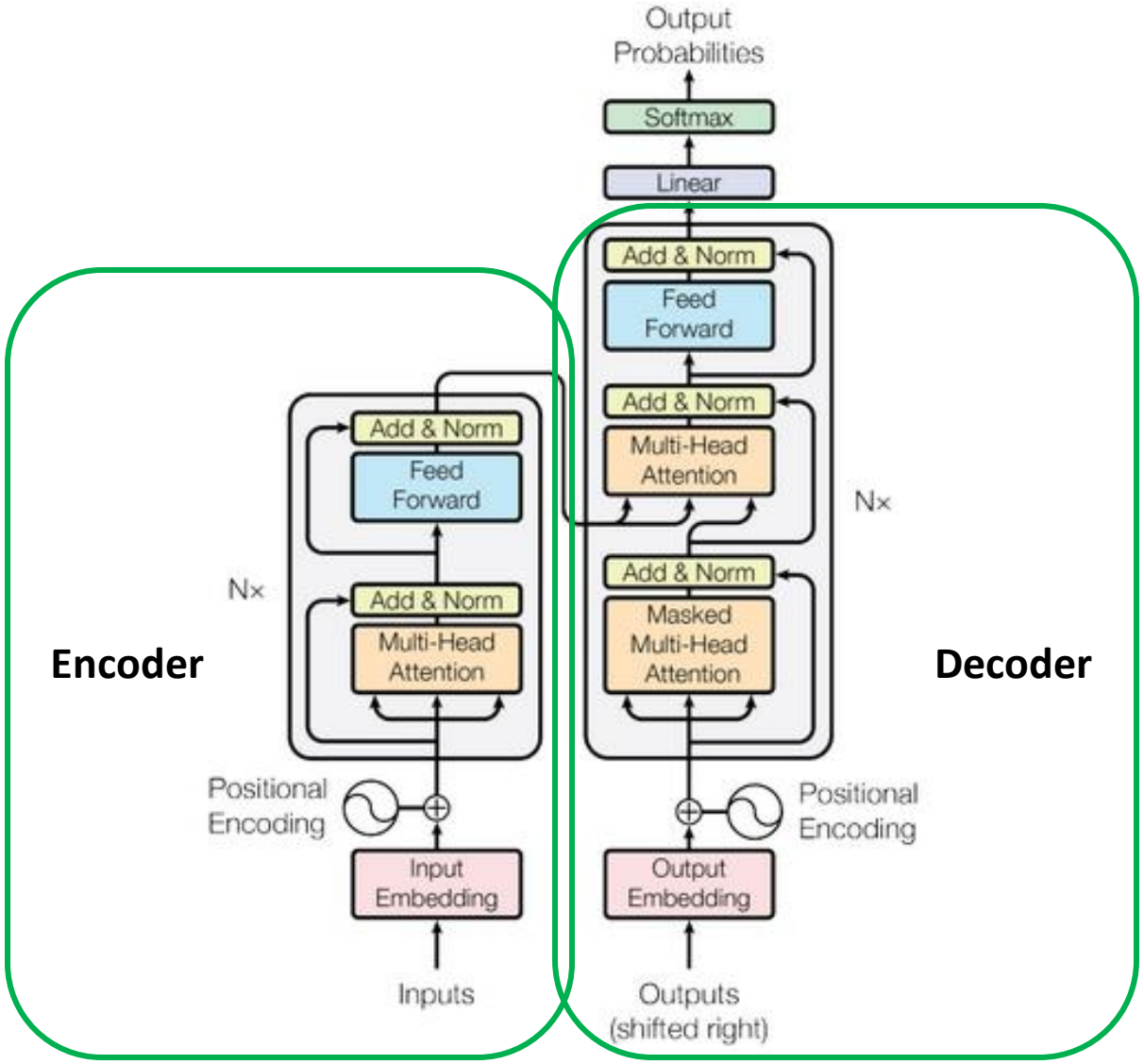
❖ Introduction



Transformers: A Transformer is a type of deep learning architecture that uses an attention mechanism to process text sequences.

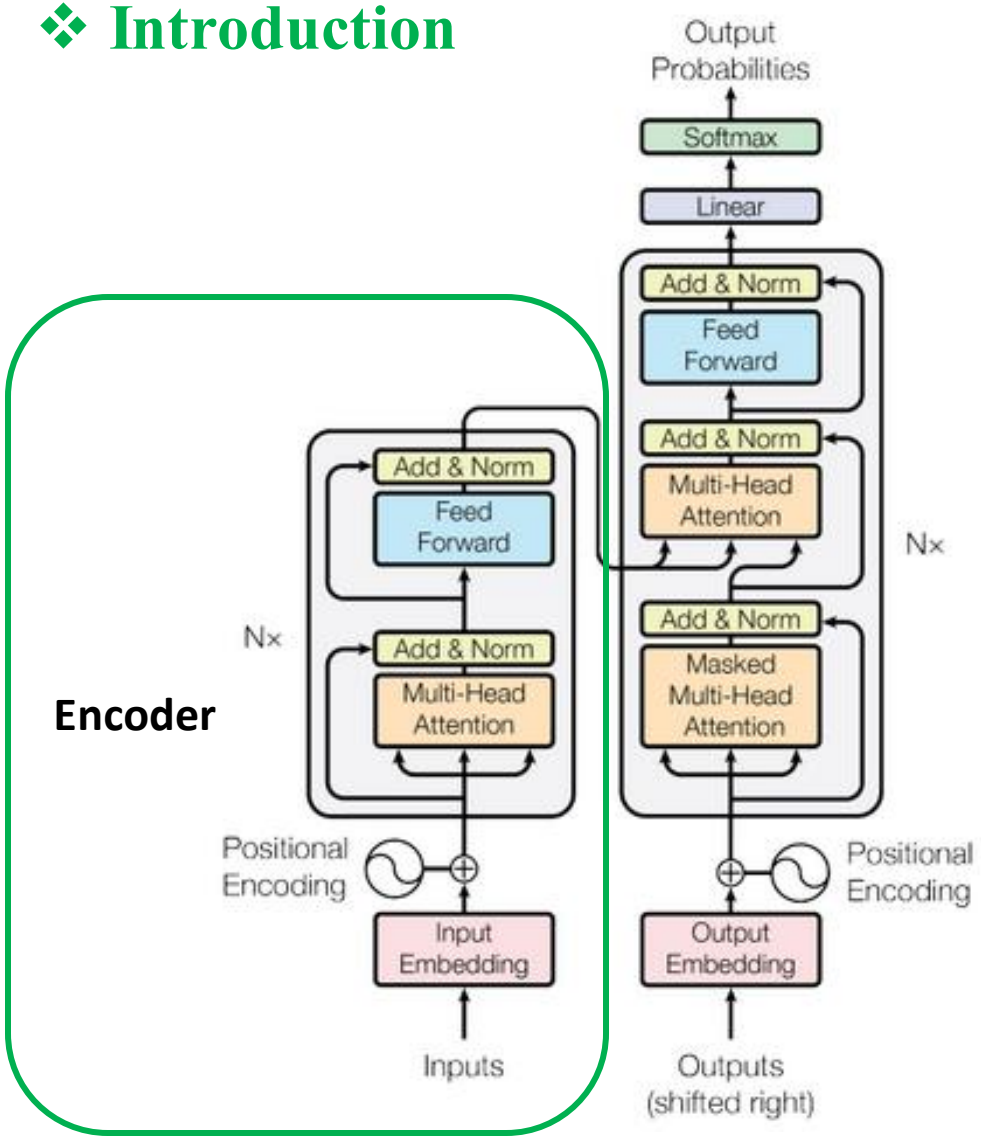
ViT+RoBERTa Approach

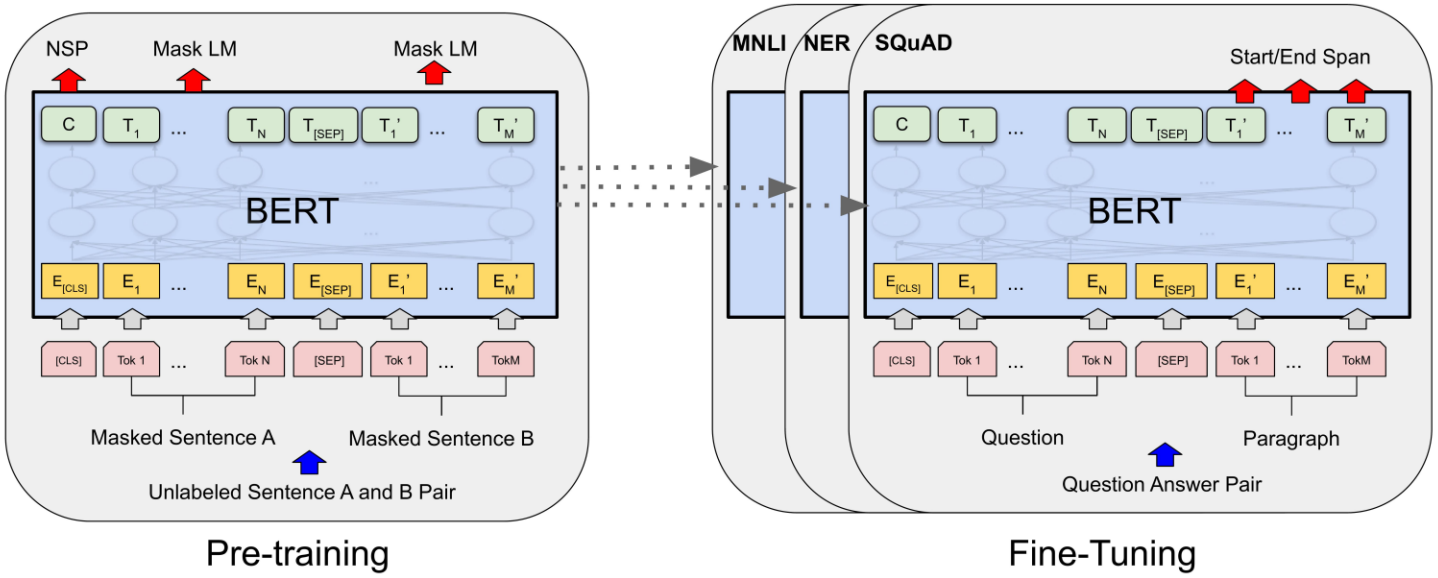
❖ Introduction



ViT+RoBERTa Approach

❖ Introduction



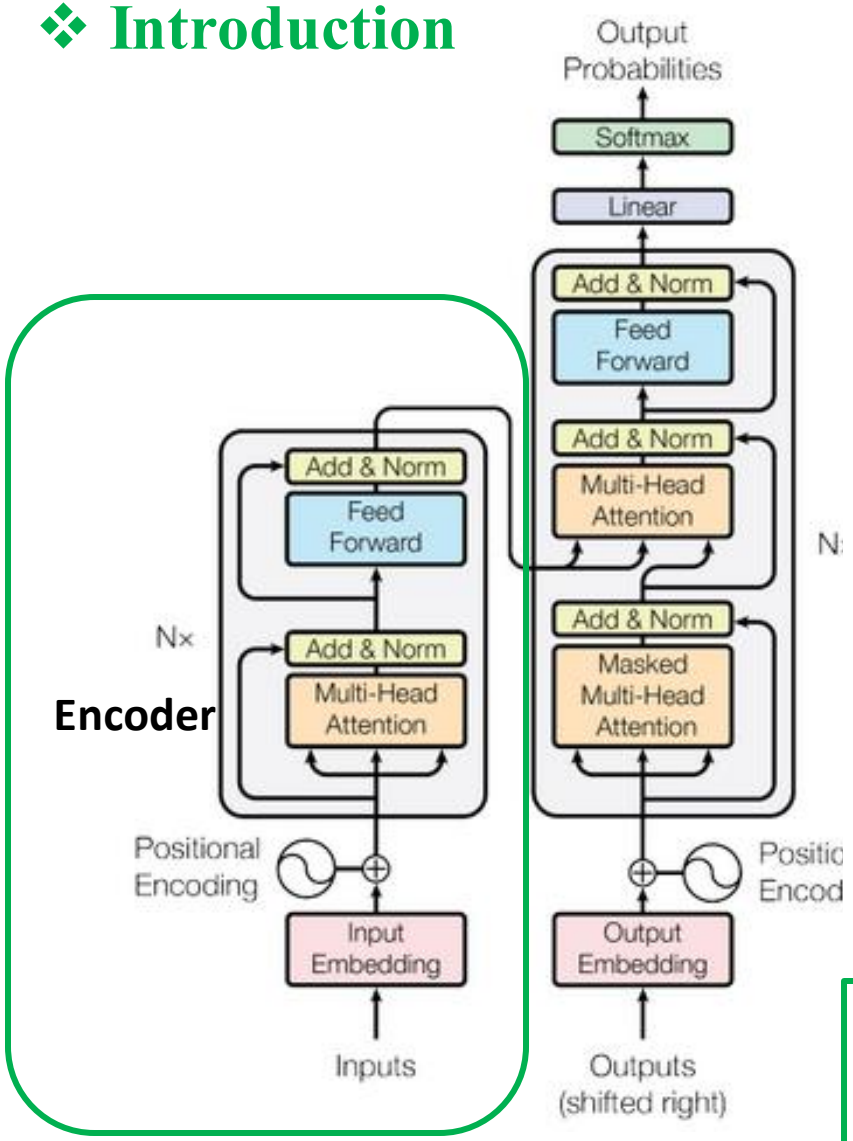


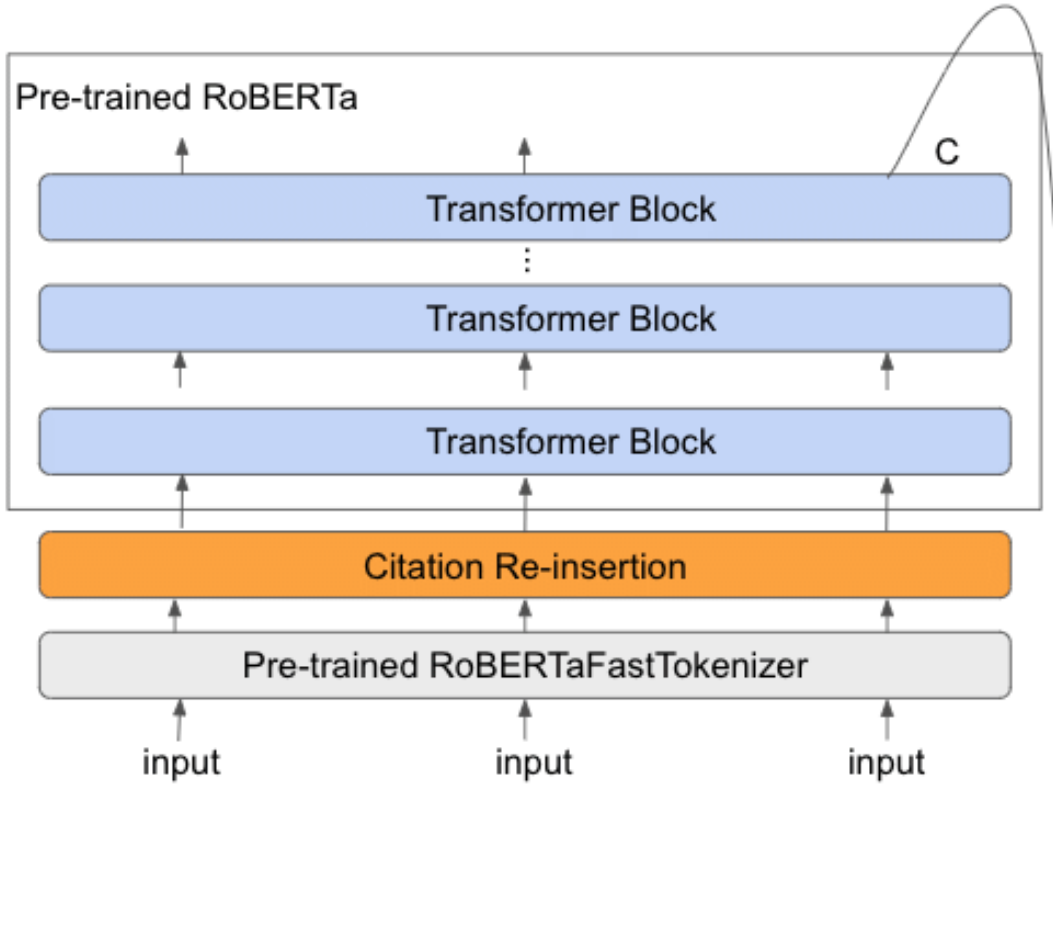
BERT (Bidirectional Encoder Representations from Transformers): A language model based on the transformer architecture.

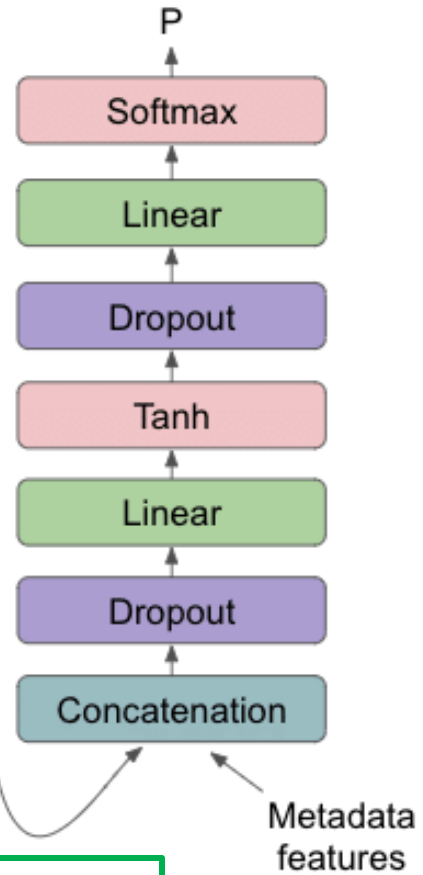
https://arxiv.org/abs/1810.04805

ViT+RoBERTa Approach

❖ Introduction





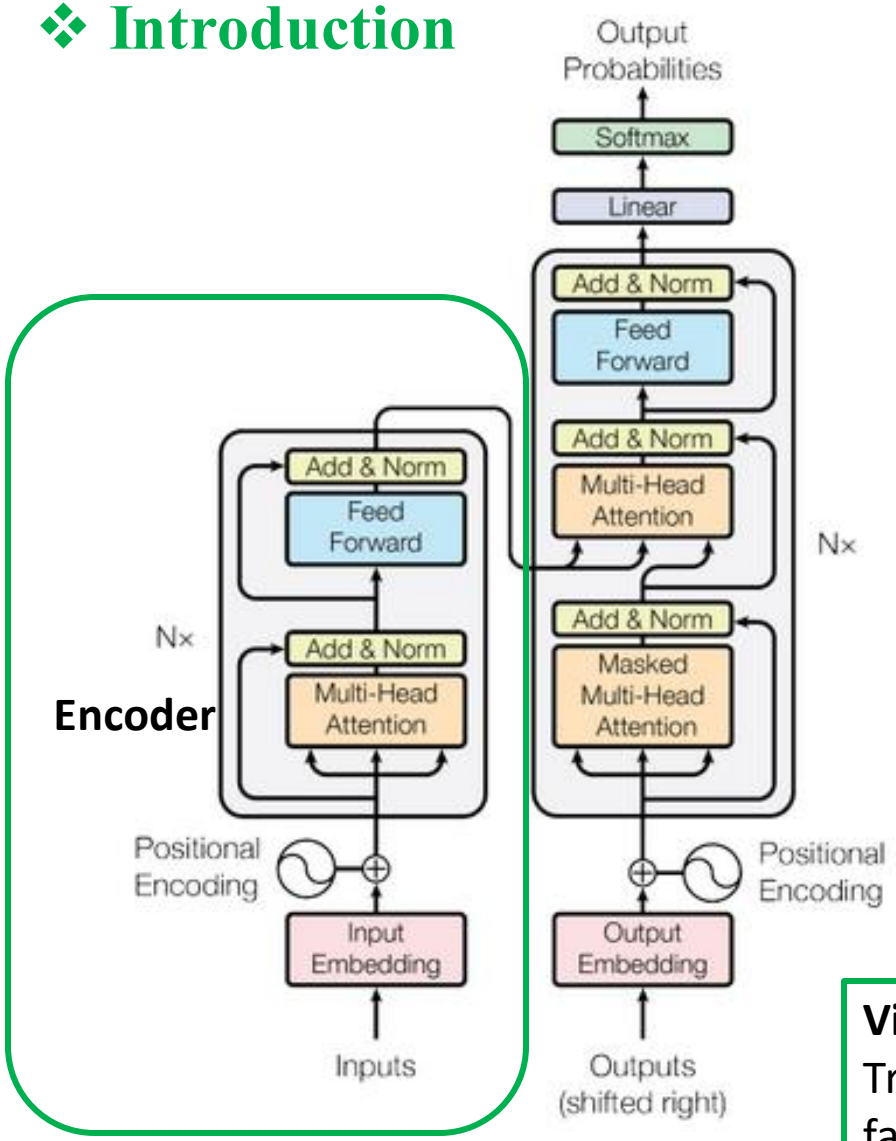


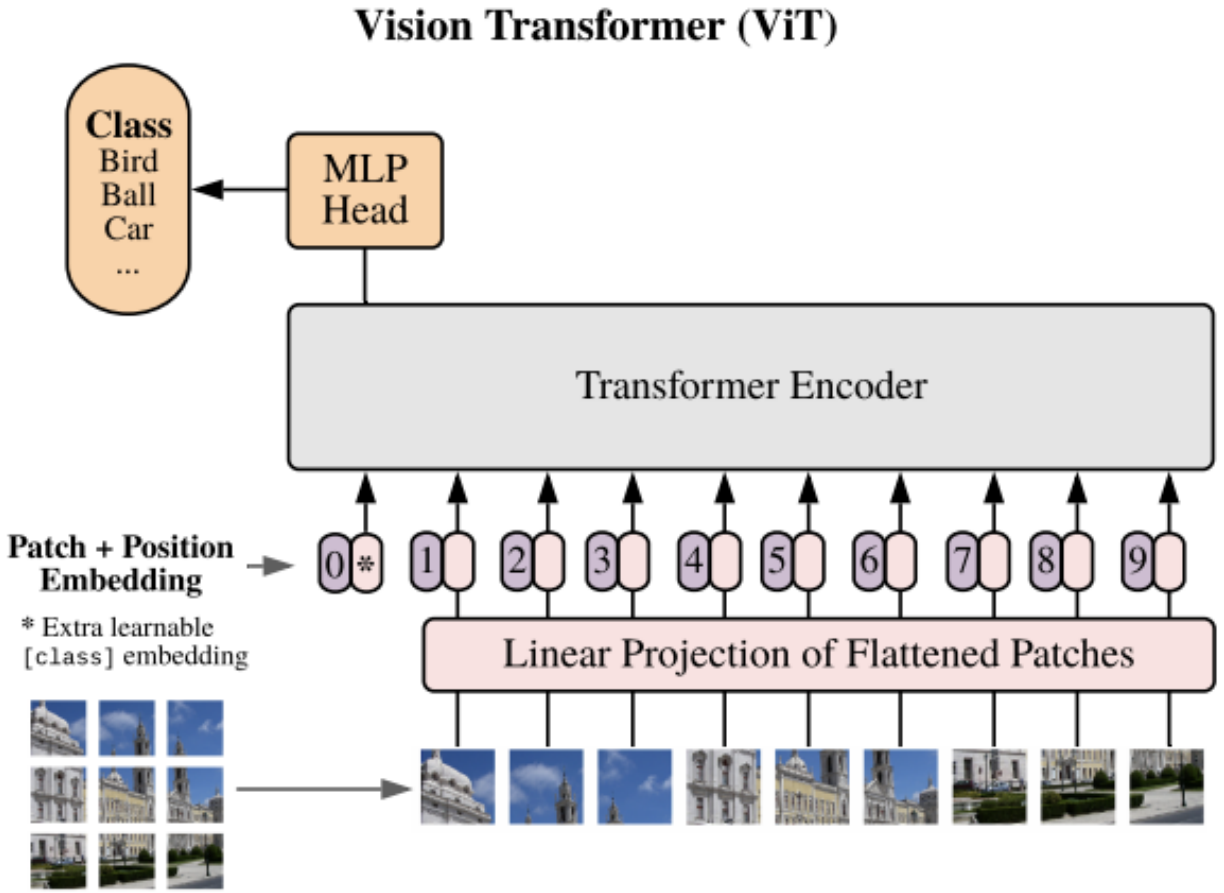
RoBERTa: Builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates.

https://arxiv.org/abs/1907.11692

ViT+RoBERTa Approach

❖ Introduction



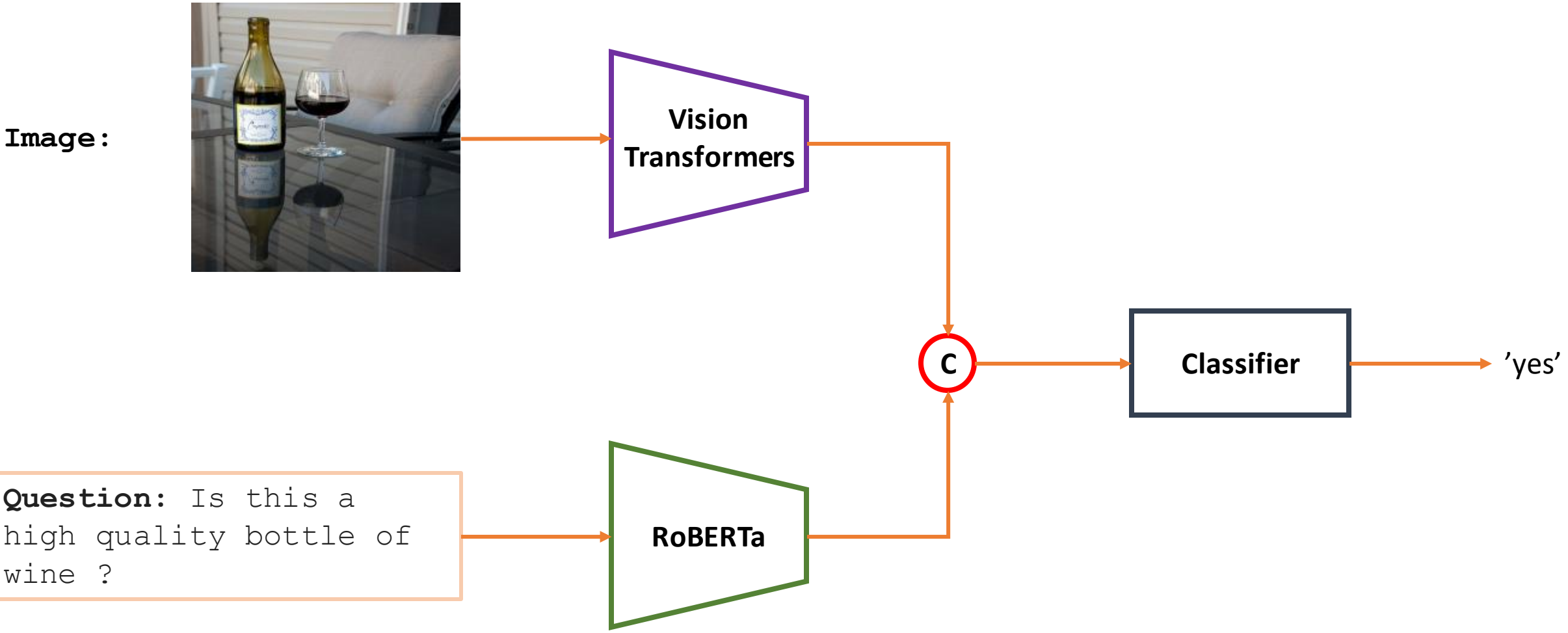


ViT (Vision Transformer): The first paper that successfully trains a Transformer encoder on ImageNet, attaining very good results compared to familiar convolutional architectures

https://arxiv.org/abs/2010.11929

ViT+RoBERTa Approach

❖ Pipeline



ViT+RoBERTa Approach

❖ Step 1: Import libraries

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

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

ViT+RoBERTa Approach

❖ Step 2: Read dataset

```
train_data = []
train_set_path = 'datasets/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)
```

```
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)
}
print(idx_to_classes)
```

```
{0: 'no', 1: 'yes'}
```

```
[{'image_path': 'COCO_val2014_000000393225.jpg',
  'question': 'Is this a creamy soup ?',
  'answer': 'no'},
 {'image_path': 'COCO_val2014_000000393243.jpg',
  'question': 'Is this person wearing a tie ?',
  'answer': 'no'},
 {'image_path': 'COCO_val2014_000000262197.jpg',
  'question': 'Is this a hospital ?',
  'answer': 'yes'},
 {'image_path': 'COCO_val2014_000000393277.jpg',
  'question': 'Are there any tour buses ?',
  'answer': 'no'},
 {'image_path': 'COCO_val2014_000000393277.jpg',
  'question': 'Is this a one way street ?',
  'answer': 'no'},
```

ViT+RoBERTa Approach

❖ Step 3: Create pytorch datasets

```
class VQADataset(Dataset):
    def __init__(
        self,
        data,
        classes_to_idx,
        max_seq_len=20,
        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
```

ViT+RoBERTa Approach

❖ Step 3: Create pytorch datasets

```
class VQADataset(Dataset):
    def __init__(
        self,
        data,
        classes_to_idx,
        max_seq_len=20,
        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
```

Previous CNN+LSTM Approach pytorch dataset

ViT+RoBERTa Approach

❖ Step 3: Create pytorch datasets

Image:

ViTImageProcessor

Vision Transformers

Question: Is this a high quality bottle of wine ?

Tokenizer

RoBERTa

ViT+RoBERTa Approach

❖ Step 3: Create pytorch datasets

```
class VQADataset(Dataset):
    def __init__(
        self,
        data,
        classes_to_idx,
        img_feature_extractor,
        text_tokenizer,
        device,
        transforms=None,
        root_dir='datasets/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
        self.transforms = transforms
```

ViT+RoBERTa Approach

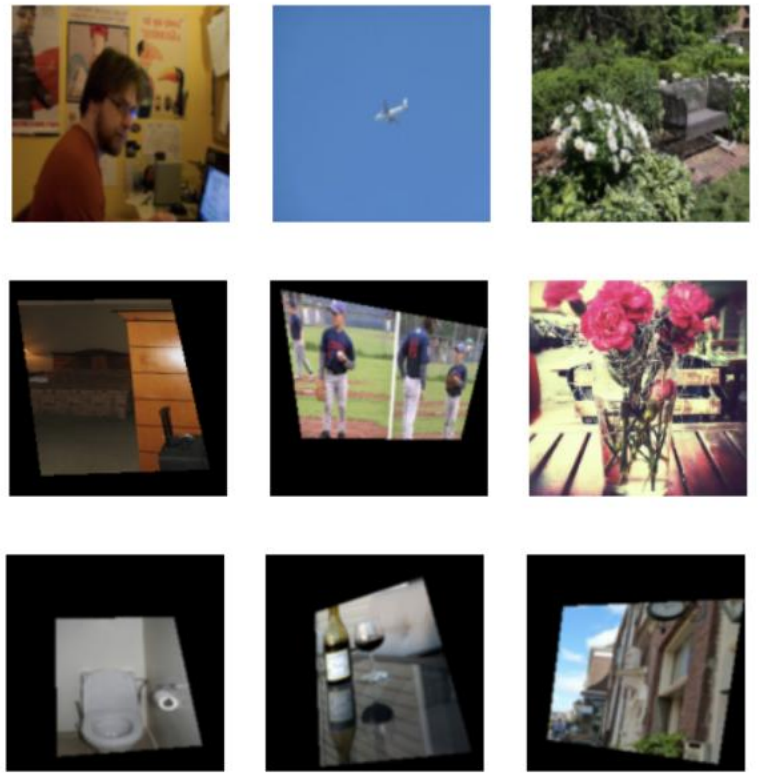
❖ Step 3: Create pytorch datasets

```
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.transforms:
        img = self.transforms(img)

    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
```

ViT+RoBERTa Approach

❖ Step 4: Create data preprocessing function

```
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(size=(224, 224)),
        transforms.CenterCrop(size=180),
        transforms.RandomHorizontalFlip(),
        transforms.ColorJitter(
            brightness=0.1,
            contrast=0.1,
            saturation=0.1
        ),
        transforms.RandomHorizontalFlip(),
        transforms.RandomPerspective(distortion_scale=0.4),
        transforms.GaussianBlur(3),
    ]),
    'val': transforms.Compose([
        transforms.Resize(size=(224, 224))
    ]),
}
```



ViT+RoBERTa Approach

❖ Step 5: Create dataloader

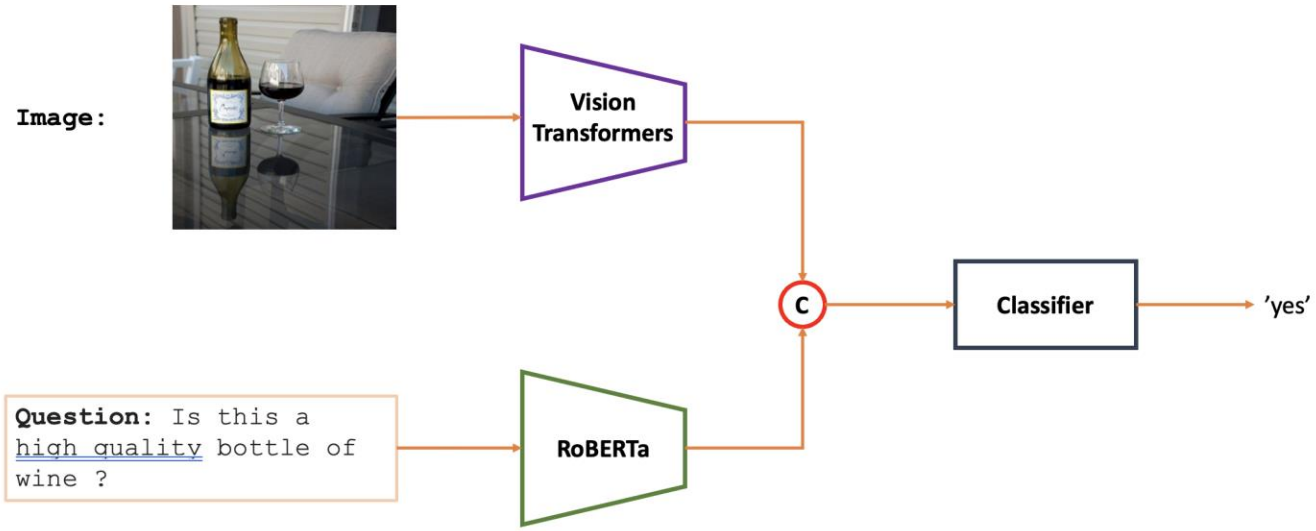
```
train_dataset = VQADataset(
    train_data,
    classes_to_idx=classes_to_idx,
    img_feature_extractor=img_feature_extractor,
    text_tokenizer=text_tokenizer,
    device=device,
    transforms=data_transform
)
val_dataset = VQADataset(
    val_data,
    classes_to_idx=classes_to_idx,
    img_feature_extractor=img_feature_extractor,
    text_tokenizer=text_tokenizer,
    device=device
)
test_dataset = VQADataset(
    test_data,
    classes_to_idx=classes_to_idx,
    img_feature_extractor=img_feature_extractor,
    text_tokenizer=text_tokenizer,
    device=device
)
```

```
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_batch_size = 256
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
)
```

ViT+RoBERTa Approach

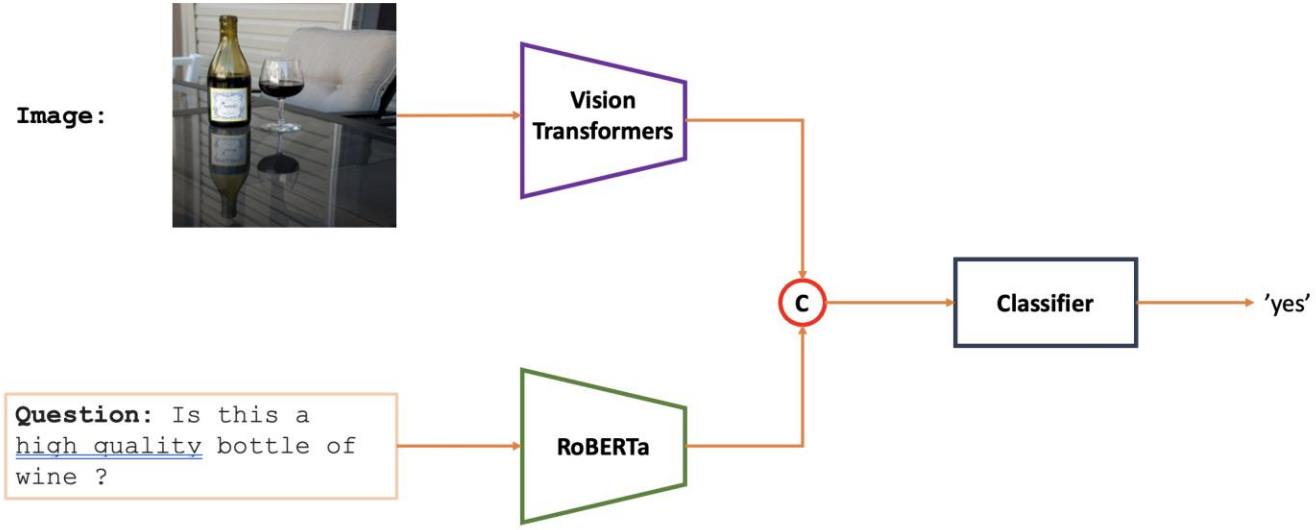
❖ Step 6: Create model




```python
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
```

ViT+RoBERTa Approach

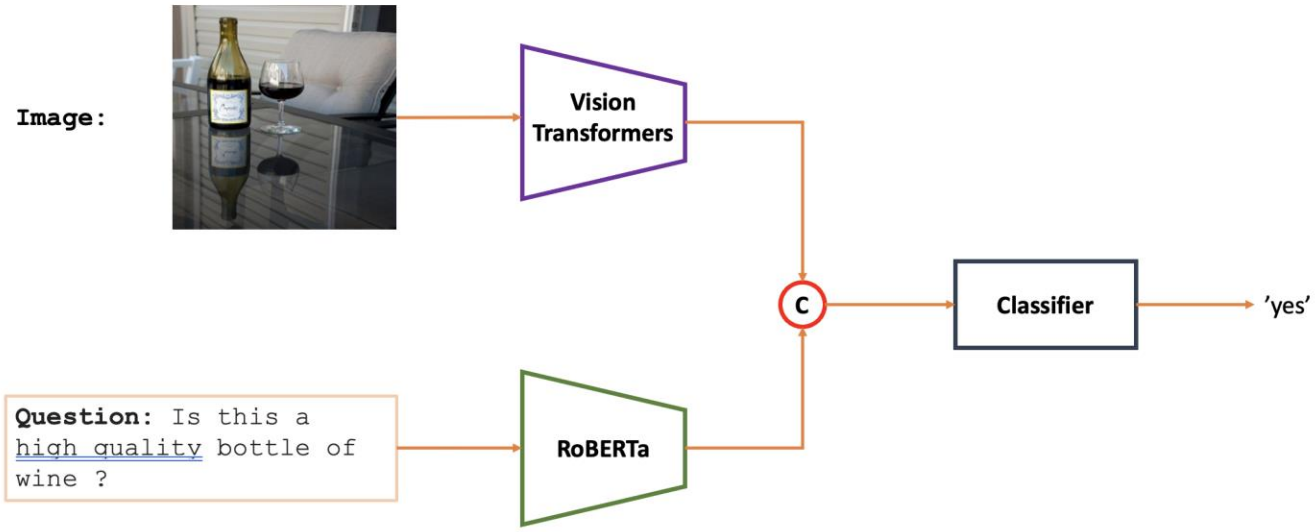
❖ Step 6: Create model




```python
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
```

ViT+RoBERTa Approach

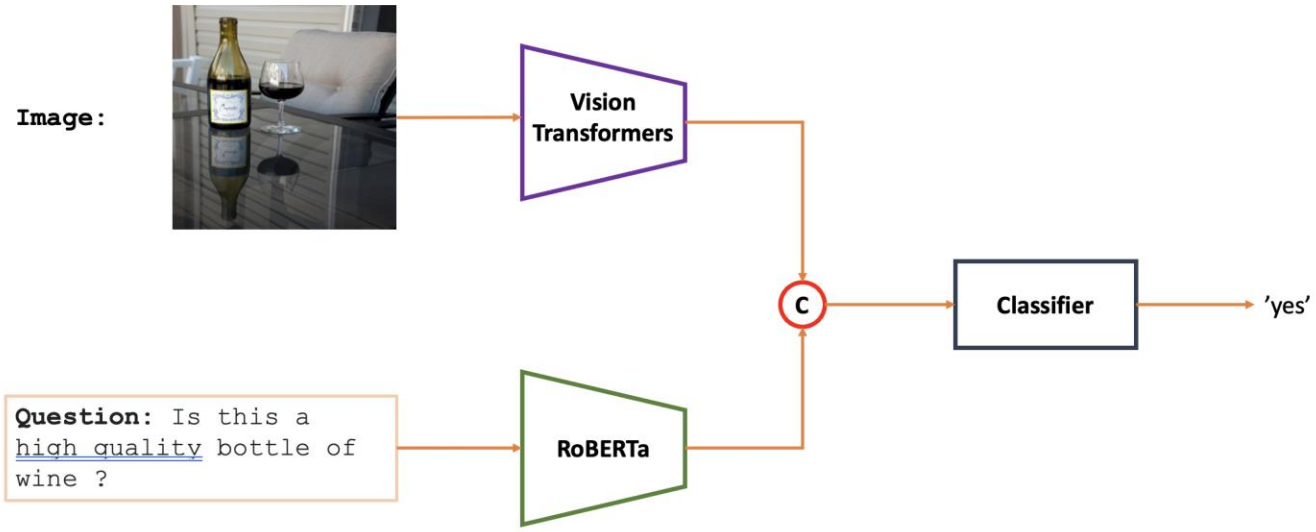
❖ Step 6: Create model




```python
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
```

ViT+RoBERTa Approach

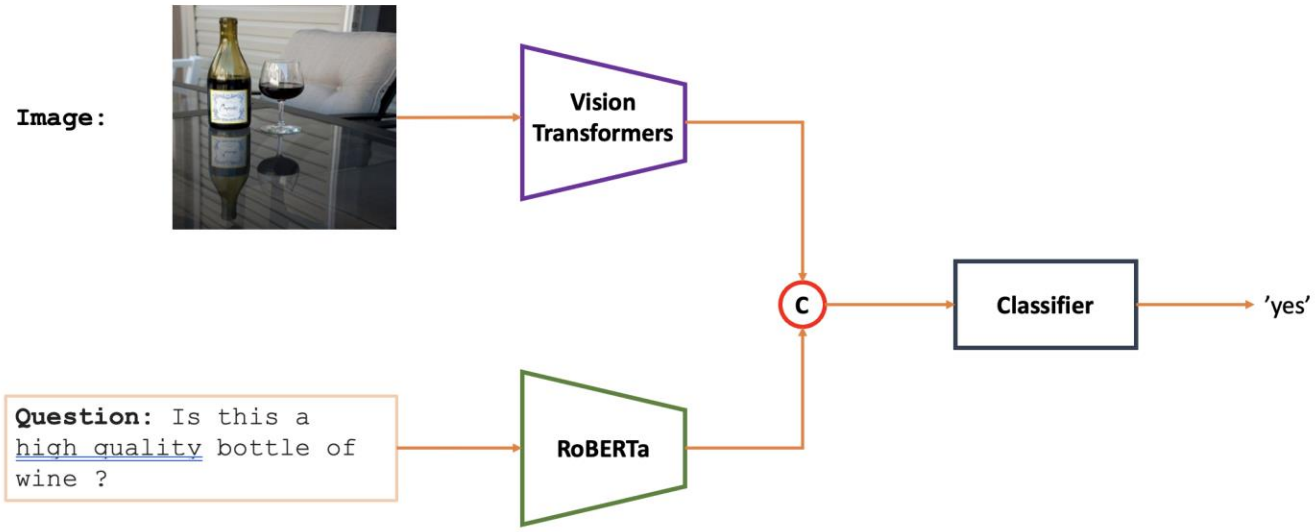
❖ Step 6: Create model




```python
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
```

ViT+RoBERTa Approach

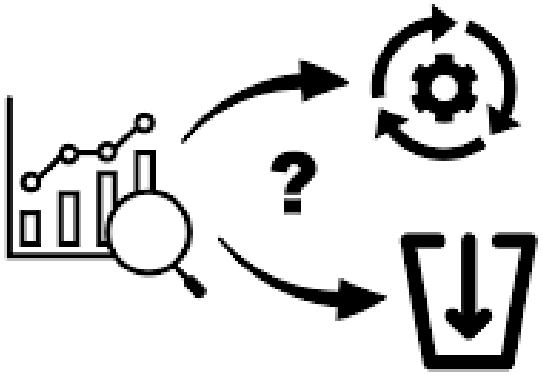
❖ Step 6: Create model

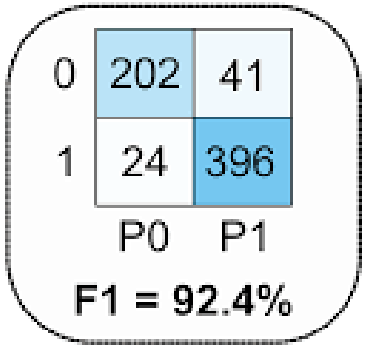



```
n_classes = len(classes)
hidden_size = 256
n_layers = 2
dropout_prob = 0.2

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()
```

ViT+RoBERTa Approach

❖ Step 7: Create evaluate and training function






```python
def evaluate(model, dataloader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for image, question, labels in dataloader:
            image = image.to(device)
            question = question.to(device)
            labels = 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
```

ViT+RoBERTa Approach

❖ Step 7: Create evaluate and training function

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

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

        train_correct = 0
        train_total = 0

        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)
            _, predicted = torch.max(outputs.data, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()
            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)

        train_acc = train_correct / train_total
        train_accs.append(train_acc)

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

        print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tTrain acc:

        scheduler.step()

    return train_losses, val_losses, train_accs, val_accs
```

ViT+RoBERTa Approach

❖ Step 8: Training

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

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

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

```
EPOCH 1:        Train loss: 0.6970      Val loss: 0.6900        Val Acc: 0.5358606557377049
EPOCH 2:        Train loss: 0.6897      Val loss: 0.6858        Val Acc: 0.5466188524590164
EPOCH 3:        Train loss: 0.6812      Val loss: 0.6849        Val Acc: 0.5589139344262295
EPOCH 4:        Train loss: 0.6714      Val loss: 0.6765        Val Acc: 0.5840163934426229
EPOCH 5:        Train loss: 0.6545      Val loss: 0.6740        Val Acc: 0.5901639344262295
EPOCH 6:        Train loss: 0.6334      Val loss: 0.6738        Val Acc: 0.5993852459016393
EPOCH 7:        Train loss: 0.6047      Val loss: 0.6684        Val Acc: 0.6060450819672131
EPOCH 8:        Train loss: 0.5681      Val loss: 0.6849        Val Acc: 0.6065573770491803
EPOCH 9:        Train loss: 0.5151      Val loss: 0.7003        Val Acc: 0.6209016393442623
EPOCH 10:       Train loss: 0.4667      Val loss: 0.7066        Val Acc: 0.6460040983606558
EPOCH 11:       Train loss: 0.4201      Val loss: 0.7514        Val Acc: 0.6398565573770492
EPOCH 12:       Train loss: 0.3923      Val loss: 0.7326        Val Acc: 0.649077868852459
EPOCH 13:       Train loss: 0.3428      Val loss: 0.7740        Val Acc: 0.6521516393442623
EPOCH 14:       Train loss: 0.3251      Val loss: 0.8054        Val Acc: 0.6506147540983607
EPOCH 15:       Train loss: 0.3069      Val loss: 0.8258        Val Acc: 0.6552254098360656
EPOCH 16:       Train loss: 0.2895      Val loss: 0.8004        Val Acc: 0.6521516393442623
EPOCH 17:       Train loss: 0.2699      Val loss: 0.8237        Val Acc: 0.65625
EPOCH 18:       Train loss: 0.2540      Val loss: 0.8453        Val Acc: 0.6567622950819673
EPOCH 19:       Train loss: 0.2430      Val loss: 0.9236        Val Acc: 0.6552254098360656
EPOCH 20:       Train loss: 0.2416      Val loss: 0.9300        Val Acc: 0.65625
EPOCH 21:       Train loss: 0.2259      Val loss: 0.9083        Val Acc: 0.6577868852459017
EPOCH 22:       Train loss: 0.2147      Val loss: 0.9157        Val Acc: 0.6613729508196722
EPOCH 23:       Train loss: 0.2183      Val loss: 0.8782        Val Acc: 0.673155737704918
EPOCH 24:       Train loss: 0.2056      Val loss: 0.9739        Val Acc: 0.6577868852459017
EPOCH 25:       Train loss: 0.1978      Val loss: 0.9554        Val Acc: 0.6685450819672131
EPOCH 26:       Train loss: 0.1929      Val loss: 0.9313        Val Acc: 0.663422131147541
EPOCH 27:       Train loss: 0.1998      Val loss: 0.9055        Val Acc: 0.6659836065573771
```

ViT+RoBERTa Approach

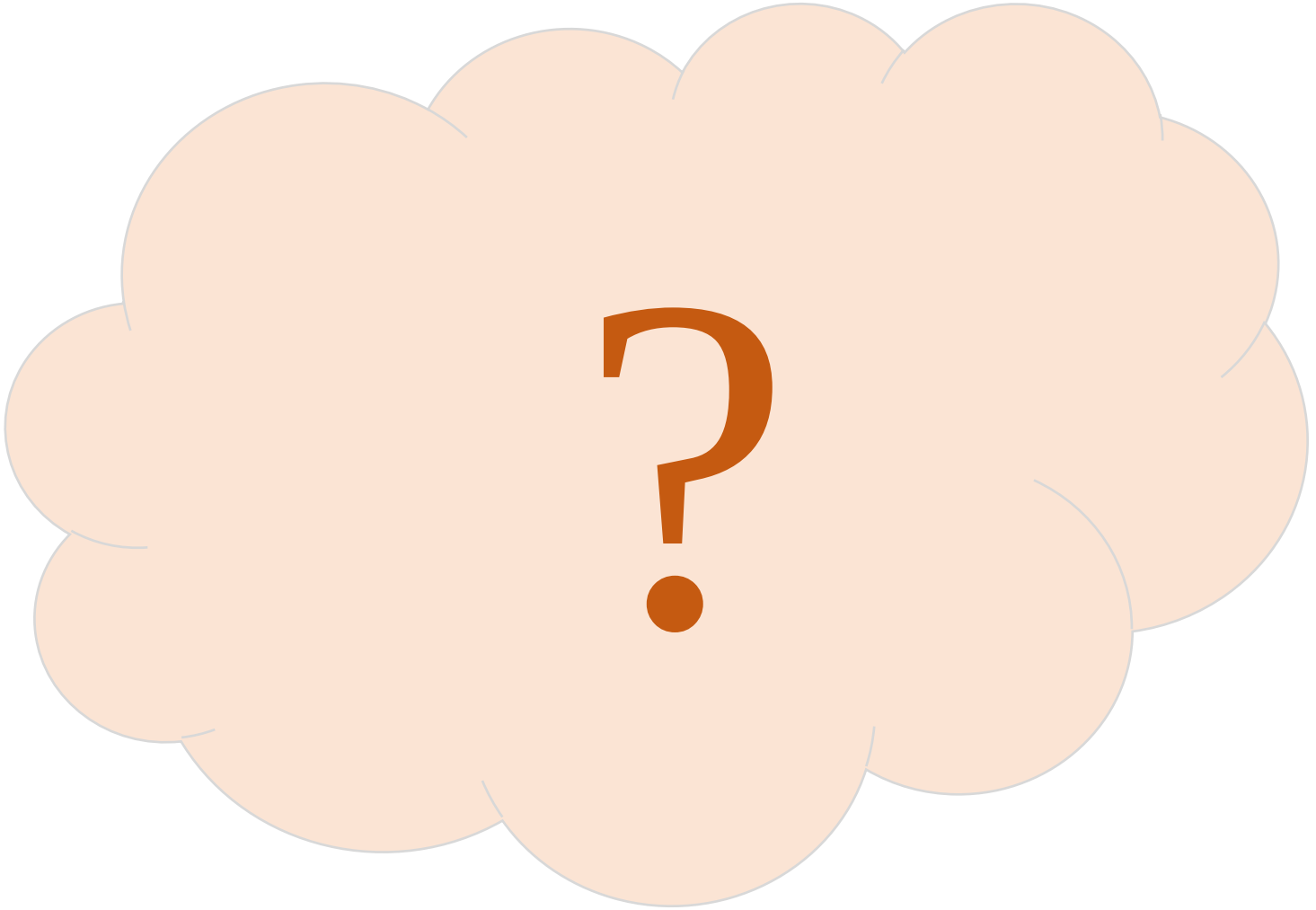
❖ Step 9: Evaluation

```
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)
```

```
Evaluation on val/test dataset
Val accuracy:  0.7054303278688525
Test accuracy:  0.701780415430267
```

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for image, question, labels in dataloader:
            image = image.to(device)
            question = question.to(device)
            labels = 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
```

Question




Thank you!