

CLIP

Contrastive Language-Image Pre-Training : 对比语言图像预训练

解决问题：如何把图像和文本放在一起。

论文

Learning Transferable Visual Models from Natural Language Supervision(利用文本的监督信号训练一个迁移能力强的视觉模型)

传统：

训练1000个类别，预测就只能限制于这1000个类别，无法扩展。

新增类别还要重新训练、重新标准。——重复工作、拓展能力（泛化能力、迁移能力）差

CLIP：预训练模型直接**zero-shot**——给提示就能完成任务

CLIP论文指出：以前的方法不行，是资源不到位。——更大的模型、数据。

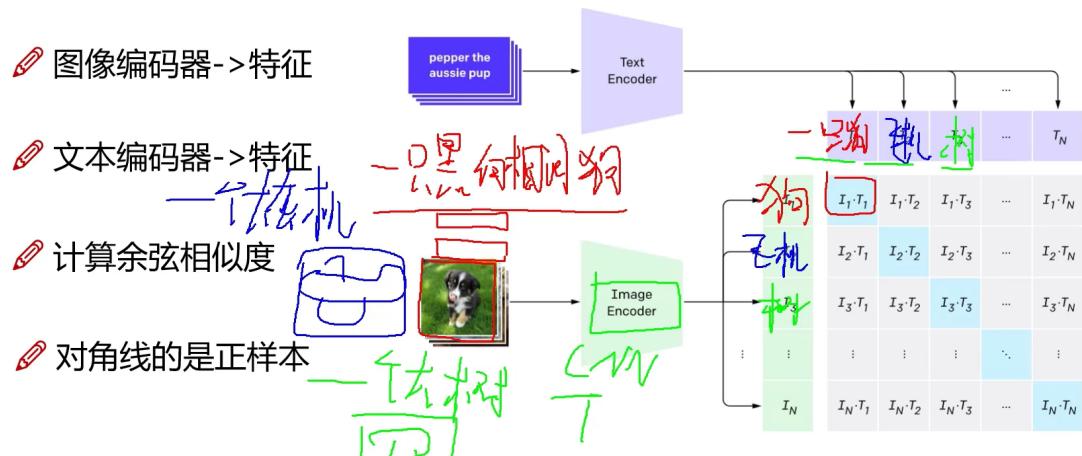
成名一战：

CLIP在完全不使用ImageNet中所有数据训练的前提下：

- 直接Zero-shot得到的结果与Resnet在128W Imagenet数据训练后效果一样
- 使用4亿个配对的数据和文本来进行训练，直接爬取的，不标注
- 现在CLIP下游任务已经很多了，GAN，检测，分割，检索等

如何训练模型：对比学习

✓ 如何训练模型



不做预测，做对比。

对比学习：

正样本：图像和样本配对，就是正样本对；（上图矩阵对角线）

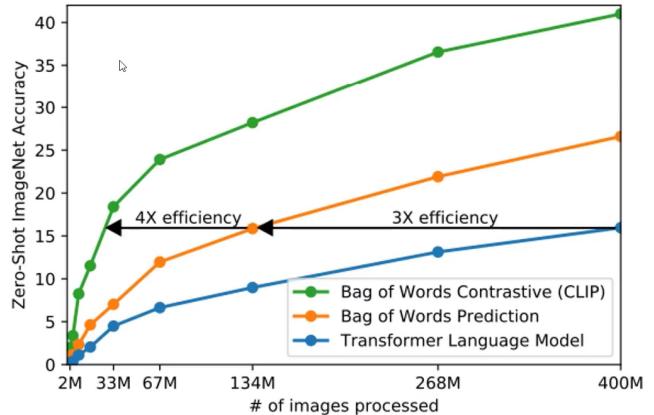
负样本：图像和样本不配对，图是狗，文本是树；

希望正样本相似度越大越好：让对角线相似度最大。

训练策略

✓ 训练策略

- 📎 这种规模的训练要想想招了
- 📎 论文里说了好多GPU YEAR
- 📎 4X就是对比学习的效率提升
- 📎 只看配对不，不预测具体词



✓ 合理的提示

- 📎 预测的时候提示也很重要
- 📎 首先得是一句话来描述
- 📎 而且最好跟你要预测的场景相关
- 📎 提示的也全面，结果也会提升

对比学习效率比预测提高4倍。

合理运用prompt提示。

specify the category. For example on Oxford-IIIT Pets, using “A photo of a {label}, a type of pet.” to help provide context worked well. Likewise, on Food101 specifying *a type of food* and on FGVC Aircraft *a type of aircraft* helped too. For OCR datasets, we found that putting quotes around the text or number to be recognized improved performance. Finally, we found that on satellite image classification datasets it helped to specify that the images were of this form and we use variants of “*a satellite photo of a {label}*.”.

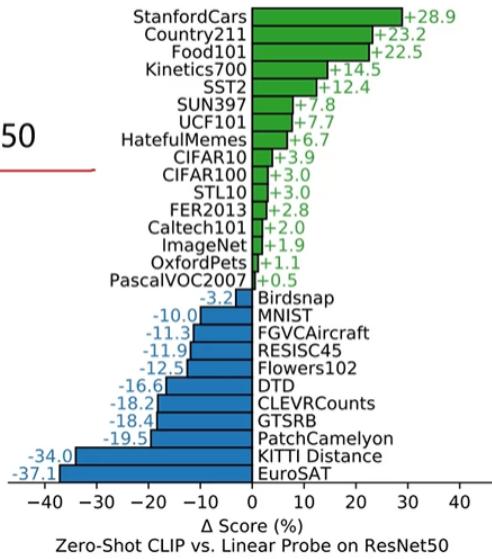
✓ 实验对比

📝 基础模型是Imagenet预训练的Resnet50

📝 不同数据集只训练最后一层FC来微调

📝 绿色的表示CLIP比微调的Resnet强的

📝 蓝色的是有所下降的数据集
(非常规场景的数据都有所下降)



CLIP在MINIST数据集效果不好，只有89%：因为MINIST是人工合成的，不属于实际场景。

可见，clip比较适合实际常见场景。 (zero-shot)

如何推理

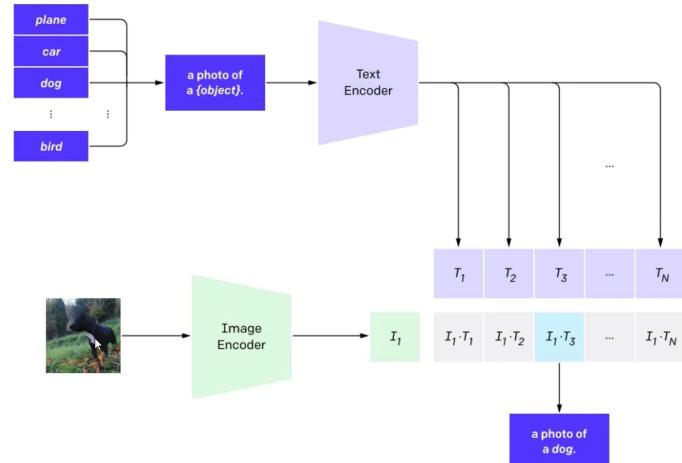
✓ 如何进行推理

📝 给一些提示文本(任意个)

📝 提示要好好说话

📝 然后每种提示算相似度

📝 找到概率最高的即可



提示：把类别扩展成一句话

CLIP的局限

跟Resnet50当成平手，但是Resnet50也不是标杆

OpenAi说如果要跟标杆（比如ViT）比较，数据集需要1000X（扩大1000倍）

只能面向常规的任务

- 在细分类的任务上，效果比有监督的差；
- 在Mnist（人工合成的数字数据集）效果一般，因为mnist对clip是OCR数据（out of distribution），

一些复杂的任务也不行：

- 比如数一数图片里多少个物体、在视频里区分异常和非异常

面向测试集调参?就跟Imagenet干上了, 参数都适合imagenet

代码：CLIP包

```
In [ ]: import torch
import clip
from PIL import Image

# Load the pre-trained CLIP model
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device) # 保留路径: C:\Users\HP\.co
```

```
In [ ]: # Load image
image_path = "F:\FNii\VLM\image.png"
image = Image.open(image_path)

# Define your input image and text
image = preprocess(image).unsqueeze(0).to(device) # Preprocess your image
text = clip.tokenize(["city", "people", "park"]).to(device) # Tokenize your text, 必

# Encode the image and text
with torch.no_grad():
    image_features = model.encode_image(image)
    text_features = model.encode_text(text)

    logits_per_image, logits_per_text = model(image, text)
    probs = logits_per_image.softmax(dim=-1).cpu().numpy()

# Perform similarity calculation
# similarity = (100.0 * image_features @ text_features.T).softmax(dim=-1)

# Print the similarity scores
print("Label probs:", probs)
```

Label probs: [[0.981818 0.01033928 0.00784269]]

```
In [ ]: # unsqueeze(0): 在维度0上增加一个维度

# unsqueeze()函数是在张量 (Tensor) 上操作的方法, 它的作用是在指定的维度上增加一个维

# 举个例子来说明, 假设有一个形状为(3,)的一维张量, 即一个包含3个元素的向量。调用unsg

# 这个函数在某些情况下非常有用, 例如在进行某些运算时需要保持张量的维度一致, 或者在将

# 需要注意的是, unsqueeze()函数返回的是一个新的张量, 原始张量并没有被修改。因此, 在使
```

```
In [ ]: image_path = "F:\FNii\VLM\image.png"
image = Image.open(image_path)

pre_image = preprocess(image)
pre_image.shape
```

```
In [ ]: pre_image.unsqueeze(0).shape
```

代码：Huggingface

```
In [ ]: from PIL import Image
from transformers import CLIPProcessor, CLIPModel
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")#openai/clip-vit-
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

preprocessor_config.json: 0% | 0.00/316 [00:00<?, ?B/s]
tokenizer_config.json: 0% | 0.00/592 [00:00<?, ?B/s]
vocab.json: 0% | 0.00/862k [00:00<?, ?B/s]
merges.txt: 0% | 0.00/525k [00:00<?, ?B/s]
tokenizer.json: 0% | 0.00/2.22M [00:00<?, ?B/s]
special_tokens_map.json: 0% | 0.00/389 [00:00<?, ?B/s]
```

```
In [ ]: import requests
url = "http://images.cocodataset.org/val2017/000000039769.jpg"
image = Image.open(requests.get(url, stream=True).raw)
image
```

Out[]:



```
In [ ]: # image = Image.open('F:\FNii\VLM\image.png')
text = ['dog', 'cat', 'tiger'] # 必须英文
inputs = processor(text=text, images=image, return_tensors="pt", padding=True)
outputs = model(**inputs)
logits_per_image = outputs.logits_per_image# this is the image-text similarity score
logits_per_image # tensor([[18.2806, 23.2766, 21.0254]], grad_fn=<TBackward0>)
```

```
In [ ]: probs = logits_per_image.softmax(dim=1)# we can take the softmax to get the label p
for i in range(len(text)):
    print(text[i], ':', probs[0][i])
```

```
dog : tensor(0.0061, grad_fn=<SelectBackward0>)
cat : tensor(0.8993, grad_fn=<SelectBackward0>)
tiger : tensor(0.0947, grad_fn=<SelectBackward0>)
```

图像生成：DALL-E

从文本生成图片

TEXT PROMPT

an illustration of a baby daikon radish in a tutu walking a dog

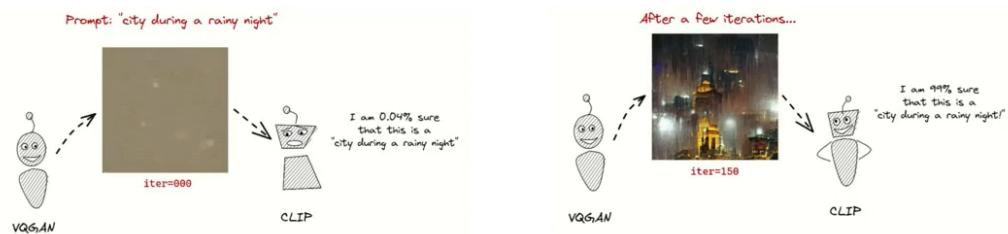
AI-GENERATED IMAGES



Edit prompt or view more images↓

VQGAN

- ✓ 得先熟悉下VQGAN这个家伙 (Vector Quantized)
- ✍ 它跟CLIP有啥关系呢？我们现在希望生成一些图像结果
- ✍ VQGAN就相当于生成器， CLIP就相当于判断器（看生成结果与描述相同不）

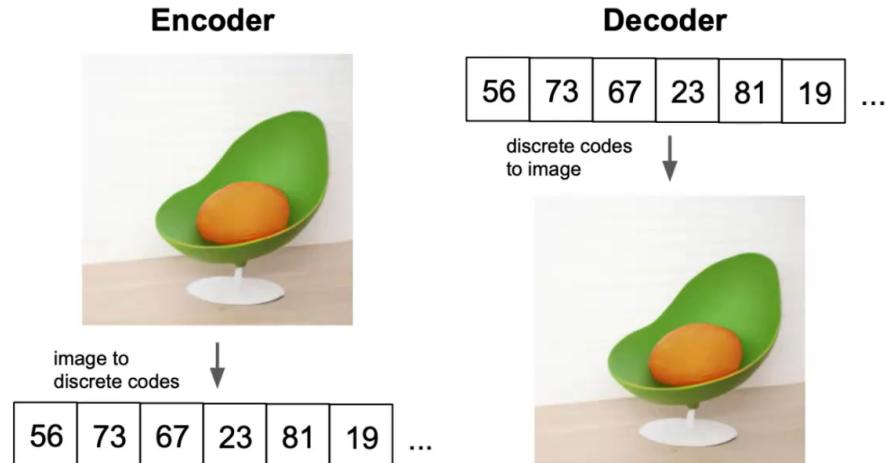


对抗生成网络

图像如何表示

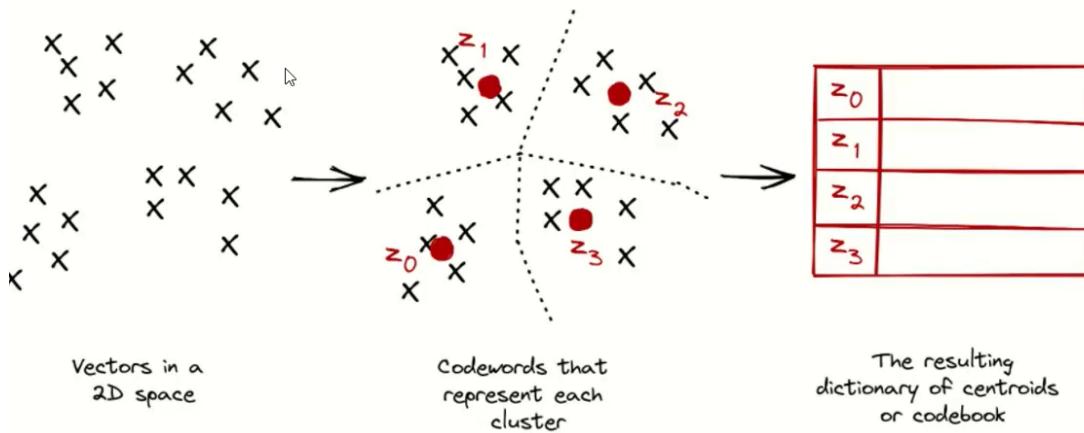
✓ 图像如何表示

📝 NLP中我们讲文本向量化，这里我们能否对图像离散向量化呢？



1、先把特征离散化再整合

离散化特征：查表：



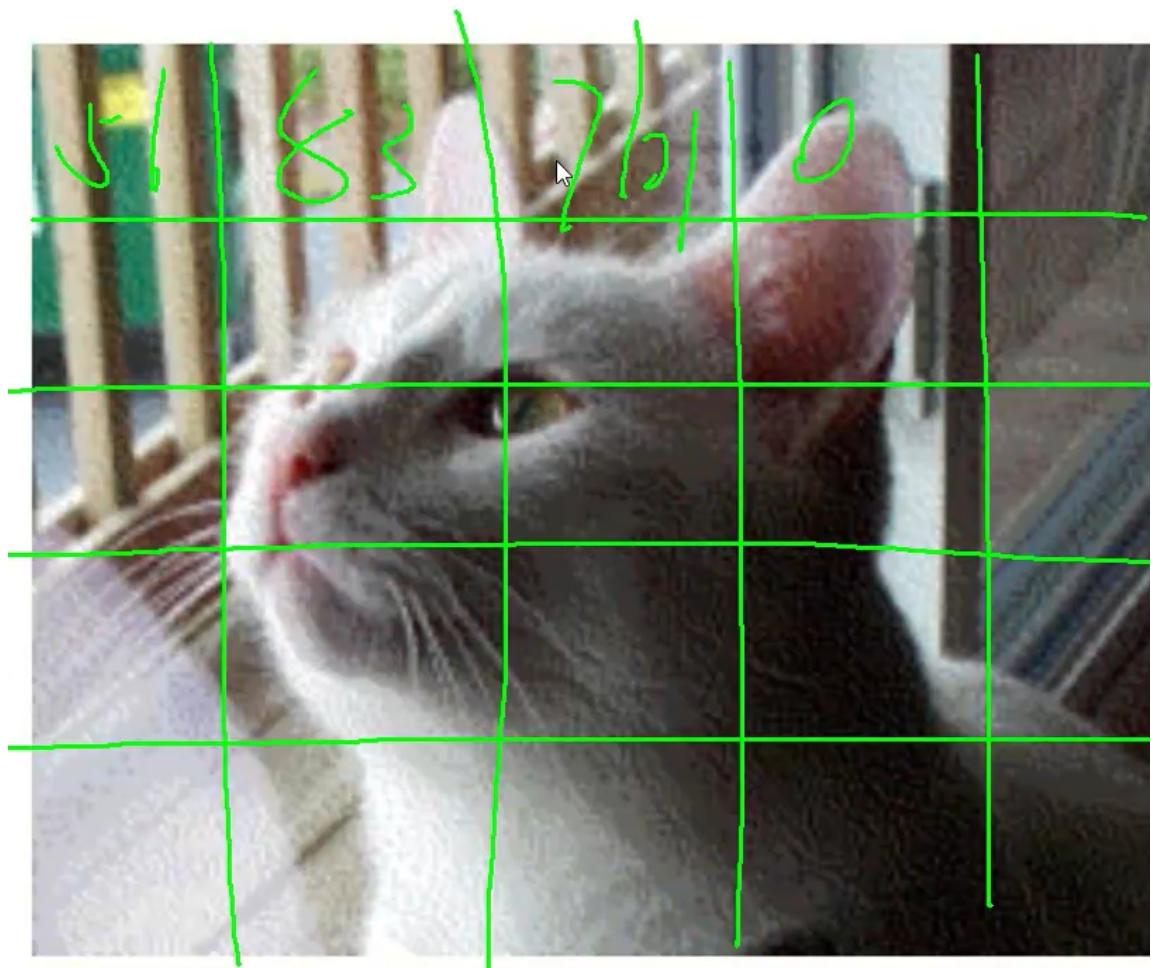


表: ——codebook

56: 绿色

80: 蓝色

761: 栅栏

图像特征提取流程

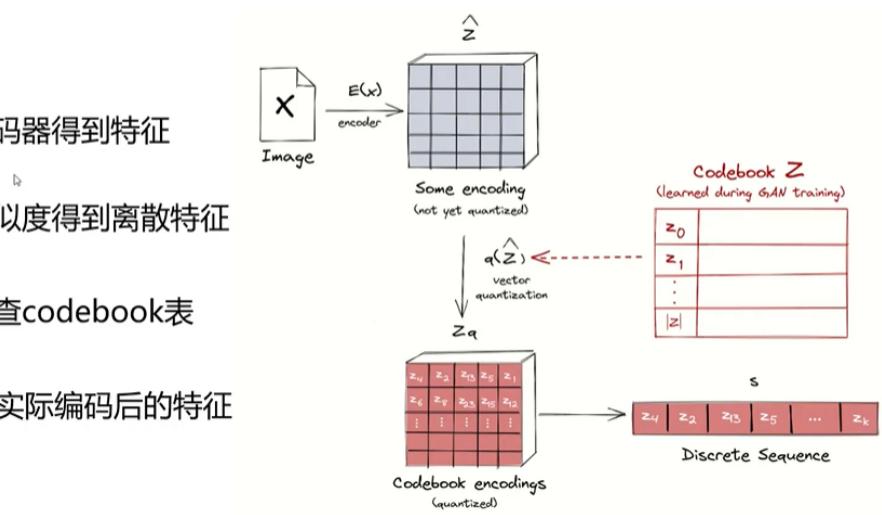
✓ 大致流程

📝 先通过编码器得到特征

📝 然后选相似度得到离散特征

📝 其实就是查codebook表

📝 最后得到实际编码后的特征

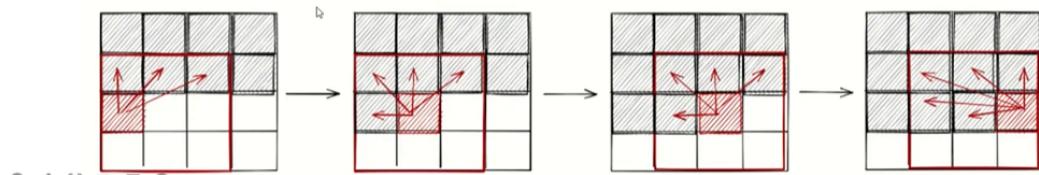


✓ 根据特征序列逐步生成

📎 类似transformer的感觉，一点点生成

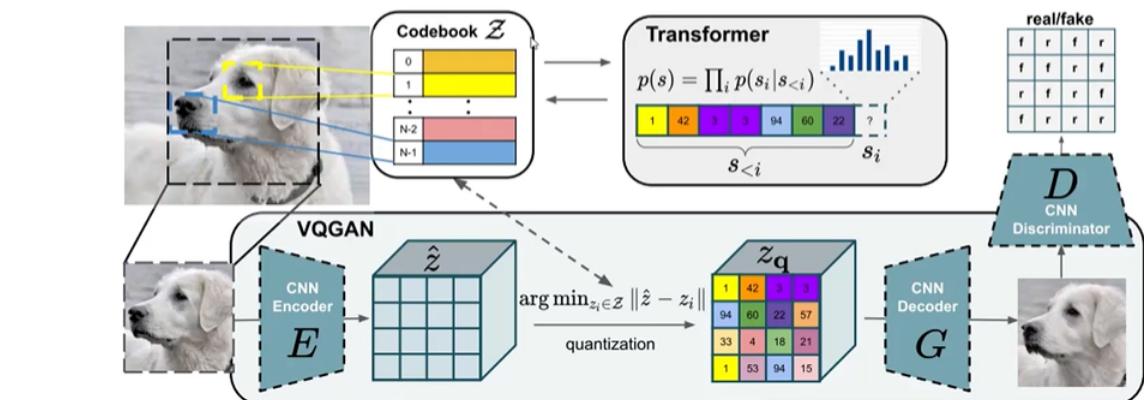
📎 生成的过程中还要它们之间的考虑关系

📎 但是并不是全局特征都会用到，论文貌似指的是邻居特征



✓ VQGAN相当于给DALL-E提供了基本出发点

📎 整理流程，别忘了还有transformer更新codebook



把decoder换成CLIP的文本图像配对

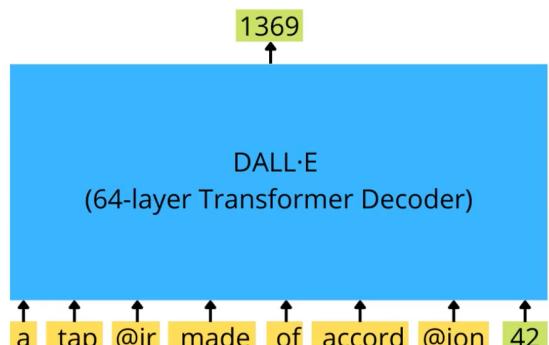
✓ DALL-E中如何生成结果呢

📎 类似GPT，只不过输入多了图像特征

📎 3种注意力（还有文本和图像的）

📎 42就是图像特征编码，预测输出图像

📎 例如输出的图像编码是1369(万物GPT化)



视频分类： ActionCLIP

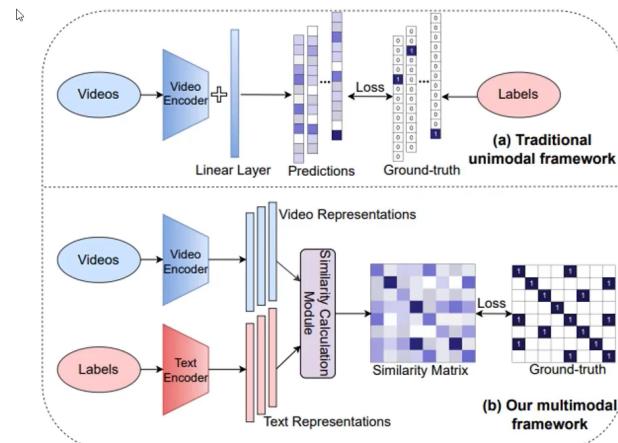
✓ ActionCLIP: A New Paradigm for Video Action Recognition

💡 视频分类，行为识别也类似

💡 其其实本质也是构建特征提取器

📝 同样是zero-shot来预测

 那感觉CV又可以通吃了



事件动作识别：CLIP-Event

✓ CLIP-Event: Connecting Text and Images with Event Structures

这个还挺有意思的，能将事件中的人与动作链接起来

💡 相当于先通过文本时间抽取得一些关系组合，再与图像进行配对

