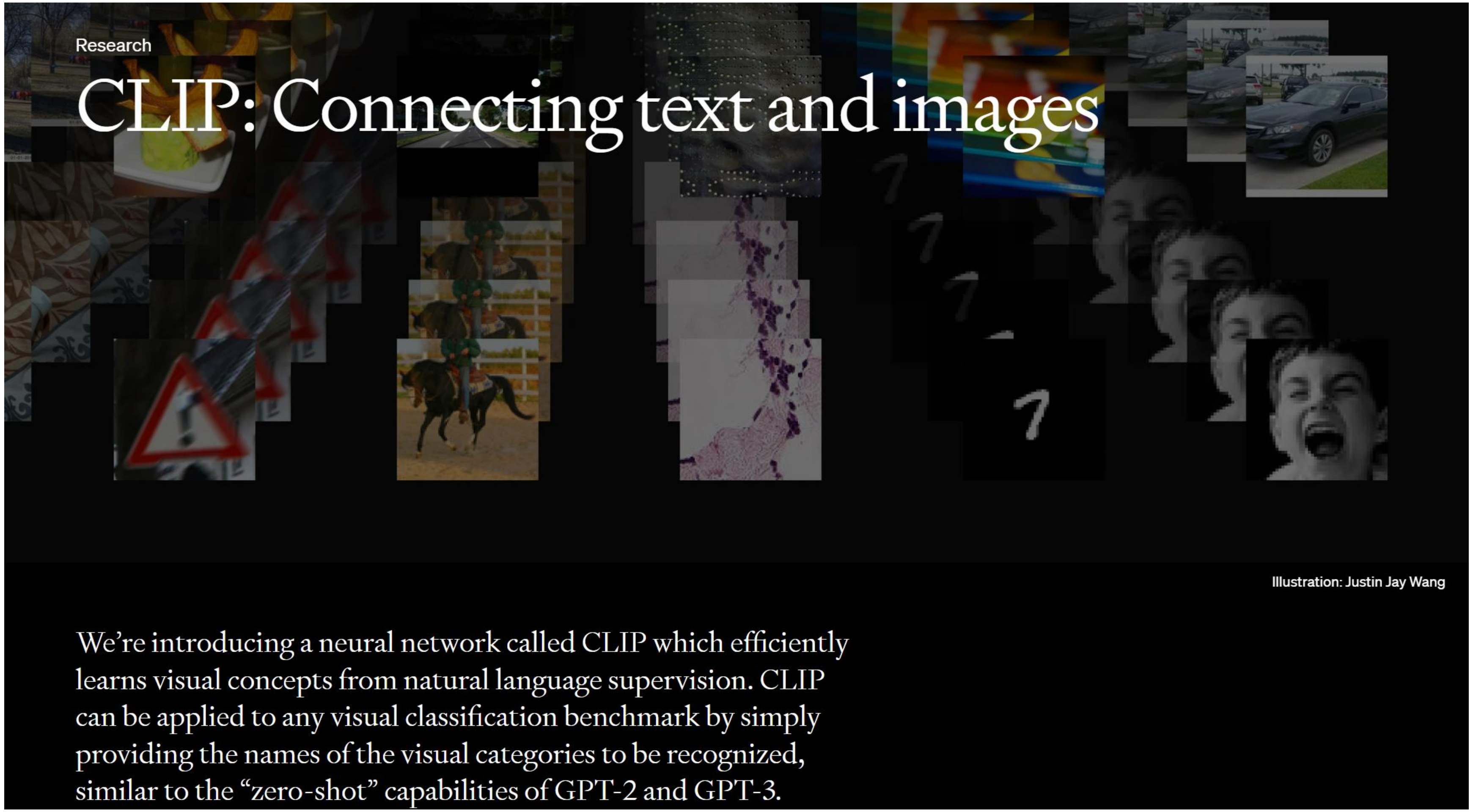
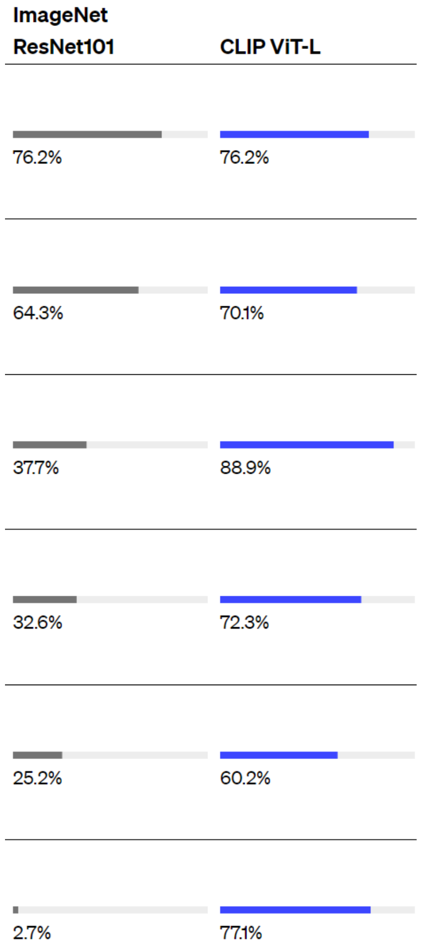
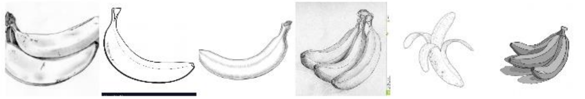
CLIP: Learning Transferable Visual Models From Natural Language Supervision

- Perform a great variety of classification benchmarks(zero-shot)
- Matching the performance of the original ResNet-50 on ImageNet zero-shot

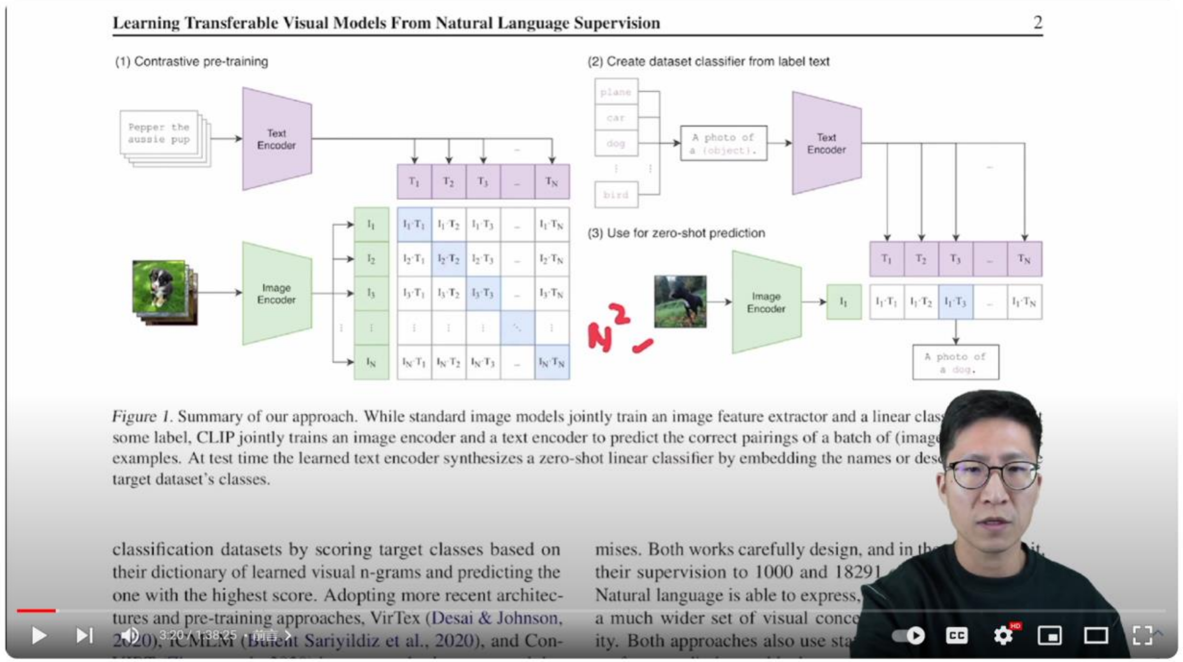




UNIVERSITY OF South Carolina

Dataset		ImageNet ResNet101	CLIP ViT-L
ImageNet		76.2%	76.2%
ImageNet V2		64.3%	70.1%
ImageNet Rendition		37.7%	88.9%
ObjectNet		32.6%	72.3%
ImageNet Sketch		25.2%	60.2%
ImageNet Adversarial		2.7%	77.1%

https://openai.com/research/clip



CLIP 论文逐段精读【论文精读】

Mu Li
73.9K subscribers

Subscribed · 631 · Share · Download · Clip · Save

Lil'Log

Posts Archive Search Tags

Diffusion Models for Video Generation

Date: April 12, 2024 | Estimated Reading Time: 20 min | Author: Lilian Weng

Lilian Weng

Lilianweng.github.io

Self-Supervised Representation Learning

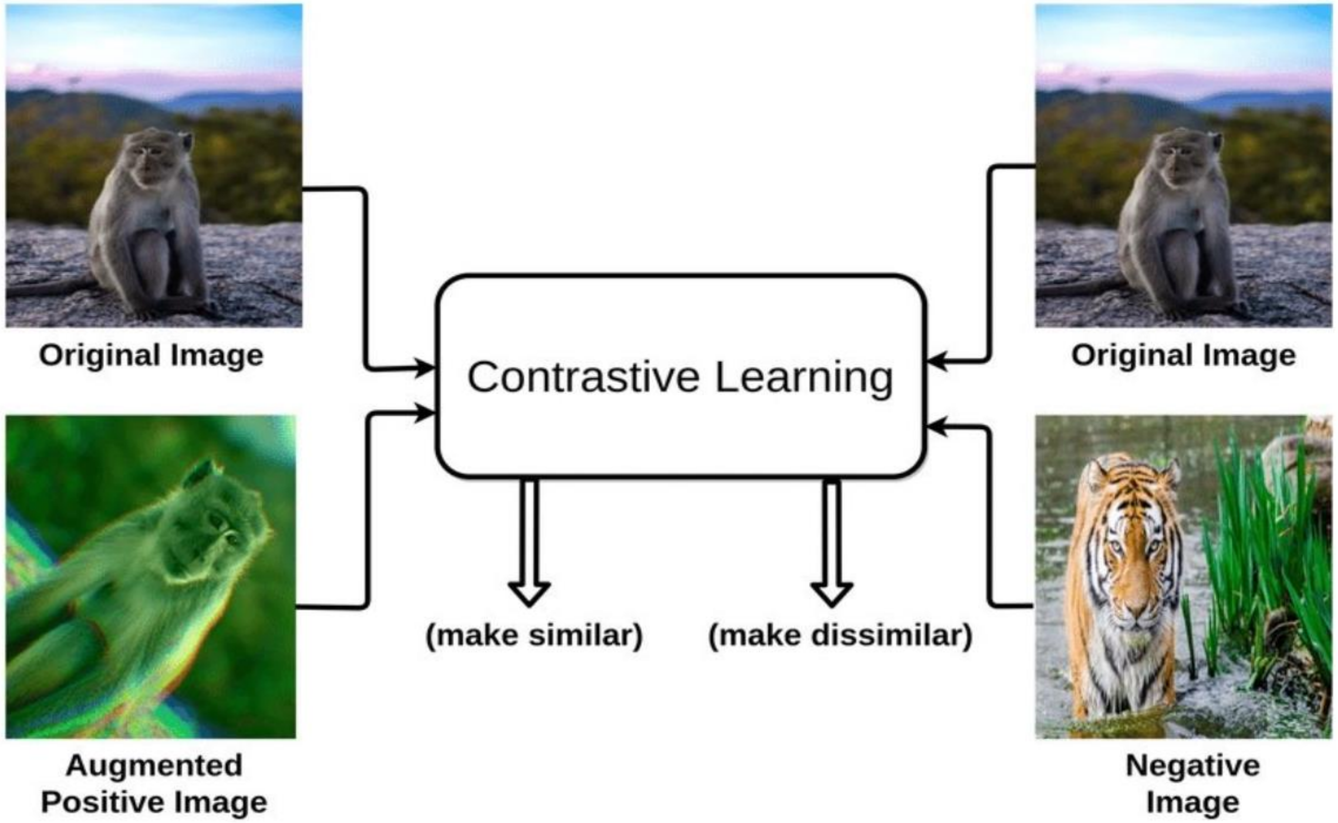
Date: November 10, 2019 | Estimated Reading Time: 38 min | Author: Lilian Weng

▶ Table of Contents

UNIVERSITY OF South Carolina

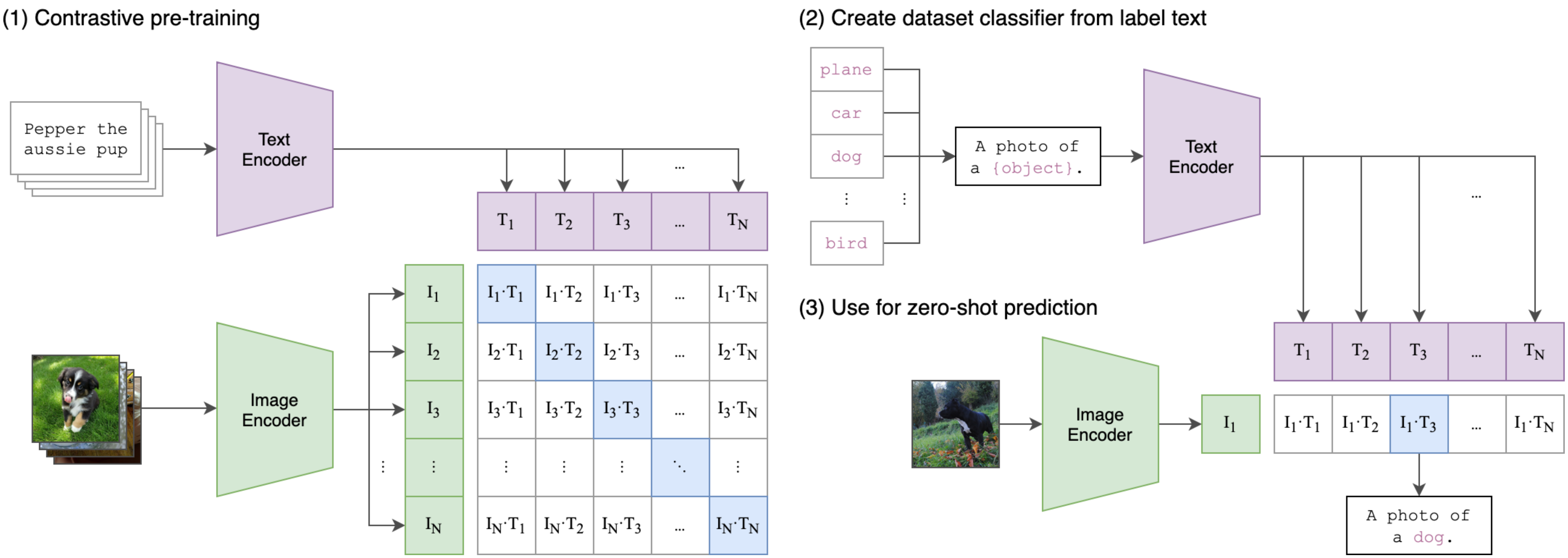
1. CONTRAST LEARNING

- CLIP (**Contrastive Language–Image Pre-training**) is a model developed by OpenAI that learns visual concepts from natural language descriptions.
- What is contrast learning?



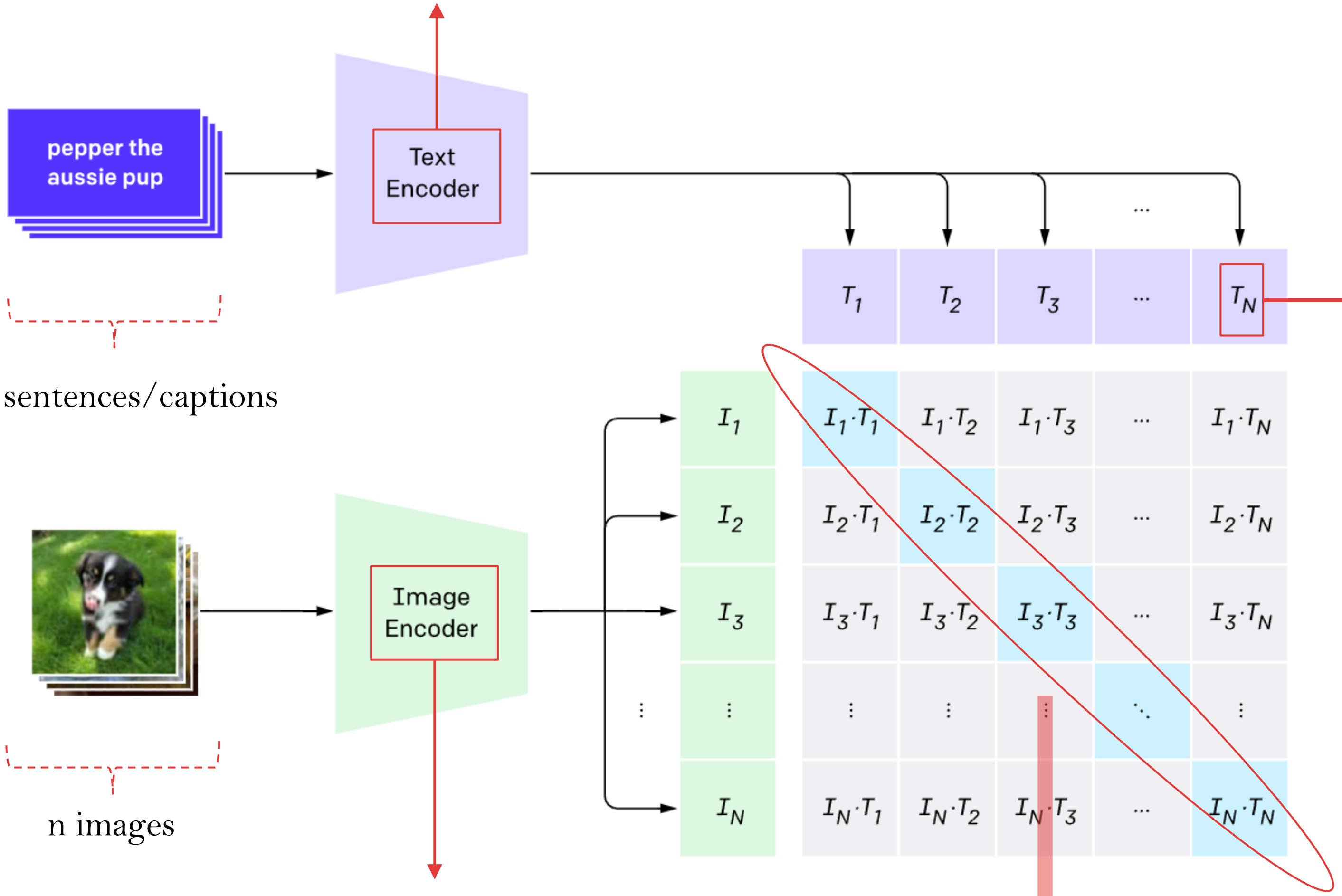
1. **Anchor Sample**: This is your main item or example.
2. **Positive Sample**: This is something that is similar to the anchor.
3. **Negative Sample**: This is something that is different from the anchor.

The goal is for the computer to learn that the anchor and the positive sample are similar, so it should treat them as being close together. On the other hand, it should learn that the negative sample is different from the anchor, so it should treat them as being far apart.



OpenAI convert all of a dataset's classes into captions such as "a photo of a dog" and predict the class of the caption CLIP estimates best pairs with a given image.

Encoder Model: transformer



n sentences/captions

n images

Encoder Model: ResNet/ViT(vision transformer)

Positive Sample

Training Batch: N
N image-text pairs

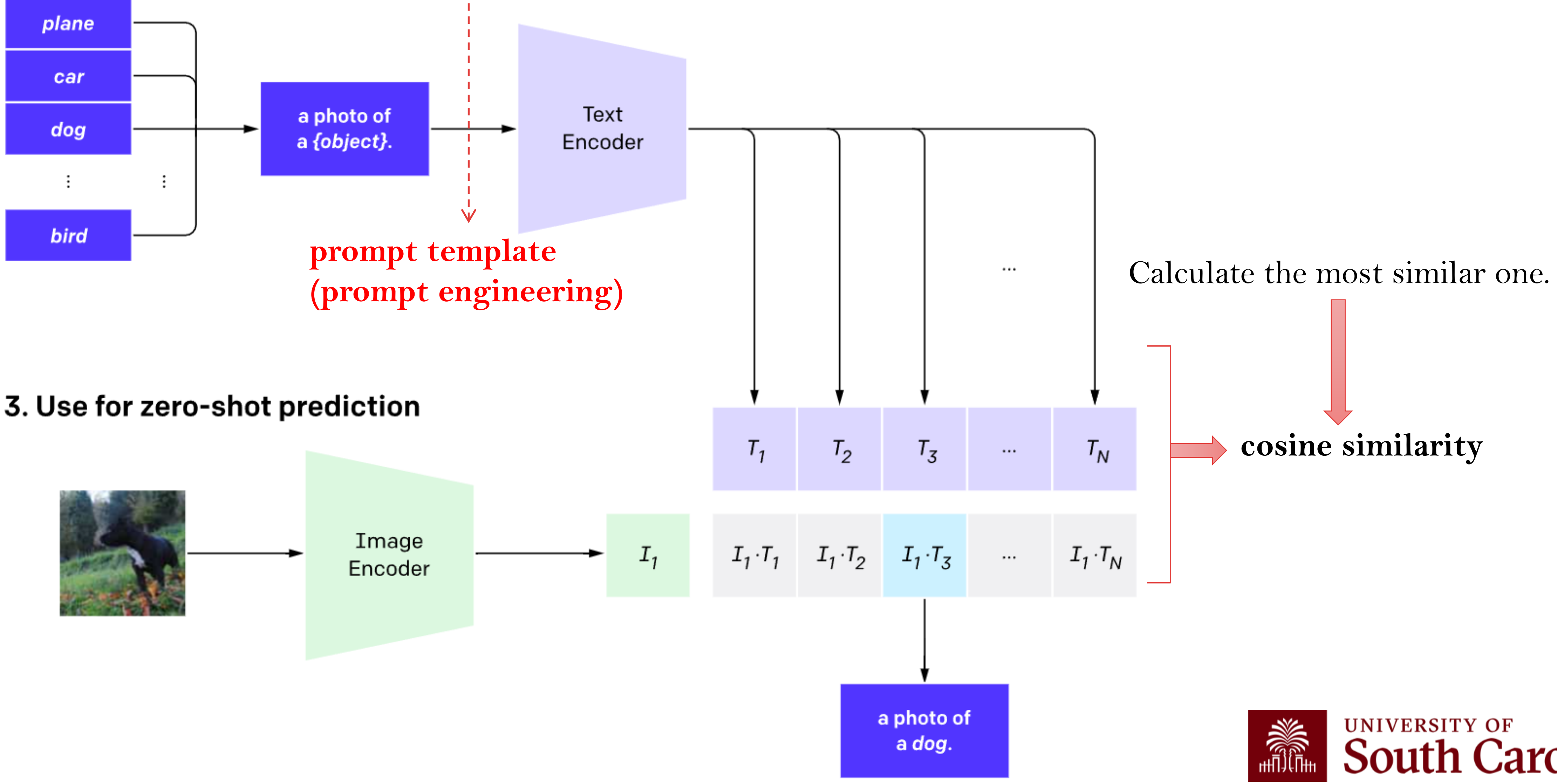
positive sample: I1-T1; I2-T2; … (n)
negative sample: others (n^2 - n)

unsupervised learning, without manual labeling

UNIVERSITY OF South Carolina

2. Create dataset classifier from label text

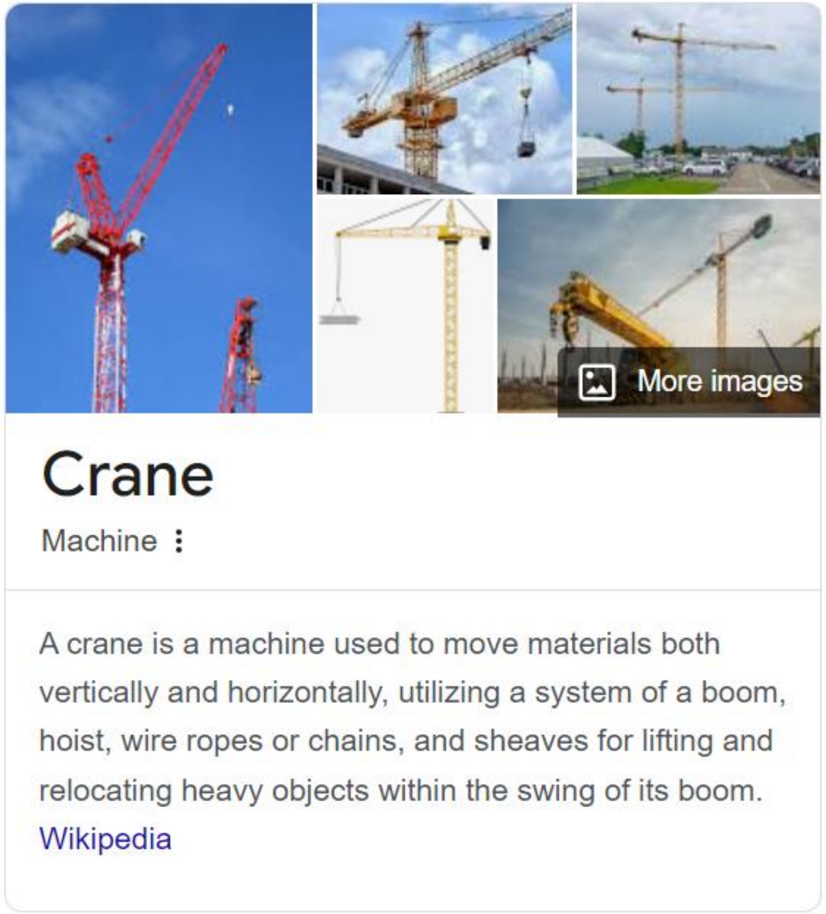
- how to do zero-shot without class label and classifier?



UNIVERSITY OF South Carolina

PROMPT ENGINEERING

polysemy: 多义性 → 歧义性出现，相似度的计算可能是错误的



Remote Sensing

遥远？
anything else?

如果提前知道一些信息，对zero-shot的推理是很有帮助的。(e.g. animal)

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}.

2. CLIP ARCHITECTURE

- Uses a **dual-encoder architecture** to map images and text into a shared latent space.

It works by jointly training two encoders. One encoder for images (**Vision Transformer**) and one for text (**Transformer-based language model**).

- 用文本的监督信号去训练一个视觉模型

a) 为什么用这种方法？---- (1) 不需要再去标注图像数据，只需要进行图像—文本的配对，数据的规模很容易扩大。(2) 因为训练的监督信号是text，不是一个个的label，Model输入与输出的自由度会大很多 (对比ImageNet)。(3) 因为是图像—文本配对的训练，模型学习到的会是一个多模态的特征。(DALL.E)

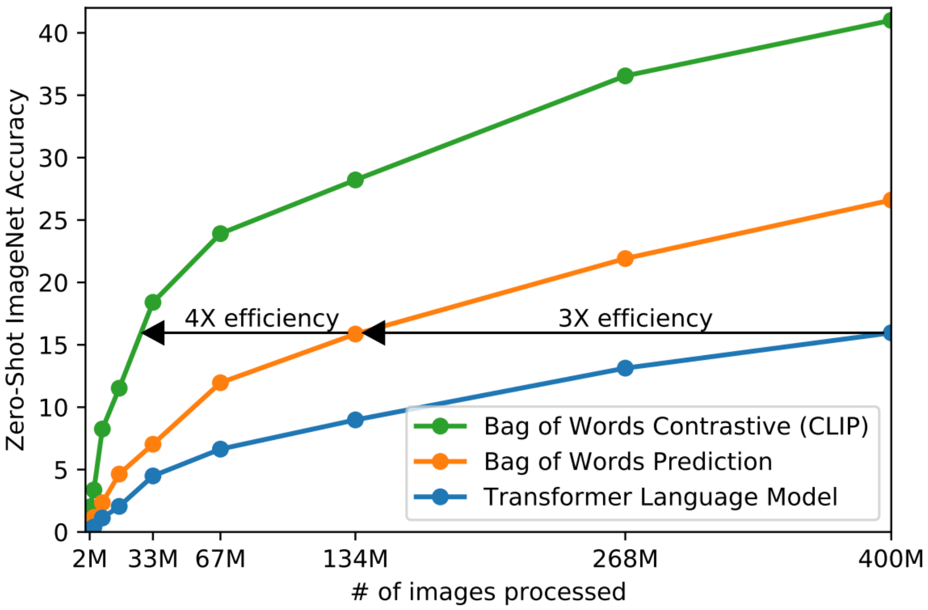
b) Dataset？---- 四个亿的图像—文本配对(*Feb 2021*)。Vision: Google JFT 300 Million; NLP: WebText (GPT-2)；够大够用

- 为什么使用对比学习？-- 效率

"Xie et al.(2020) required 33 TPUv3 core-years to train their Noisy Student EffcientNet-L2."

a) initial approach: 去预测一个图片对应的文本。预测的方式：逐字逐句去predicate。(movement特征描述/object特征描述)

b) 对比学习:训练任务是一个对比任务，**判断这个图片和这个文本是不是一个配对**。更合理/简单/高效(比预测型目标函数快4倍, GPT系列模型)



```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)
```

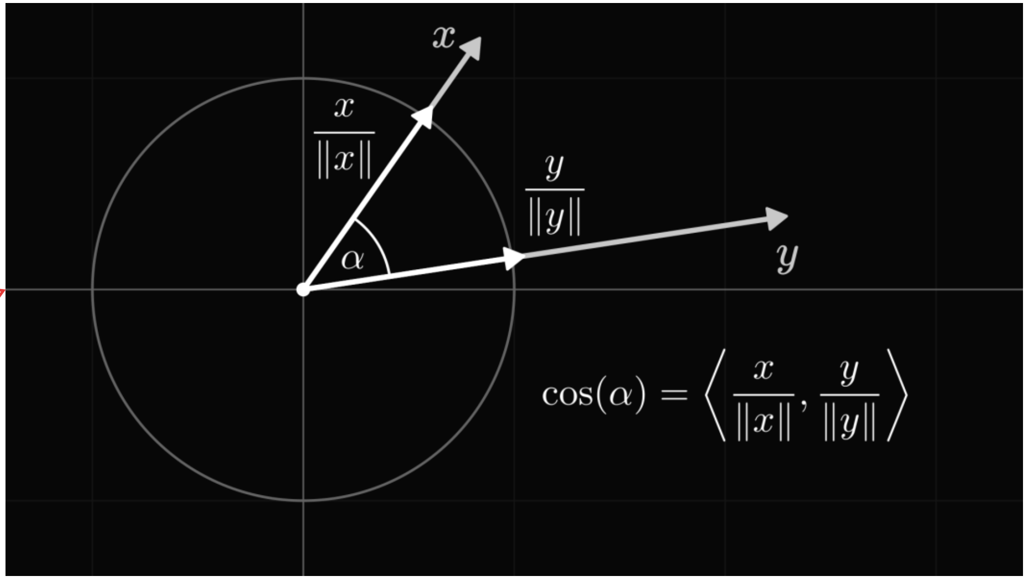
raw output of the last neural network

```
# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)
```

Index

```
# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

Figure 3. Numpy-like pseudocode for the core of an implementation of CLIP.



$$\mathrm{loss}_i = -\frac{1}{n}\sum_{j=1}^{n} \log\left(\frac{\exp(\mathrm{similarity}(I_j, T_j)/\tau)}{\sum_{k=1}^{n}\exp(\mathrm{similarity}(I_j, T_k)/\tau)}\right)$$ image-to-text

$$\mathrm{loss}_t = -\frac{1}{n}\sum_{j=1}^{n} \log\left(\frac{\exp(\mathrm{similarity}(T_j, I_j)/\tau)}{\sum_{k=1}^{n}\exp(\mathrm{similarity}(T_j, I_k)/\tau)}\right)$$ text-to-image

- I_j and T_j are the image and text embeddings, respectively, for the j-th true pair.
- similarity() is the function calculating the cosine similarity between two embeddings.
- τ is the temperature parameter scaling the logits before the softmax.

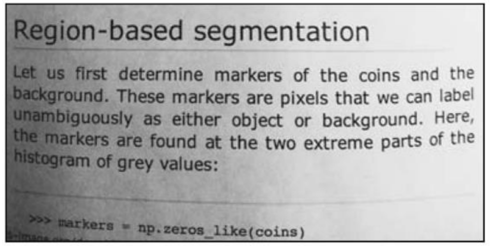
确保图像编码器和文本编码器在学习时能均衡地将嵌入信息投射到共享空间

3. OpenCLIP

- OpenCLIP is an open source implementation of OpenAI's CLIP (Contrastive Language-Image Pre-training). OpenAI没有开源训练的数据，也没有开源训练的代码。

Model	Training data	Resolution	# of samples seen	ImageNet zero-shot acc.
ConvNext-Base	LAION-2B	256px	13B	71.5%
ConvNext-Large	LAION-2B	320px	29B	76.9%
ConvNext-XXLarge	LAION-2B	256px	34B	79.5%
ViT-B/32	DataComp-1B	256px	34B	72.8%
ViT-B/16	DataComp-1B	224px	13B	73.5%
ViT-L/14	LAION-2B	224px	32B	75.3%
ViT-H/14	LAION-2B	224px	32B	78.0%
ViT-L/14	DataComp-1B	224px	13B	79.2%
ViT-G/14	LAION-2B	224px	34B	80.1%
ViT-L/14	OpenAI's WIT	224px	13B	75.5%

page.png
a page of text about segmentation


Region-based segmentation

Let us first determine markers of the coins and the background. These markers are pixels that we can label unambiguously as either object or background. Here, the markers are found at the two extreme parts of the histogram of grey values:

```
>>> markers = np.zeros_like(coins)
```

rocket.jpg
a rocket standing on a launchpad

UNIVERSITY OF South Carolina

Image Preprocessing

- 调整输入图像的大小并居中裁剪，使其符合模型所期望的图像分辨率
- 在此之前，使用数据集的Mean和Standard Deviation对像素强度进行归一化处理。

```
Compose(
    Resize(size=256, interpolation=bicubic, max_size=None, antialias=warn)
    CenterCrop(size=(256, 256))
    <function _convert_to_rgb at 0x7fac58819c10>
    ToTensor()
    Normalize(mean=(0.48145466, 0.4578275, 0.40821073), std=(0.26862954, 0.26130258, 0.27577711))
)
```

Text Preprocessing

- We use a case-insensitive tokenizer, which can be invoked using `tokenizer.tokenize()`. By default, the outputs are padded to become 77 tokens long, which is what the CLIP models expects.

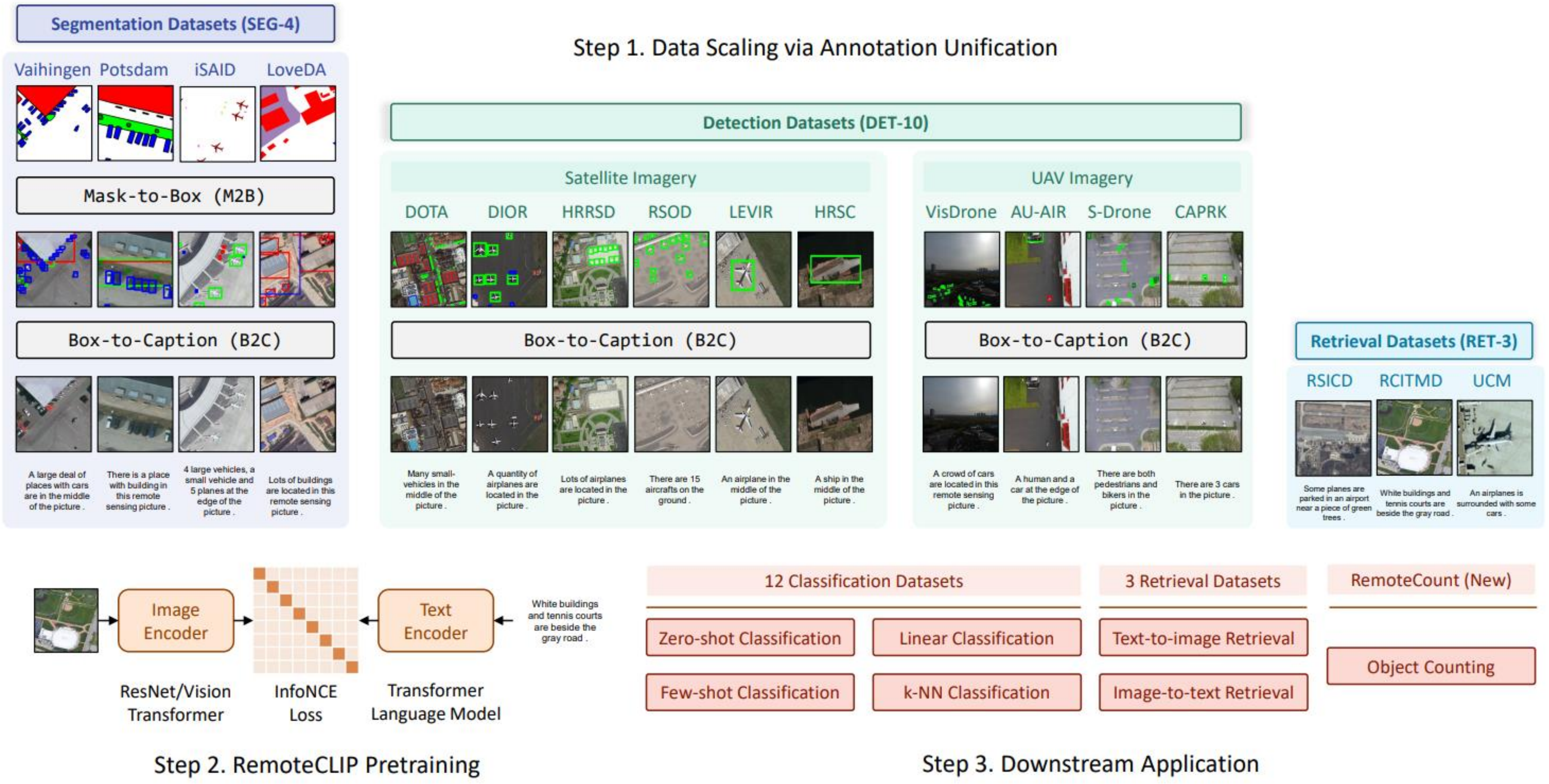
```
tokenizer.tokenize("Hello World!")
```

```
tensor([[49406,  3306,  1002,   256, 49407,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0]])
```

- framework: PyTorch(pytorch.org)
- GPUs: Google Colab(colab.research.google.com) or Lightning.AI (lightning.ai)
- Finetuning Reference: A beginner's guide to fine-tuning the CLIP model

4. RemoteCLIP

- RemoteCLIP的模型架构直接继承了CLIP，论文的贡献主要在于数据集的构建、基础模型的训练评测与开源。



UNIVERSITY OF South Carolina

Datasets: Zilun / RS5M

like 9

Dataset card | Viewer | Files | Community

Split (2)

train

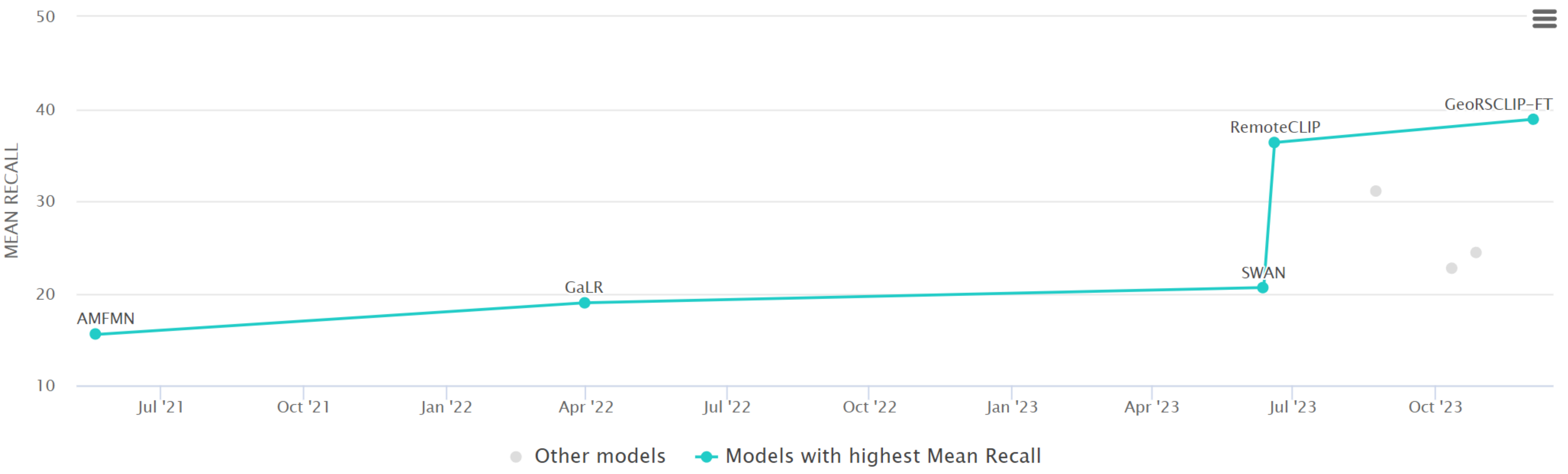
▶ The full dataset viewer is not available (click to read why). Only showing a preview of the rows.

img_name string	caption string
laion2b_0_0.jpg	Aerial photography Pattern on the Earth Field Corn Farm Abstract Harvest Season
laion2b_0_2.jpg	San AntonioTexas suburban housing development neighborhood - aerial view stock photo
laion2b_0_4.jpg	Aerial view of historical orthodox monasteries on the top of meteors cliffs
laion2b_0_5.jpg	Overhead view of a car parking entrance road. Aerial view....
laion2b_0_7.jpg	Aerial view of Albert Park and the Melbourne skyline, Australia
laion2b_0_9.jpg	Aerial photo taken on Oct. 6, 2019 shows tourists viewing pink muhly grass in the Fenghuanggou scenic area during the National Day holiday in Nanchang, capital of east China's Jiangxi…
laion2b_0_10.jpg	Aerial view of the City, London
laion2b_0_12.jpg	Vancouver - panoramic aerial view with downtown, Kitsilano beach and Coast Mountains - stock photo
laion2b_0_13.jpg	Aerial view of Cardiff

Cross-Modal Retrieval on RSICD

Leaderboard Dataset

View Mean Recall by Date for All models



Now, you can initialize a CLIP model with `OpenCLIP`, then load the RemoteCLIP checkpoint with a few lines of code:

```
import torch, open_clip
from PIL import Image

model_name = 'ViT-L-14' # 'RN50' or 'ViT-B-32' or 'ViT-L-14'
model, _, preprocess = open_clip.create_model_and_transforms(model_name)
tokenizer = open_clip.get_tokenizer(model_name)

ckpt = torch.load(f"path/to/your/checkpoints/RemoteCLIP-{model_name}.pt", map_location="cpu")
message = model.load_state_dict(ckpt)
print(message)

model = model.cuda().eval()
```

RemoteCLIP

Both of them based on OpenCLIP fine-tuning framework

- (Optional) If you just want to load the GeoRSCLIP model:

```
import open_clip
import torch
from inference_tool import get_preprocess


ckpt_path = "/your/local/path/to/RS5M_ViT-B-32.pt"
model, _, _ = open_clip.create_model_and_transforms("ViT-B/32", pretrained="openai")
checkpoint = torch.load(ckpt_path, map_location="cpu")
msg = model.load_state_dict(checkpoint, strict=False)
model = model.to("cuda")
img_preprocess = get_preprocess(
        image_resolution=224,
)
```

GeoRSCLIP

UNIVERSITY OF South Carolina

1. **Prepare Dataset and Environment**
2. **Defining the data loader**
3. **Detecting correct data input**
4. **Fine-tuning Model**
5. **Evaluate Model**

Pretrained model: RemoteCLIP

Vision Encoder: ViT/ResNet

Prepare Dataset and Environment

In [ ]:
```
!pip install huggingface_hub open_clip_torch
!git clone https://github.com/ChenDelong1999/RemoteCLIP/
```

In [2]:
```
# Load dataset
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
# @title Load packages and download model weights
from huggingface_hub import hf_hub_download
import torch, open_clip
from PIL import Image
from IPython.display import display

# for model_name in ['RN50'] #, 'ViT-B-32', 'ViT-L-14']: #faster loading
for model_name in ['RN50', 'ViT-B-32', 'ViT-L-14']: #all models
    checkpoint_path = hf_hub_download("chendelong/RemoteCLIP", f"RemoteCLIP-{model_name}.pt", cache_dir='checkpoints')
    print(f'{model_name} is downloaded to {checkpoint_path}.')
```

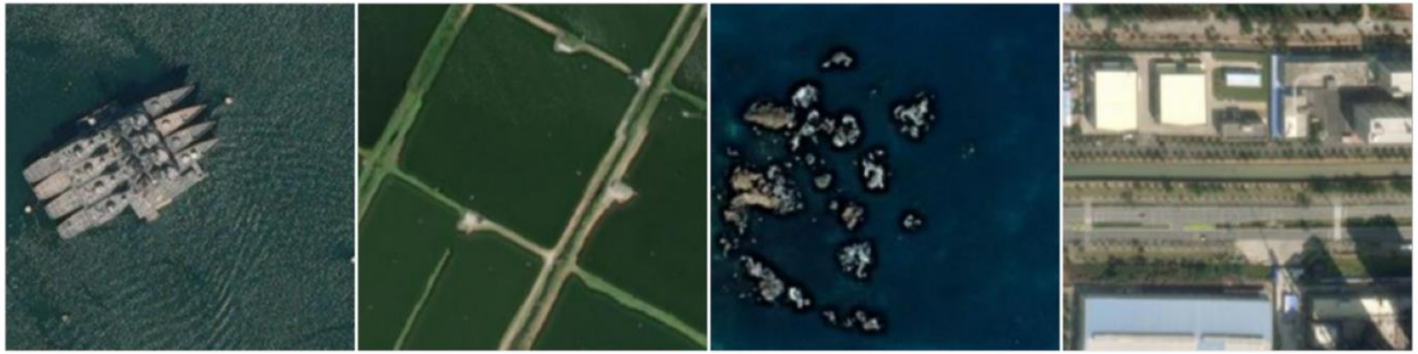
In [4]:
```
# @title Select Model
model_name = 'RN50' # @param ['RN50', 'ViT-B-32', 'ViT-L-14']
model, _, preprocess = open_clip.create_model_and_transforms(model_name)
tokenizer = open_clip.get_tokenizer(model_name)

path_to_your_checkpoints = 'checkpoints/models--chendelong--RemoteCLIP/snapshots/bf1d8a3ccf2ddbf7c875705e46373bfe542bce38'

ckpt = torch.load(f"{path_to_your_checkpoints}/RemoteCLIP-{model_name}.pt", map_location="cpu")
message = model.load_state_dict(ckpt)
print(message)
model = model.cuda().eval()
```

```
<All keys matched successfully>
```

Defining Data Loader

```python
import os
import pandas as pd
from PIL import Image
import torch
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader

class ImageTextDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None):
        # self.img_labels = pd.read_csv(annotations_file)
        self.img_labels = pd.read_csv(annotations_file)

        self.img_dir = img_dir
        self.transform = transform

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

    def __getitem__(self, idx):
      img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
      image = Image.open(img_path)
      image = image.convert('RGB')  # Convert to RGB
      caption = self.img_labels.iloc[idx, 1]
      if self.transform:
          image = self.transform(image)
      return image, caption
```

```
An satellite view of a container, a grouping tool designed for transporting packaged or unpackaged goods, facilitating loadin
g and unloading with mechanical equipment, in transportation settings.
An satellite view of a merchantman, primarily intended for commercial purposes, carrying passengers, mail, and cargo, in navi
gational waters.
An satellite view of aquaculture devices anchored with ropes, featuring black speckled or black net-like patterns across the
surface, in oceanic environments.
An satellite view of cultivated land, featuring surfaces covered with crops and outlined in rectangular plots, in agricultura
l areas.
```



```python
# data transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images to the expected input size of the model
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),  # Normalize images
])


# creat dataset and dataloader
dataset = ImageTextDataset(annotations_file='/content/drive/MyDrive/my_clip/sea_clip_dataset/caption.csv',
                           img_dir='/content/drive/MyDrive/my_clip/sea_clip_dataset/images',
                           transform=transform)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

UNIVERSITY OF South Carolina

Fine-tuning the model

Once the data loader has been defined, fine-tuning the model can begin. This involves iterating the data loader, feeding each batch of images and text into the model, calculating the loss, and updating the model's weights. The following is a simplified example of the fine-tuning process.

```
from torch import nn, optim, from_numpy
import numpy as np
from open_clip import tokenize

# Load the model
device = "cuda" if torch.cuda.is_available() else "cpu"

# If the model isn't automatically moved to the correct device, explicitly do so
model = model.to(device)


# 假设已经定义了optimizer和loss function
optimizer = optim.Adam(model.parameters(), lr=5e-5)
criterion = nn.CrossEntropyLoss()

num_epochs = 100  # Example number of epochs

for epoch in range(num_epochs):
    running_loss = 0.0
    for i, (images, captions) in enumerate(dataloader):
        images = images.to("cuda")
        text_tokens = tokenize(captions).to("cuda")  # Ensure captions are properly proc

        # Zero the parameter gradients
        optimizer.zero_grad()
        # Temporarily capture the entire output
        output = model(images, text_tokens)

        image_features, text_features, _ = output

        # Example: Calculating a simple similarity-based loss
        similarity = torch.nn.functional.cosine_similarity(image_features, text_features, dim=1)

        # Compute loss
        # loss = criterion(image_features, text_features)  # Placeholder, adjust as necessary
        loss = 1 - similarity.mean()  # Aiming to maximize similarity
```

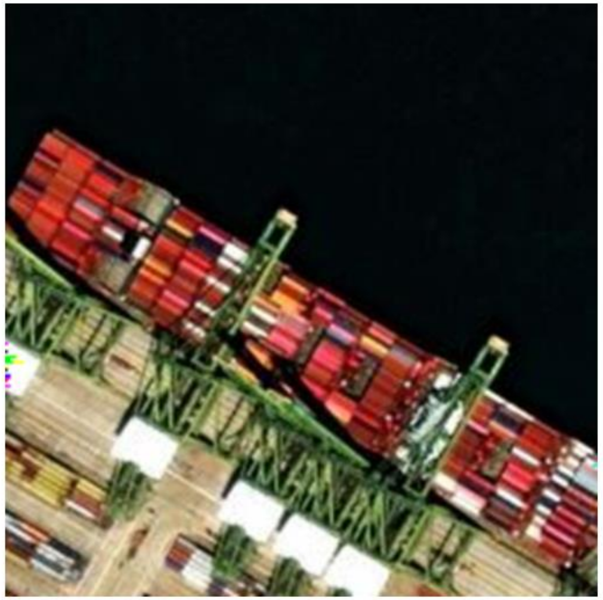
```
optimizer = optim.Adam(model.parameters(), lr=5e-5)
criterion = nn.CrossEntropyLoss()
```

Test model for zero-shot

```
import torch
import open_clip

# Define the model architecture (should be the same as used for training)
device = "cuda" if torch.cuda.is_available() else "cpu"
model_name = 'RN50' # @param ['RN50', 'ViT-B-32', 'ViT-L-14']
model, _, preprocess = open_clip.create_model_and_transforms(model_name)
tokenizer = open_clip.get_tokenizer(model_name)
# model, preprocess = clip.load("ViT-B/32", device=device)  # Assuming you used ViT-B/32 for training
# model, _, preprocess = open_clip.create_model_and_transforms('ViT-B-32', pretrained='laion2b_s34b_b79k')

# Load the state dictionary
model.load_state_dict(torch.load('/content/model_save/remoteclip_finetuning_state_dict.pth'))

# Move model to evaluation mode
model.eval()
```

Text caption queries

```
[13] # @title Text caption queries
text_queries = [
    "A busy airport with many aeroplanes.",
    "Satellite view of Wuhan university.",
    "An Aerial view of cargo ship near a sea port.",
    "An Aerial view of aircraft carrier",
    "A broken mirror in the water",
    "A little cat playing a ball",
]
text = tokenizer(text_queries)
image = Image.open("/content/img11.png").convert('RGB') #convert to rgb allows it to display as png if the jpg is in cmyk
display(image)
```



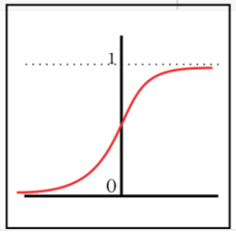
Predicted probabilities

```
# @title Predicted probabilities
image = preprocess(image).unsqueeze(0)

with torch.no_grad(), torch.cuda.amp.autocast():
    image_features = model.encode_image(image.cuda())
    text_features = model.encode_text(text.cuda())
    image_features /= image_features.norm(dim=-1, keepdim=True)
    text_features /= text_features.norm(dim=-1, keepdim=True)

    text_probs = (100.0 * image_features @ text_features.T).softmax(dim=-1).cpu().numpy()[0]

print(f'Predictions of {model_name}:')
for query, prob in zip(text_queries, text_probs):
    print(f"{query:<40} {prob * 100:5.1f}%")
```

```
Predictions of ViT-B-32:
A busy airport with many aeroplanes.        0.0%
Satellite view of Wuhan university.         0.0%
An Aerial view of cargo ship near a sea port.   54.6%
An Aerial view of aircraft carrier         38.1%
A broken mirror in the water                7.1%
A little cat playing a ball                 0.2%
```

Sigmoid / SoftMax

Input $X \in \mathbb{R} \longrightarrow$



$\longrightarrow$ **Output** $P(Y{=}k \mid X) \in [0,1]$