[1]

import subprocess

CUDA\_version = [s for s in subprocess.check\_output(["nvcc", "--version"]).decode("UTF- 8").split(", ") if s.startswith("release")][0].split(" ")[-1]

print("CUDA version:", CUDA\_version)

if CUDA\_version == "10.0": torch\_version\_suffix = "+cu100"

elif CUDA\_version == "10.1": torch\_version\_suffix = "+cu101"

elif CUDA\_version == "10.2": torch\_version\_suffix = ""

else:

torch\_version\_suffix = "+cu110"

! pip install torch==1.7.1{torch\_version\_suffix} torchvision==0.8.2{torch\_version\_suffix} - f https://download.pytorch.org/whl/torch\_stable.html ftfy regex

[2]

! pip install ftfy regex tqdm

! pip install git+https://github.com/openai/CLIP.git

[3]

import numpy as np import torch

from pkg\_resources import packaging print("Torch version:", torch. version )

[4]

import clip clip.available\_models()

[out]

['RN50',

'RN101',

'RN50x4', 'RN50x16', 'RN50x64', 'ViT-B/32',

'ViT-B/16',

'ViT-L/14',

'ViT-L/14@336px']

[5]

model, preprocess = clip.load("ViT-B/32") model.cuda().eval()

input\_resolution = model.visual.input\_resolution context\_length = model.context\_length vocab\_size = model.vocab\_size

print("Model parameters:", f"{np.sum([int(np.prod(p.shape)) for p in model.parameters()]):,

}")

print("Input resolution:", input\_resolution) print("Context length:", context\_length) print("Vocab size:", vocab\_size)

[6]

preprocess

[7]

clip.tokenize("Hello World!")

[8]

import os import skimage

import IPython.display

import matplotlib.pyplot as plt from PIL import Image import numpy as np

from collections import OrderedDict import torch

%matplotlib inline

%config InlineBackend.figure\_format = 'retina'

# images in skimage to use and their textual descriptions

descriptions = {

"astronaut": "a portrait of an astronaut with the American flag", "rocket": "a rocket standing on a launchpad", "motorcycle\_right": "a red motorcycle standing in a garage", "camera": "a person looking at a camera on a tripod",

"horse": "a black-and-white silhouette of a horse", "coffee": "a cup of coffee on a saucer"

}

[9]

original\_images = [] images = []

texts = [] plt.figure(figsize=(16, 5))

for filename in [filename for filename in os.listdir(skimage.data\_dir) if filename.endswith(". png") or filename.endswith(".jpg")]:

name = os.path.splitext(filename)[0] if name not in descriptions:

continue

image = Image.open(os.path.join(skimage.data\_dir, filename)).convert("RGB") plt.subplot(2, 4, len(images) + 1)

plt.imshow(image) plt.title(f"{filename}\n{descriptions[name]}") plt.xticks([])

plt.yticks([]) original\_images.append(image) images.append(preprocess(image))

texts.append(descriptions[name]) plt.tight\_layout()

[10]

image\_input = torch.tensor(np.stack(images)).cuda()

text\_tokens = clip.tokenize(["This is " + desc for desc in texts]).cuda()

[11]

with torch.no\_grad():

image\_features = model.encode\_image(image\_input).float() text\_features = model.encode\_text(text\_tokens).float()

[12]

image\_features /= image\_features.norm(dim=-1, keepdim=True) text\_features /= text\_features.norm(dim=-1, keepdim=True)

similarity = text\_features.cpu().numpy() @ image\_features.cpu().numpy().T

[13]

count = len(descriptions) plt.figure(figsize=(20, 14)) plt.imshow(similarity, vmin=0.1, vmax=0.3)

# plt.colorbar()

plt.yticks(range(count), texts, fontsize=18) plt.xticks([])

for i, image in enumerate(original\_images):

plt.imshow(image, extent=(i - 0.5, i + 0.5, -1.6, -0.6), origin="lower") for x in range(similarity.shape[1]):

for y in range(similarity.shape[0]):

plt.text(x, y, f"{similarity[y, x]:.2f}", ha="center", va="center", size=12) for side in ["left", "top", "right", "bottom"]: plt.gca().spines[side].set\_visible(False)

plt.xlim([-0.5, count - 0.5])

plt.ylim([count + 0.5, -2])

plt.title("Cosine similarity between text and image features", size=20)



[14]

from torchvision.datasets import CIFAR100

cifar100 = CIFAR100(os.path.expanduser("~/.cache"), transform=preprocess, download=Tr ue)

[15]

text\_descriptions = [f"This is a photo of a {label}" for label in cifar100.classes] text\_tokens = clip.tokenize(text\_descriptions).cuda()

[16]

with torch.no\_grad():

text\_features = model.encode\_text(text\_tokens).float() text\_features /= text\_features.norm(dim=-1, keepdim=True)

text\_probs = (100.0 \* image\_features @ text\_features.T).softmax(dim=-1) top\_probs, top\_labels = text\_probs.cpu().topk(5, dim=-1)

[17]

plt.figure(figsize=(16, 16))

for i, image in enumerate(original\_images):

plt.subplot(4, 4, 2 \* i + 1) plt.imshow(image) plt.axis("off") plt.subplot(4, 4, 2 \* i + 2)

y = np.arange(top\_probs.shape[-1]) plt.grid()

plt.barh(y, top\_probs[i]) plt.gca().invert\_yaxis() plt.gca().set\_axisbelow(True)

plt.yticks(y, [cifar100.classes[index] for index in top\_labels[i].numpy()]) plt.xlabel("probability")

plt.subplots\_adjust(wspace=0.5) plt.show()