Colab FAQ

For some basic overview and features offered in Colab notebooks, check out: Overview of Colaboratory Features

You need to use the colab GPU for this assignment by selecting:

> **Runtime → Change runtime type → Hardware Accelerator: GPU**

Part 4: Connecting Text and Images with CLIP

Acknowledgement: This notebook is based on the code from https://colab.research.google.com/github/openai/clip/blob/master/notebooks/Interacting_with_CLIP.ipynb. Credit to OpenAI.

Section I: Interacting with CLIP

This is a self-contained notebook that shows how to download and run CLIP models, calculate the similarity between arbitrary image and text inputs, and perform zero-shot image classifications. The next cells will install the clip package and its dependencies, and check if PyTorch 1.7.1 or later is installed.

In [1]:

```
! pip install ftfy regex tqdm
! pip install git+https://github.com/openai/CLIP.git
```

```
Collecting ftfy
  Downloading ftfy-6.1.1-py3-none-any.whl (53 kB)
     |████████████████████████████████| 53 kB 996 kB/s
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (2019.12.20)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (4.63.0)
Requirement already satisfied: wcwidth>=0.2.5 in /usr/local/lib/python3.7/dist-packages (from ftfy) (0.2.5)
Installing collected packages: ftfy
Successfully installed ftfy-6.1.1
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-ri9zw6nw
  Running command git clone -q https://github.com/openai/CLIP.git /tmp/pip-req-build-ri9zw6nw
Requirement already satisfied: ftfy in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (6.1.1)
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (2019.12.20)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (4.63.0)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (1.10.0+cu111)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (0.11.1+cu111)
Requirement already satisfied: wcwidth>=0.2.5 in /usr/local/lib/python3.7/dist-packages (from ftfy->clip==1.0) (0.2.5)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch->clip==1.0) (3.10.0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision->clip==1.0) (1.21.5)
Requirement already satisfied: pillow!=8.3.0,>=5.3.0 in /usr/local/lib/python3.7/dist-packages (from torchvision->clip==1.0) (7.1.2)
Building wheels for collected packages: clip
  Building wheel for clip (setup.py) ... done
  Created wheel for clip: filename=clip-1.0-py3-none-any.whl size=1369221 sha256=a03df40b14e04b8d01a72bb6ec53d12a4c9fbb1d4a2b39bda95697569c318e9e
  Stored in directory: /tmp/pip-ephem-wheel-cache-5bqsmcp3/wheels/fd/b9/c3/5b4470e35ed76e174bff77c92f91da82098d5e35fd5bc8cdac
Successfully built clip
Installing collected packages: clip
Successfully installed clip-1.0
```

In [2]:

```
import numpy as np
import torch

print("Torch version:", torch.__version__)
torch_version = torch.__version__.split(".")
assert (int(torch_version[0]) == 1 and int(torch_version[1]) >=7) or int(torch_version[0]) > 1, "PyTorch 1.7.1 or later is required"
```

```
Torch version: 1.10.0+cu111
```

Loading the model

`clip.available_models()` will list the names of available CLIP models.

In [3]:

```
import clip

clip.available_models()
```

Out[3]:

```
['RN50',
 'RN101',
 'RN50x4',
 'RN50x16',
 'RN50x64',
 'ViT-B/32',
 'ViT-B/16',
 'ViT-L/14']
```

In [4]:

```
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)
```

```
100%|████████████████████████████████████████| 338M/338M [00:05<00:00, 59.9MiB/s]
Model parameters: 151,277,313
Input resolution: 224
Context length: 77
Vocab size: 49408
```

Image Preprocessing

We resize the input images and center-crop them to conform with the image resolution that the model expects. Before doing so, we will normalize the pixel intensity using the dataset mean and standard deviation.

The second return value from `clip.load()` contains a torchvision `Transform` that performs this preprocessing.

In [5]:

```
preprocess
```

Out[5]:

```
Compose(
    Resize(size=224, interpolation=bicubic, max_size=None, antialias=None)
    CenterCrop(size=(224, 224))
    <function _convert_image_to_rgb at 0x7ff83e82eb90>
    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 `clip.tokenize()`. By default, the outputs are padded to become 77 tokens long, which is what the CLIP models expects.

In [6]:

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

Out[6]:

```
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]])
```

Setting up input images and texts

We are going to feed 8 example images and their textual descriptions to the model, and compare the similarity between the corresponding features.

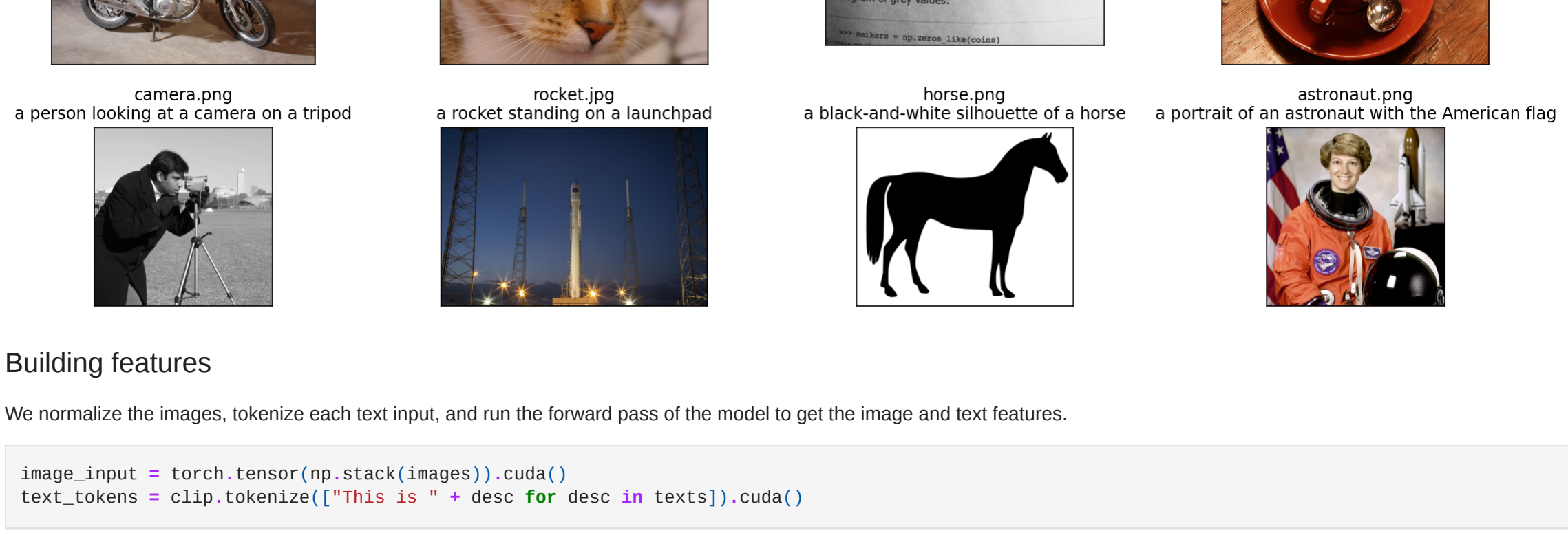
The tokenizer is case-insensitive, and we can freely give any suitable textual descriptions.

In [7]:

```
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 = {
    "page": "a page of text about segmentation",
    "chelsea": "a facial photo of a tabby cat",
    "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"
}
```

In [8]:

```
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")
    #print(image.__dict__.keys())
    #print(image._size)
    #image_sequence = image.getdata()
    #image_array = np.array(image_sequence)
    #print(image_array.shape)

    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()
```



Building features

We normalize the images, tokenize each text input, and run the forward pass of the model to get the image and text features.

In [9]:

```
image_input = torch.tensor(np.stack(images)).cuda()
text_tokens = clip.tokenize(["This is " + desc for desc in texts]).cuda()
```

In [10]:

```
with torch.no_grad():
    image_features = model.encode_image(image_input).float()
    text_features = model.encode_text(text_tokens).float()
```

Calculating cosine similarity

We normalize the features and calculate the dot product of each pair.

In [11]:

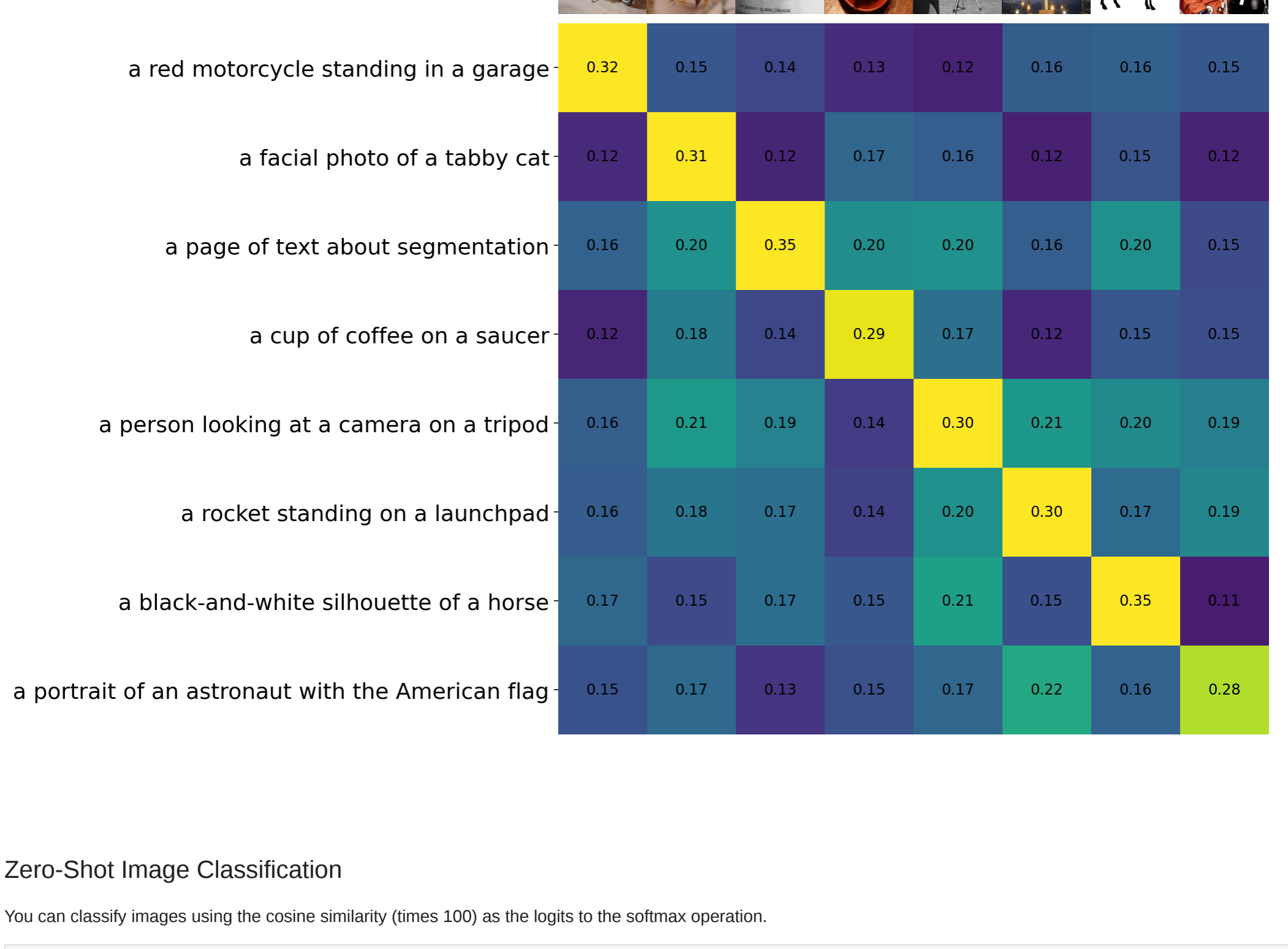
```
image_features /= image_features.norm(dim=-1, keepdim=True)
text_features /= text_features.norm(dim=-1, keepdim=True)
similarity = text_features.cpu().numpy() @ image_features.cpu().numpy().T
```

In [12]:

```
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)
```

Out[12]:

```
Text(0.5, 1.0, 'Cosine similarity between text and image features')
```



Zero-Shot Image Classification

You can classify images using the cosine similarity (times 100) as the logits to the softmax operation.

In [13]:

```
from torchvision.datasets import CIFAR100

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

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz to /root/.cache/cifar-100-python.tar.gz

Extracting /root/.cache/cifar-100-python.tar.gz to /root/.cache
```

In [14]:

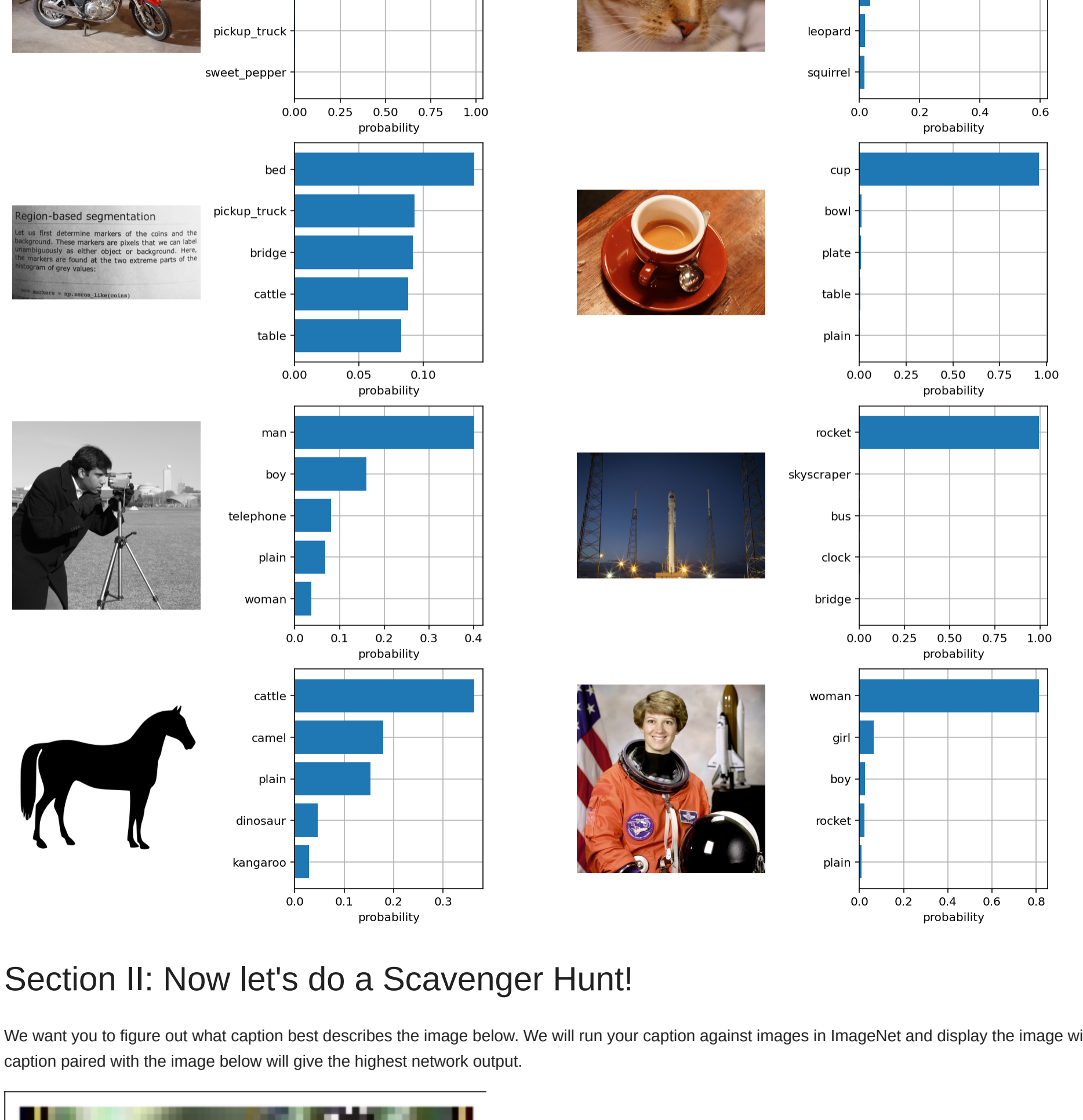
```
text_descriptions = [f"This is a photo of a  {label}" for label in cifar100.classes]
text_tokens = clip.tokenize(text_descriptions).cuda()
```

In [15]:

```
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)
```

In [16]:

```
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()
```



Section II: Now let's do a Scavenger Hunt!

We want you to figure out what caption best describes the image below. We will run your caption against images in ImageNet and display the image with the highest network probability. The goal is that your caption paired with the image below will give the highest network output.

We will download a subset of ImageNet called Tiny ImageNet. Tiny ImageNet has only 200 classes, with each class having 500 traininig images, 50 validation images and 50 test images.

In [17]:

```
! git clone https://github.com/seshuad/IMagenet
```

```
Cloning into 'IMagenet'...
remote: Enumerating objects: 120594, done.
remote: Total 120594 (delta 0), reused 0 (delta 0), pack-reused 120594
Receiving objects: 100% (120594/120594), 212.68 MiB | 25.58 MiB/s, done.
Resolving deltas: 100% (1115/1115), done.
Checking out files: 100% (120206/120206), done.
```

In order to reduce time and memory consumption, we will only consider the first 1000 images in the test set as the possible search space.

In [18]:

```
import os
img_paths = []
for rootdir, subdir, filenames in os.walk("IMagenet/tiny-imagenet-200/test/images"):
  for file_ in sorted(filenames)[:1000]:
    img_paths.append(os.path.join(rootdir, file_))
```

In [21]:

```
'''
TO DO: change caption below to produce target image
'''

caption = "a butterfly on a purple flower"
```

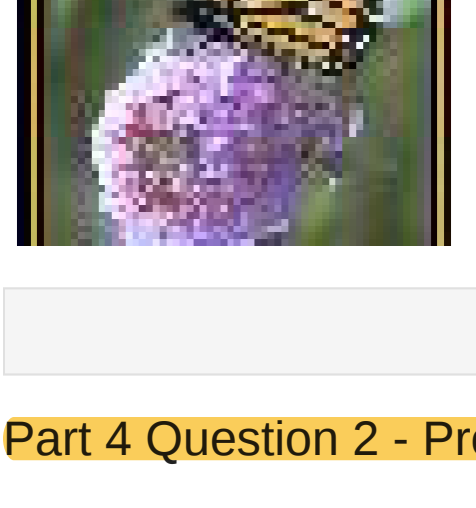
Now, we will run the model for the first 1000 images in the Tiny ImageNet test set. We will display the image that produces the highest network probability with your written caption

In [22]:

```
original_images = []
images = []

for img_path in img_paths:

    image = Image.open(img_path).convert("RGB")
    original_images.append(image)
    images.append(preprocess(image))

image_input = torch.tensor(np.stack(images)).cuda()
with torch.no_grad():
    image_features = model.encode_image(image_input).float()
image_features /= image_features.norm(dim=-1, keepdim=True)

text_tokens = clip.tokenize(caption).cuda()

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=0).cpu().detach().numpy()
highest_prob = np.argmax(text_probs)
plt.axis('off')
plt.imshow(original_images[highest_prob])
```

Out[22]:

```
<matplotlib.image.AxesImage at 0x7ff834960cd0>
```



In [20]:

Part 4 Question 2 - Prompting CLIP:

Finding the caption for this image was easy, which I succeed in my second try. Basically, my initial approach was indicated the two obvious features/objects in the image, butterfly and flower. Then I narrow down to the second feature, which is the colour of the flower and succeeded to find this image.