Preparation for Colab

Make sure you're running a GPU runtime; if not, select "GPU" as the hardware accelerator in Runtime > Change Runtime Type in the menu. The next cells will install the `clip` package and its dependencies, and check if PyTorch 1.7.1 or later is installed.

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

```
Collecting ftfy
  Downloading ftfy-6.0.3.tar.gz (64 kB)
     |████████████████████████████████| 64 kB 1.3 MB/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 (4.41.1)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages (from ftfy) (0.2.5)
Building wheels for collected packages: ftfy
  Building wheel for ftfy (setup.py) ... done
  Created wheel for ftfy: filename=ftfy-6.0.3-py3-none-any.whl size=41934 sha256=90ec193331444b2c4ff1cd81935e7de42065b89d304db7efac6
  Stored in directory: /root/.cache/pip/wheels/19/f5/38/273eb3b5e76dfd850619312f693716ac4518b498f5ffb6f56d
Successfully built ftfy
Installing collected packages: ftfy
Successfully installed ftfy-6.0.3
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-hqnbveqi
  Running command git clone -q https://github.com/openai/CLIP.git /tmp/pip-req-build-hqnbveqi
Requirement already satisfied: ftfy in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (6.0.3)
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) (4.41.1)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (1.9.0+cu102)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (0.10.0+cu102)
Requirement already satisfied: wcwidth 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) (3.7.4.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision->clip==1.0) (1.19.5)
Requirement already satisfied: pillow>=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=1369080 sha256=fda43d2b80cfb2b33c2d43e23ea5f53293a9a8b48d5f9e341de
  Stored in directory: /tmp/pip-ephem-wheel-cache-kmmplf44/wheels/fd/b9/c3/5b4470e35ed76e174bff77c92f91da82098d5e35fd5bc8cdac
Successfully built clip
Installing collected packages: clip
Successfully installed clip-1.0
```

```
import numpy as np
import torch
import clip
from tqdm.notebook import tqdm
from pkg_resources import packaging

print("Torch version:", torch.__version__)
```

```
Torch version: 1.9.0+cu102
```

Loading the model

Download and instantiate a CLIP model using the `clip` module that we just installed.

```
clip.available_models()
```

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

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

```
100%|████████████████████████████████████| 338M/338M [00:05<00:00, 63.6MiB/s]
```

```
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)
```

```
Model parameters: 151,277,313
Input resolution: 224
```

```
Context length: 77
Vocab size: 49408
```

Preparing ImageNet labels and prompts

The following cell contains the 1,000 labels for the ImageNet dataset, followed by the text templates we'll use as "prompt engineering".

```
imagenet_classes = ["tench", "goldfish", "great white shark", "tiger shark", "hammerhead shark", "electric ray", "stingray", "rooster",
```

A subset of these class names are modified from the default ImageNet class names sourced from Anish Athalye's imagenet-simple-labels.

These edits were made via trial and error and concentrated on the lowest performing classes according to top_1 and top_5 accuracy on the ImageNet training set for the RN50, RN101, and RN50x4 models. These tweaks improve top_1 by 1.5% on ViT-B/32 over using the default class names. Alec got bored somewhere along the way as gains started to diminish and never finished updating / tweaking the list. He also didn't revisit this with the better performing RN50x16, RN50x64, or any of the ViT models. He thinks it's likely another 0.5% to 1% top_1 could be gained from further work here. It'd be interesting to more rigorously study / understand this.

Some examples beyond the crane/crane -> construction crane / bird crane issue mentioned in Section 3.1.4 of the paper include:

- CLIP interprets "nail" as "fingernail" so we changed the label to "metal nail".
- ImageNet kite class refers to the bird of prey, not the flying toy, so we changed "kite" to "kite (bird of prey)"
- The ImageNet class for red wolf seems to include a lot of mislabeled maned wolfs so we changed "red wolf" to "red wolf or maned wolf"

```
imagenet_templates = [
    'a bad photo of a {}.',
    'a photo of many {}.',
    'a sculpture of a {}.',
    'a photo of the hard to see {}.',
    'a low resolution photo of the {}.',
    'a rendering of a {}.',
    'graffiti of a {}.',
    'a bad photo of the {}.',
    'a cropped photo of the {}.',
    'a tattoo of a {}.',
    'the embroidered {}.',
    'a photo of a hard to see {}.',
    'a bright photo of a {}.',
    'a photo of a clean {}.',
    'a photo of a dirty {}.',
    'a dark photo of the {}.',
    'a drawing of a {}.',
    'a photo of my {}.',
    'the plastic {}.',
    'a photo of the cool {}.',
    'a close-up photo of a {}.',
    'a black and white photo of the {}.',
    'a painting of the {}.',
    'a painting of a {}.',
    'a pixelated photo of the {}.',
    'a sculpture of the {}.',
    'a bright photo of the {}.',
    'a cropped photo of a {}.',
    'a plastic {}.',
    'a photo of the dirty {}.',
    'a jpeg corrupted photo of a {}.',
    'a blurry photo of the {}.',
    'a photo of the {}.',
    'a good photo of the {}.',
    'a rendering of the {}.',
    'a {} in a video game.',
    'a photo of one {}.',
    'a doodle of a {}.',
    'a close-up photo of the {}.',
    'a photo of a {}.',
    'the origami {}.',
    'the {} in a video game.',
    'a sketch of a {}.',
    'a doodle of the {}.',
    'a origami {}.',
    'a low resolution photo of a {}.',
    'the toy {}.',
    'a rendition of the {}.',
    'a photo of the clean {}.',
    'a photo of a large {}.',
    'a rendition of a {}.',
```

```
    'a photo of a nice {}.',
    'a photo of a weird {}.',
    'a blurry photo of a {}.',
    'a cartoon {}.',
    'art of a {}.',
    'a sketch of the {}.',
    'a embroidered {}.',
    'a pixelated photo of a {}.',
    'itap of the {}.',
    'a jpeg corrupted photo of the {}.',
    'a good photo of a {}.',
    'a plushie {}.',
    'a photo of the nice {}.',
    'a photo of the small {}.',
    'a photo of the weird {}.',
    'the cartoon {}.',
    'art of the {}.',
    'a drawing of the {}.',
    'a photo of the large {}.',
    'a black and white photo of a {}.',
    'the plushie {}.',
    'a dark photo of a {}.',
    'itap of a {}.',
    'graffiti of the {}.',
    'a toy {}.',
    'itap of my {}.',
    'a photo of a cool {}.',
    'a photo of a small {}.',
    'a tattoo of the {}.',
]

print(f"{len(imagenet_classes)} classes, {len(imagenet_templates)} templates")
```

```
1000 classes, 80 templates
```

A similar, intuition-guided trial and error based on the ImageNet training set was used for templates. This list is pretty haphazard and was gradually made / expanded over the course of about a year of the project and was revisited / tweaked every few months. A surprising / weird thing was adding templates intended to help ImageNet-R performance (specifying different possible renditions of an object) improved standard ImageNet accuracy too.

After the 80 templates were "locked" for the paper, we ran sequential forward selection over the list of 80 templates. The search terminated after ensembling 7 templates and selected them in the order below.

1. itap of a {}.
2. a bad photo of the {}.
3. a origami {}.
4. a photo of the large {}.
5. a {} in a video game.
6. art of the {}.
7. a photo of the small {}.

Speculating, we think it's interesting to see different scales (large and small), a difficult view (a bad photo), and "abstract" versions (origami, video game, art), were all selected for, but we haven't studied this in any detail. This subset performs a bit better than the full 80 ensemble reported in the paper, especially for the smaller models.

Loading the Images

The ILSVRC2012 datasets are no longer available for download publicly. We instead download the ImageNet-V2 dataset by Recht et al..

If you have the ImageNet dataset downloaded, you can replace the dataset with the official torchvision loader, e.g.:

```
images = torchvision.datasets.ImageNet("path/to/imagenet", split='val', transform=preprocess)
```

```
! pip install git+https://github.com/modestyachts/ImageNetV2_pytorch

from imagenetv2_pytorch import ImageNetV2Dataset

images = ImageNetV2Dataset(transform=preprocess)
loader = torch.utils.data.DataLoader(images, batch_size=32, num_workers=2)
```

```
Collecting git+https://github.com/modestyachts/ImageNetV2_pytorch
  Cloning https://github.com/modestyachts/ImageNetV2_pytorch to /tmp/pip-req-build-0kih0kn2
  Running command git clone -q https://github.com/modestyachts/ImageNetV2_pytorch /tmp/pip-req-build-0kih0kn2
Building wheels for collected packages: imagenetv2-pytorch
```

```
  Building wheel for imagenetv2-pytorch (setup.py) ... done
  Created wheel for imagenetv2-pytorch: filename=imagenetv2_pytorch-0.1-py3-none-any.whl size=2663 sha256=ac31e0ed9c61afc5e0271eed31
  Stored in directory: /tmp/pip-ephem-wheel-cache-745b5n1m/wheels/ab/ee/f4/73bce5c7f68d28ce632ef33ae87ce60aaca021eb2b3b31a6fa
Successfully built imagenetv2-pytorch
Installing collected packages: imagenetv2-pytorch
Successfully installed imagenetv2-pytorch-0.1
Dataset matched-frequency not found on disk, downloading....
100%|██████████| 1.26G/1.26G [01:02<00:00, 20.2MiB/s]
Extracting....
```

Creating zero-shot classifier weights

```
def zeroshot_classifier(classnames, templates):
    with torch.no_grad():
        zeroshot_weights = []
        for classname in tqdm(classnames):
            texts = [template.format(classname) for template in templates] #format with class
            texts = clip.tokenize(texts).cuda() #tokenize
            class_embeddings = model.encode_text(texts) #embed with text encoder
            class_embeddings /= class_embeddings.norm(dim=-1, keepdim=True)
            class_embedding = class_embeddings.mean(dim=0)
            class_embedding /= class_embedding.norm()
            zeroshot_weights.append(class_embedding)
        zeroshot_weights = torch.stack(zeroshot_weights, dim=1).cuda()
    return zeroshot_weights


zeroshot_weights = zeroshot_classifier(imagenet_classes, imagenet_templates)
```

```
100% 1000/1000 [16:51<00:00, 1.01s/it]
```

Zero-shot prediction

```
def accuracy(output, target, topk=(1,)):
    pred = output.topk(max(topk), 1, True, True)[1].t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))
    return [float(correct[:k].reshape(-1).float().sum(0, keepdim=True).cpu().numpy()) for k in topk]
```

```
with torch.no_grad():
    top1, top5, n = 0., 0., 0.
    for i, (images, target) in enumerate(tqdm(loader)):
        images = images.cuda()
        target = target.cuda()

        # predict
        image_features = model.encode_image(images)
        image_features /= image_features.norm(dim=-1, keepdim=True)
        logits = 100. * image_features @ zeroshot_weights

        # measure accuracy
        acc1, acc5 = accuracy(logits, target, topk=(1, 5))
        top1 += acc1
        top5 += acc5
        n += images.size(0)

top1 = (top1 / n) * 100
top5 = (top5 / n) * 100

print(f"Top-1 accuracy: {top1:.2f}")
print(f"Top-5 accuracy: {top5:.2f}")
```

```
100% 313/313 [02:31<00:00, 2.07it/s]

Top-1 accuracy: 55.93
Top-5 accuracy: 83.36
```