Google Universal Image Embedding

Create image representations that work across many visual domains

比赛介绍

在这次比赛中，参赛者开发的模型用作于检索到与查询相同对象的数据库图像。比赛数据集中的图像包括各种物体类型，如服装、艺术品、地标、家具、包装品等等。

比赛是以表征学习的形式进行的：你将创建一个模型来提取图像的特征嵌入（Embedding），并通过Kaggle Notebooks提交该模型。Kaggle将在测试集上运行你的模型，进行k-nearest-neighbors查找，并对所得的Embedding质量进行评分。Tensorflow和PyTorch模型均支持。

- 竞赛类型：本次竞赛属于**计算机视觉/特征嵌入**，所以推荐使用的模型或者库：**EfficientNet/CLIP/Swin Transformer**
- 赛题数据：**本次比赛不提供训练集。**根据规则，任何训练数据都可以使用，只要在相关截止日期前在论坛上披露即可。
- 评估标准：我们根据 mean Precision@5的指标对提交的数据进行评估，其中我们引入了一个小的修改，以避免对少于5张预期索引图片的查询进行惩罚。详见：Google Universal Image Embedding | Kaggle
- 推荐阅读 Kaggle 内的一篇 EDA（探索数据分析）来获取一些预备知识：guie_CLIP_TensorFlow_train_example | Kaggle

数据说明

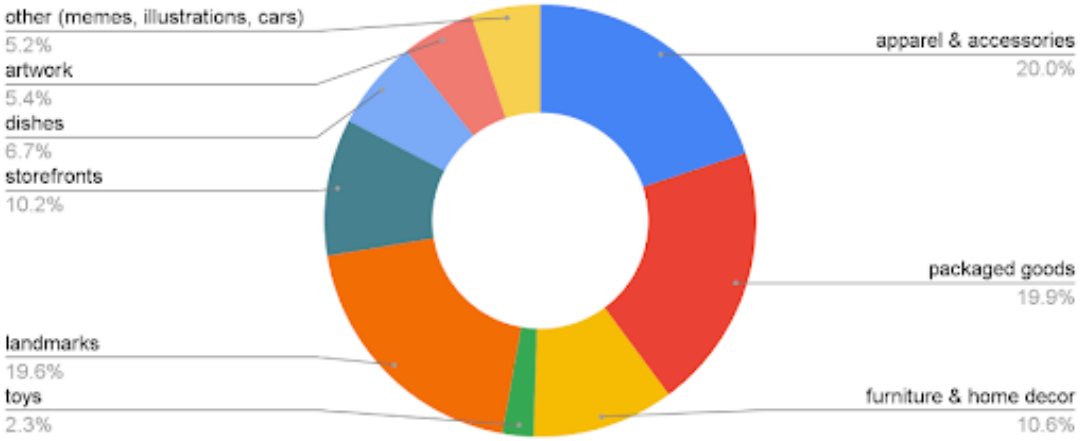
官方数据页面 Google Universal Image Embedding | Kaggle

在这个竞赛中，参赛者被要求开发能够从一个大型数据库中有效检索图像的模型。**对于每张查询图片，模型要从一个索引集中检索出最相似的图片。**

主办方不提供训练集。根据规则，任何训练数据都可以被使用，只要在相关的截止日期前在论坛上披露。目前存在许多针对不同对象类型（如艺术品、地标、产品等）的公共数据集，鼓励参赛者根据需要进行实验。

最终的评估数据集是保密的，它包含以下类型的对象的图像：服装和配件、包装商品、地标、家具和家庭装饰、店面、餐具、艺术品、玩具、备忘录、插图和汽车。最顶部的标题图片显示了这些领域中的图像例子，它们与评估集中的图像相似。标签一般是在实例层面上定义的，例如：同样的T恤，同样的建筑，同样的绘画，同样的菜肴。

下面的图显示了数据集中物体类型的分布。



方案提交方式

提交文件

与传统的Kaggle代码竞赛不同的是，在这个竞赛中，你将提交一个模型文件，因为在这个竞赛中，笔记本会在一个私有测试集上从头到尾地重新运行。该模型必须接受一张图片作为输入，并在输出端返回一个浮点向量（即图片embedding）。embedding的维度应不大于64。你的模型必须被打包成一个submission.zip文件，并与TensorFlow 2.6.4或Pytorch 1.11.0兼容。在大多数情况下，评分预计需要几个小时来完成。

submission.zip文件应包含以下内容之一（且只有一个）。

Tensorflow的SavedModel格式的文件和目录。
PyTorch的TorchScript格式的文件，名为 "saved_model.pt"。

再次强调:模型生成的嵌入必须不超过64D(如果不是，提交将出错)。

评分流程

Kaggle将使用提交的模型来:为私有测试和索引图像集提取嵌入。使用测试和索引嵌入之间的欧氏距离，为每个测试样本创建一个kNN (k= 5)查找。使用比赛评分公式对查找的质量进行评分。

官方baseline

要提交模型，您所需要的是一个生成包含模型的submit .zip文件的内核。以下是一些例子:

Tensorflow：查看这篇笔记了解最小baseline模型，查看这篇笔记了解更复杂的baseline模型。

PyTorch：看到这篇笔记。

生成模型的代码示例

对于TensorFlow，我们基于验证的Inception-V3模型提供最小的张量基线模型，该模型具有线性层，将输出功能投影到64D。最后，该模型以SavedModel格式保存：

```
import tensorflow as tf

image = tf.keras.layers.Input([None, None, 3], dtype=tf.uint8)
output = tf.cast(image, tf.float32)
output = tf.keras.layers.Lambda(lambda x: tf.image.resize(x, [224, 224]),
name='resize')(output)
output = tf.keras.applications.inception_v3.preprocess_input(output)
output = tf.keras.applications.inception_v3.InceptionV3(weights='imagenet',
input_shape=[None, None, 3],
                                                        include_top=False,
pooling='avg')(output)
output = tf.keras.layers.Dense(64, name='embedding')(output)
output = tf.keras.layers.Lambda(lambda x: tf.math.l2_normalize(x, axis=-1),
name='embedding_norm')(output)
model = tf.keras.Model(inputs=[image], outputs=[output])
model.summary()

tf.saved_model.save(model, 'inceptionv3/')
```

要了解更复杂的示例，请查看此仓库中给出的示例。特别地，这个脚本显示了一个模型导出的例子，这个模型是用这个脚本训练的。

对于PyTorch，请参阅下面的类似示例，该示例使用预先训练的Inception-v3模型和线性投影层，并以TorchScript格式导出模型

```
import torch
import torch.nn as nn
from torchvision import models
from torchvision import transforms

class MyModel(nn.Module):
  def __init__(self):
    super().__init__()
    inception_model = models.inception_v3(pretrained=True)
    inception_model.fc = nn.Linear(2048, 64)
    self.feature_extractor = inception_model

  def forward(self, x):
    x = transforms.functional.resize(x,size=[224, 224])
    x = x/255.0
    x = transforms.functional.normalize(x,
                                        mean=[0.485, 0.456, 0.406],
                                        std=[0.229, 0.224, 0.225])
    return self.feature_extractor(x).logits
```

```
model = MyModel()
model.eval()
saved_model = torch.jit.script(model)
saved_model.save('saved_model.pt')
```

特征提取代码

当您的模型提交后，我们将在测试集图像中运行它，以便提取embedding。请查看下面的代码片段，它们详细展示了如何使用模型。这些可以帮助参赛者确保他们的模型正确地提取embedding。

Tensorflow:

```
import numpy as np
from PIL import Image
import tensorflow as tf

# Model loading.
model = tf.saved_model.load(saved_model_path)
embedding_fn = model.signatures["serving_default"]

# Load image and extract its embedding.
image_tensor = tf.convert_to_tensor(
        np.array(Image.open(image_path).convert("RGB"))
    )
expanded_tensor = tf.expand_dims(image_tensor, axis=0)
embedding = embedding_fn(expanded_tensor)["embedding_norm"]
```

PyTorch:

```
from PIL import Image
import torch
from torchvision import transforms

# Model loading.
model = torch.jit.load(saved_model_path)
model.eval()
embedding_fn = model

# Load image and extract its embedding.
input_image = Image.open(image_path).convert("RGB")
convert_to_tensor = transforms.Compose([transforms.PILToTensor()])
input_tensor = convert_to_tensor(input_image)
input_batch = input_tensor.unsqueeze(0)
with torch.no_grad():
  embedding = torch.flatten(embedding_fn(input_batch)[0]).cpu().data.numpy()
```

解决方案思路

1. 本次竞赛的方案是基于 CLIP ViT-H/14 on LAION-2B 模型并为其添加ArcFace层，训练图像检索能力。
2. 因为本次竞赛不提供官方训练集，主办方只提供了图片的类型，如服装和配件、包装商品、地标、家具和家庭装饰等等，所以我们针对性的选取了三份数据集作为我们的训练和验证图像。
    - imagenet1k：该数据集由imagenet(1k)数据集创建。为了减少数据集的大小，这个数据集每个类只有50个图像。
    - products10k：此数据集是从product10k数据集创建的。为了减少数据集的大小，这个数据集每个类只有50个图像。
    - google landmark recognition 2021(Competition dataset)：此数据集是从kaggle过往的比赛数据集创建的。为了减少数据集的大小，这个数据集使用最上面的7k类图像和大量的图像(每个类50个图像)。
3. 本次训练使用的设备是是Google的TPU，部分代码针对TPU做了优化，可以在Kaggle或者Google Colab复现代码结果。

比赛上分历程

1. clip-vit-large-patch14 baseline，Public Score: 0.527
2. Arcface 从256D转成64D, 提升: ~0.01
3. 使用大模型CLIP ViT-H/14 on LAION-2B，提升: ~0.08
4. 减小学习率，提升: ~0.01
5. 在训练结束后，用更低的学习率微调，提升: ~0.02

代码、数据集

- 代码
    - guie code.ipynb
- 数据集
    - datasets.txt

TL;DR

在这次竞赛中，我们需要开发的一个通用图像检索的模型。最终我们需要预测出私有测试集中图像的Embedding，然后主办方根据Embedding 的Knn相似度来评估成绩。本次比赛的一大特点是，主办方并不提供训练集，只提供图像类别的分布占比，如服装、商品、地标等等。我们根据图像类别的分别，针对性的选择三个较大的数据集（其中包括imagenet1k）用作训练和验证。然后选择了few-shot能力特别突出的CLIP作为我们的预训练模型。因为模型较大，数据量较多，我们训练都在kaggle的提供的TPU上完成，代码也做出了针对性的优化调整。最终我们获得了0.65的@5 精度。