LookForJayChou

寻找周杰伦

ZF1921335 林星辰

GoogleColab:
https://colab.research.google.com/drive/13uFqZmEl9FRfQ2kip8osnCSqDtwZNk5T?usp=sharing

GitHub:

https://github.com/SUTFutureCoder/LookForJayChou

深度学习神器

Google Colab

https://colab.research.google.com/

Why LookForJayChou

寻找周杰伦

用于练习图像检索会很有意思的样子

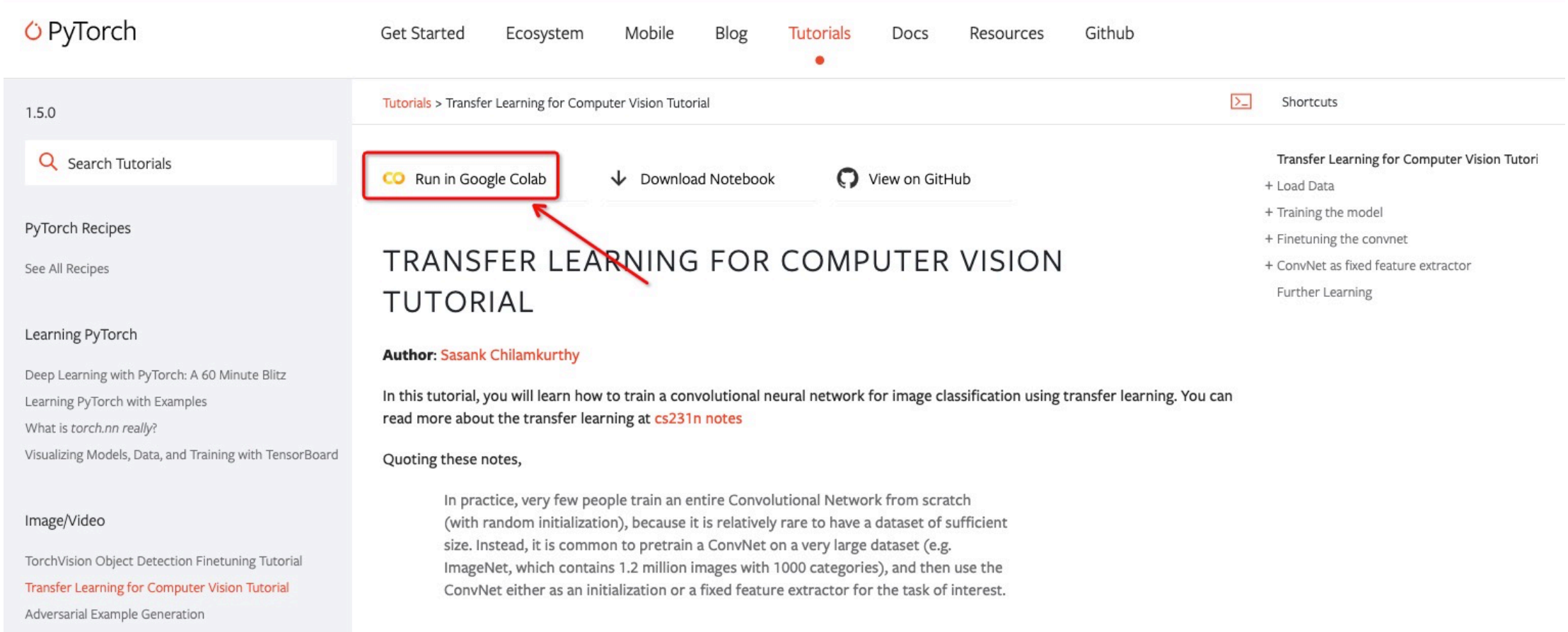
十年前我也是饭圈男孩！
┬┴⑩咋騚珮也逞飯圈楠孓亥！_/~↘

有那味了，开搞

Why Colab

- 云端执行，移动开发
- 免费预装环境及GPU资源
- PyTorch等依赖自动预装
- PyTorch等墙外依赖预训练模型资源极速下载(14M/s+)
- 方便调试、输出、保存、复现结果
- 允许挂载GoogleDrive，快速保存、下载数据集和模型

Facebook PyTorch教程挂着竞对链接



证明Google Colab真的很好用啊

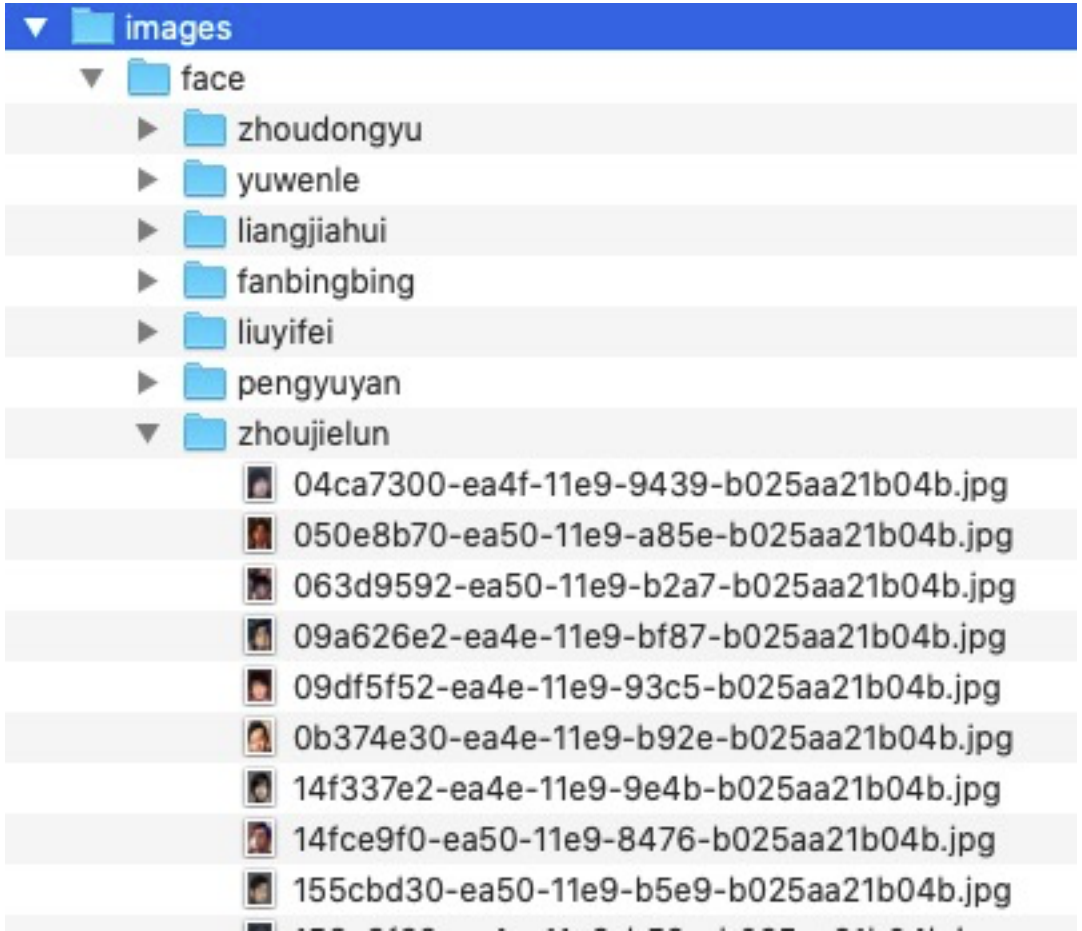
STEP 0 简化定义检索问题

检索可以看做多分类、Dense输出之间的余弦距离问题

一个基于VGGNet的图像检索小工程 https://github.com/matsui528/sis

STEP 1 数据集

十个明星的面部数据集共1354个面部特写照片 文件夹为名字
https://aistudio.baidu.com/aistudio/datasetDetail/13959



STEP 2 预训练模型选型

Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	
NASNetLarge	343 MB	0.825	0.960	88,949,818	
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

过气模型

入门模型

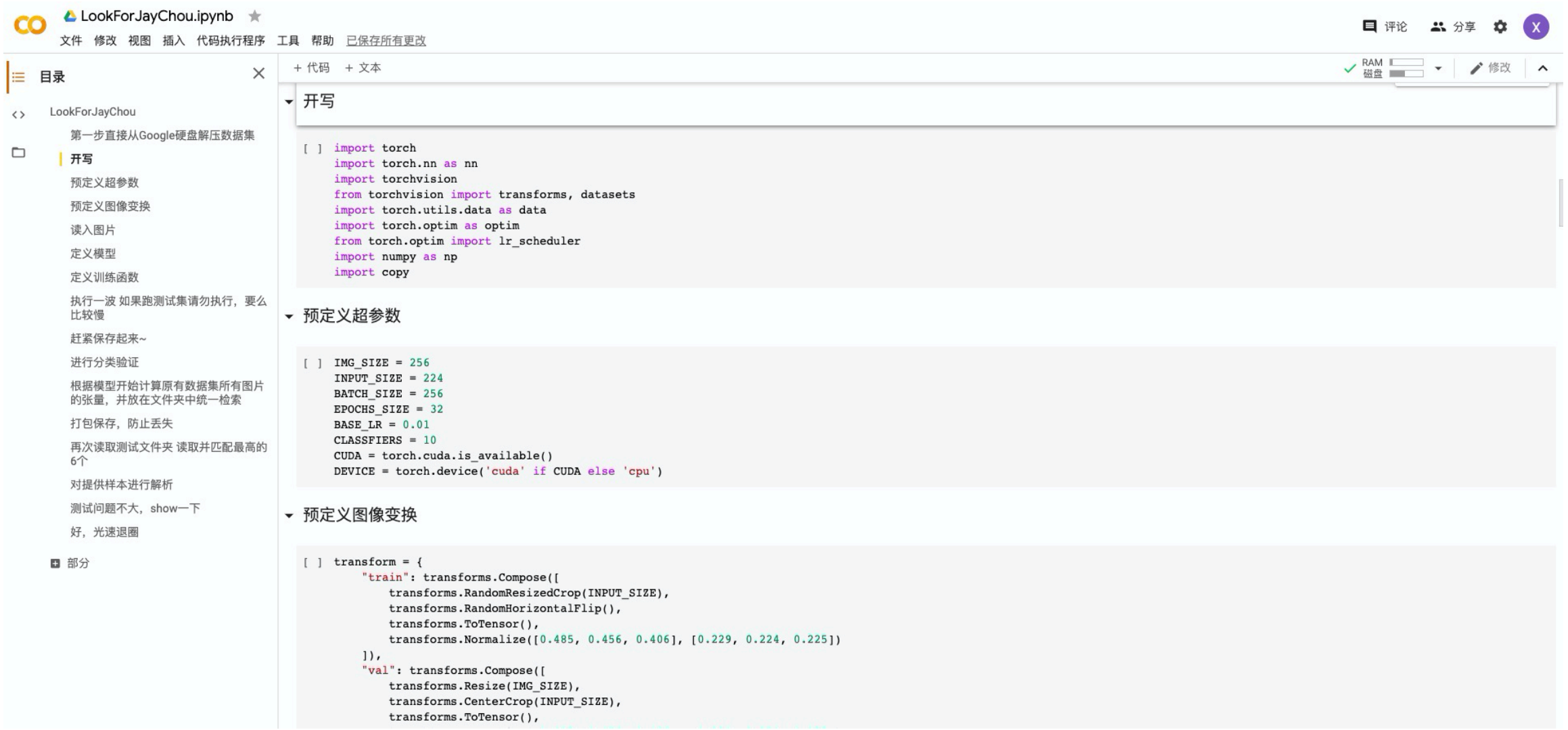
适合小型设备
参数最多训练慢

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

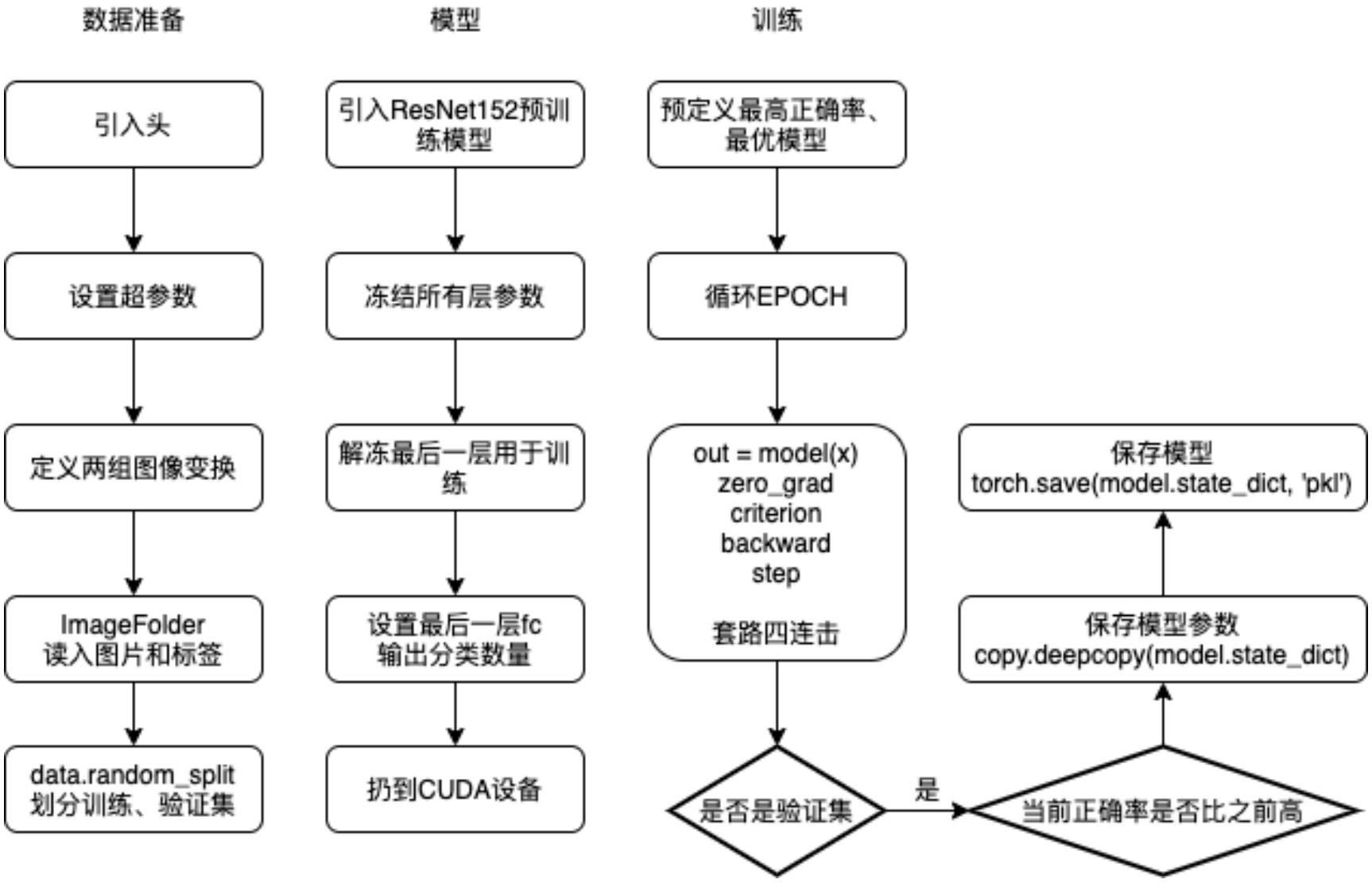
Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

https://keras.io/api/applications/

STEP 3 在Google Colab中进行开发



STEP 4 模型训练步骤【对应Colab目录1~9步】



STEP 5 测试模型 【对应Colab目录第10节】

```
# 读入模型
model_state_dict = torch.load('./mdl.pkl')

# 定义空模型
model = torchvision.models.resnet152(pretrained=False, num_cla

# 加载
model.eval()
model.load_state_dict(model_state_dict)

# dataset
testdataset = TestDataSets('./test_img/', transform['val'])

# dict
idx_name_dict = testdataset.get_idx_name_dict(data_dir)
# print(idx_name_dict)

# load
for x in data.DataLoader(testdataset):
  x.to(DEVICE)
  out = model(x)
  _, pred = torch.max(out, 1)

  print(idx_name_dict[pred.item()])
```

```
zhoujielun
zhoujielun
```

STEP 6 提取最后一层Tensor参数

```
[ ]  # 重新遍历，然后处理放入文件夹中
     yi = 0
     pre_y = -1
     test_data_set = DataSet(image_datasets, transform['val'])
     for x, y in data.DataLoader(test_data_set):
       if pre_y != y.item():
         pre_y = y.item()
         yi = 0
       x.to(DEVICE)
       out = model(x)
       torch.save(out[0], './facetensor/' + idx_name_dict[y.item()] + "_" + str(yi) + ".pkl")
       yi += 1
```

保存至./facetensor/******_id.pkl中用于检索

基于磁盘的检索感觉IO比较低，后续想写一个检索数据库

TIPS 注意备份至Google Drive

赶紧保存起来~

```
[ ]  !cp ./mdl.pkl /content/drive/My\ Drive/AIColab/LookForJayChou/
```

▾ 打包保存，防止丢失

```
[ ]  !rm -rf /content/drive/My\ Drive/AIColab/LookForJayChou/facefeature.tar.gz
     !tar zcf facefeature.tar.gz ./facetensor
     !cp facefeature.tar.gz /content/drive/My\ Drive/AIColab/LookForJayChou/
```

▾ 再次读取测试文件夹 读取并匹配最高的6个

```
[ ]  !cp /content/drive/My\ Drive/AIColab/LookForJayChou/facefeature.tar.gz ./
     !tar zxvf facefeature.tar.gz
```

STEP 7 提供验证集样本，并检查匹配度为0

对提供样本进行解析

速度还算挺快

```
for param in model.parameters():
    param.requires_grad = False

for x in data.DataLoader(testdataset):
  x.to(DEVICE)
  out = model(x)
  out.requires_grad=False
  out[0] = out[0] / np.linalg.norm(out[0])
  match_dict = {}
  for y in list(os.listdir("./facetensor")):
    y_tensor = torch.load(os.path.join("./facetensor", y))
    y_tensor.requires_grad = False
    y_tensor = y_tensor / np.linalg.norm(y_tensor)
    dists = np.linalg.norm(y_tensor - out[0])
    match_dict[y] = dists
  sorted_dict = sorted(match_dict.items(), key=lambda x: x[1])
  print(sorted_dict[:6])
  # print(idx_name_dict[pred.item()])
```

别忘了关梯度下降标记

输入图片和现有文件各自计算

核心比较方法

越小匹配度约高

取前6

用原图测试，几乎等于0

```
[('zhoujielun_54.pkl', 5.2265455e-07), ('zhoujielun_16.pkl', 0.4986408), ('zhoujielun_23.pkl', 0.5094757),
[('zhoujielun_8.pkl', 2.839358e-07), ('zhoujielun_9.pkl', 0.17602883), ('zhoujielun_20.pkl', 0.17810228),
```

STEP 8 可视化

```
[ ] for x in data.DataLoader(testdataset):
      x.to(DEVICE)
      out = model(x)
      out.requires_grad=False
      out[0] = out[0] / np.linalg.norm(out[0])
      match_dict = {}
      for y in list(os.listdir("./facetensor")):
        y_tensor = torch.load(os.path.join("./facetensor", y))
        y_tensor.requires_grad = False
        y_tensor = y_tensor / np.linalg.norm(y_tensor)
        dists = np.linalg.norm(y_tensor - out[0])
        match_dict[y] = dists
      sorted_dict = sorted(match_dict.items(), key=lambda x: x[1])[:6]
```

直接拷贝上个STEP

```
    plt.figure("寻找周杰伦")
    plt.subplots_adjust(wspace = 0.2, hspace = 0.2)
    plt.subplot(3, 3, 1)
    plt.title("INPUT")
    plt.imshow(x[0, -1])
    plt.show()
    i = 4
    for name, probabliity in sorted_dict:
      # 从文件夹中找到
      splited_dir = name.split(".")[0].split("_")
      search_orig_path = os.path.join("./images/face/", splited_dir[0])

      path_file = os.listdir(search_orig_path)
      path_file.sort()
      path_file_image = Image.open(os.path.join(search_orig_path, path_file[int(splited_dir[1])]))

      plt.subplots_adjust(top = 2, right = 2)
      plt.subplot(3, 3, i)
      plt.imshow(path_file_image)
      plt.title(str(probabliity) + "\n" + splited_dir[0])

      i += 1

    plt.show()

    print("-"*10)
```

简单可视化一下

不反归一化，图片比较绿

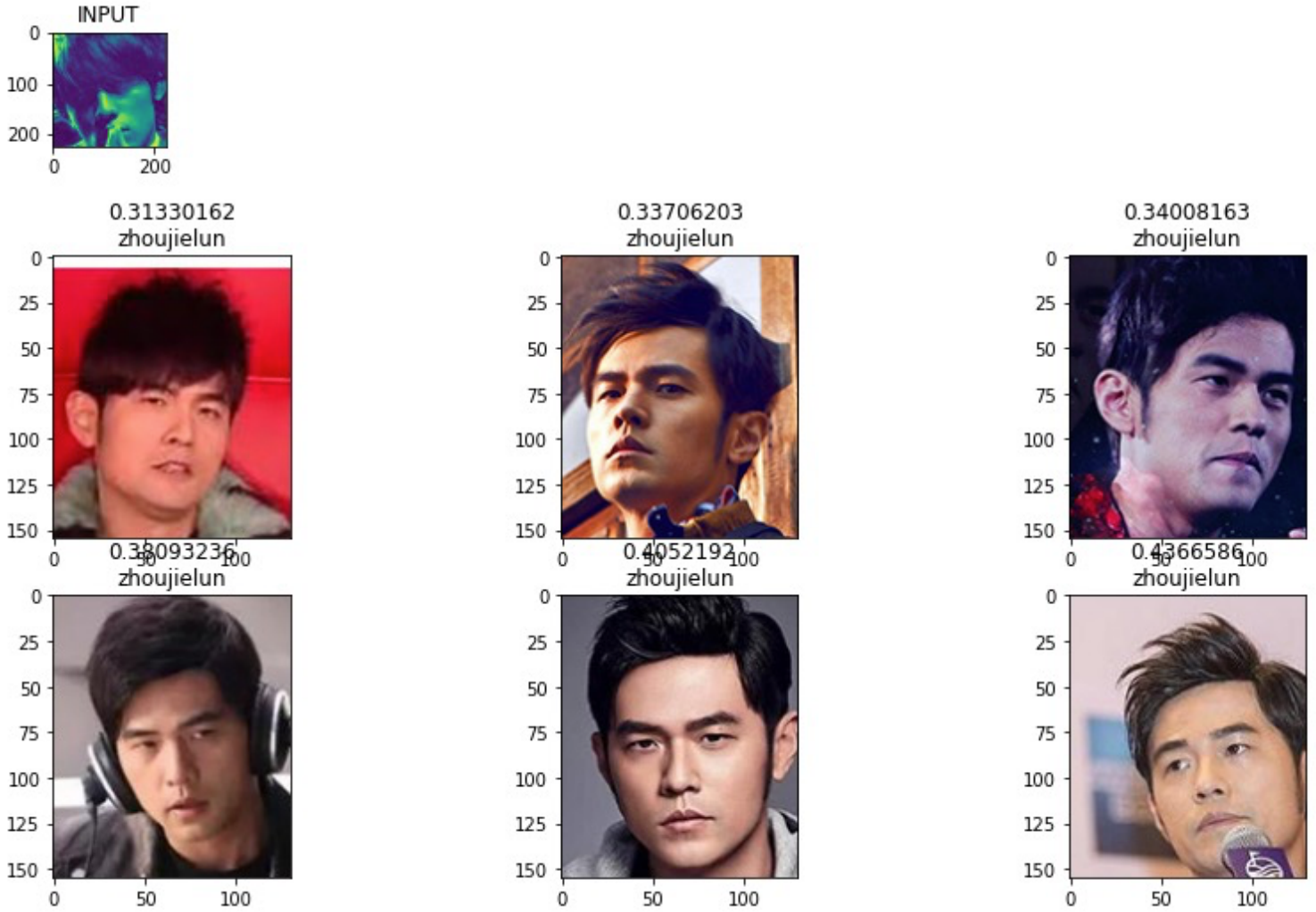
实际检索


INPUT
0.28615862
zhoujielun
0.2885676
zhoujielun
0.30444118
zhoujielun
0.35983804
zhoujielun
0.39811543
zhoujielun
0.3999648
zhoujielun

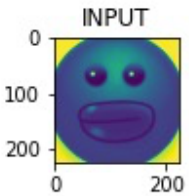
实际检索

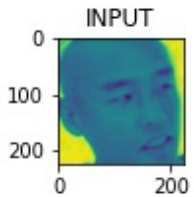

INPUT
0.20940442
zhoujielun
0.27366015
zhoujielun
0.2787481
zhoujielun
0.30029666
zhoujielun
0.31117082
zhoujielun
0.32059973
zhoujielun

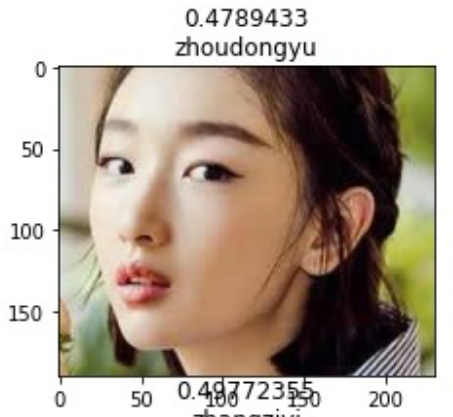
五官遮挡测试泛化能力case

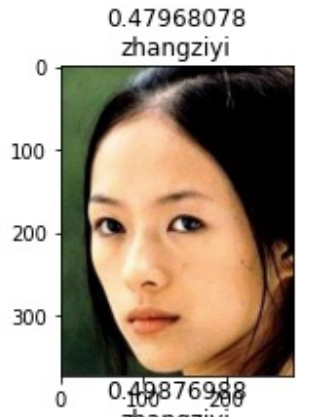


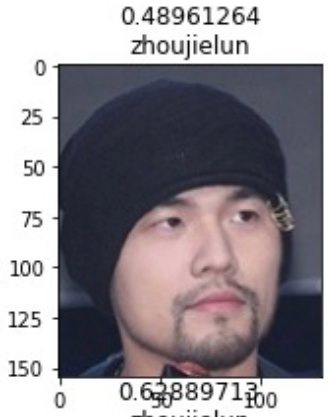
混入 “奇怪” 的例子

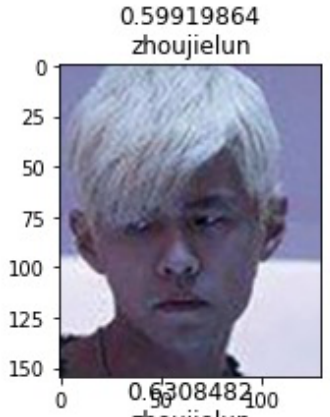

INPUT

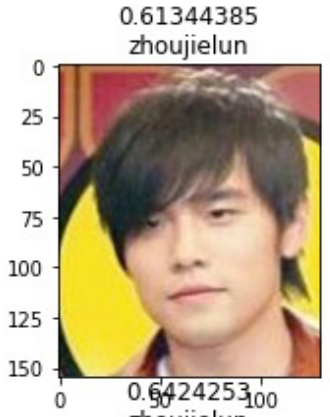

INPUT


0.4789433
zhoudongyu
0.49772355
zhangziyi


0.47968078
zhangziyi
0.49876988
zhangziyi


0.48961264
zhoujielun
0.62889713
zhoujielun


0.59919864
zhoujielun
0.6308482
zhoujielun


0.61344385
zhoujielun
0.6424253
zhoujielun

Result

- 适时脱离书本散装神经网络写法，使用预训练模型事半功倍
- 微调预训练模型最后一层 要比 散装神经网络随机初始化轻松
- 重视transforms归一化，其参数要和预训练模型保持一致
- 训练集和验证集的Resize、Crop类型需要根据情况指定
- 定义合适的BatchSize和Epoch超参数，防止训练不足或GPU OOM
- 谨慎处理图片维度,防止图像信息丢失(比如CT图像部分是4维)
- 尽量使用ResNet及以后的预训练模型
- 选择预训练模型，除准确率外，运行设备和速度制约参数size多少
- 研究不明白了赶紧求助业内人士，学习书本外套路