1. 图像分类

1.1 目的

1. 使用TensorFlow -gpu跑模型
2. 利用卷积神经网络

1.2 前期准备

1. Anaconda 进行环境管理
2. TensorFlow2.3 + TensorFlow-gpu2.3
3. nvidia驱动+cuda11.0+cudnn

1.3 实现目的过程中的解决问题

1.3.1 安装

conda 常用命令：

1. 检测目前安装了哪些环境：`conda info --envs`
2. 创建一个环境并安装不同版本的python：`conda create --name tensorflow python=3.5`
3. 激活创建的环境：`activate tensorflow`
4. 查看python版本 `python --version`
5. 退出当前环境：`deactivate`
6. 查看当前环境下有多少库 ：`pip list`

pip 常用命令 ：

1. 升级：`python -m pip install --upgrade pip`
2. 查看目前的pip版本：在命令操作窗口输入`pip show pip`

1.3.2 TensorFlow

安装CPU版本输入

pip install --ignore-installed --upgrade tensorflow

安装GPU版本输入

pip install --ignore-installed --upgrade tensorflow-gpu

查看本机显卡配置是否支持安装gpu版本的tensorflow: https://blog.csdn.net/sinat_23619409/article/details/84201743?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-4.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-4.control

TensorFlow在Win10上的安装教程和简单示例：https://blog.csdn.net/Eppley/article/details/79297503

1.3.3 nvidia

查看版本 https://blog.csdn.net/Gabriel_wei/article/details/112595642

GPU对应 tensorflow-gpu版本 https://tensorflow.google.cn/install/source_windows#gpu

对应的tensorflow要和 tensorflow-gpu版本相互兼容

搜索缺失的dll：

https://cn.dll-files.com/download/dae6bbb218bc4091223a48a97b6eba4b/cudnn64_8.dll.html?c=S0ppNzlqcVlwcnhHN2UzdC9rbTUwUT09

cuda 路径： C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA

windows10+nvidia驱动+cuda10.1+cudnn安装教程：

https://blog.csdn.net/fengxinzioo/article/details/105646969?utm_medium=distribute.pc_relevant_download.none-task-blog-baidujs-1.nonecase&depth_1-utm_source=distribute.pc_relevant_download.none-task-blog-baidujs-1.nonecase

tensorflow训练模型时指定所用的GPU： https://blog.csdn.net/qq_42250789/article/details/107070520

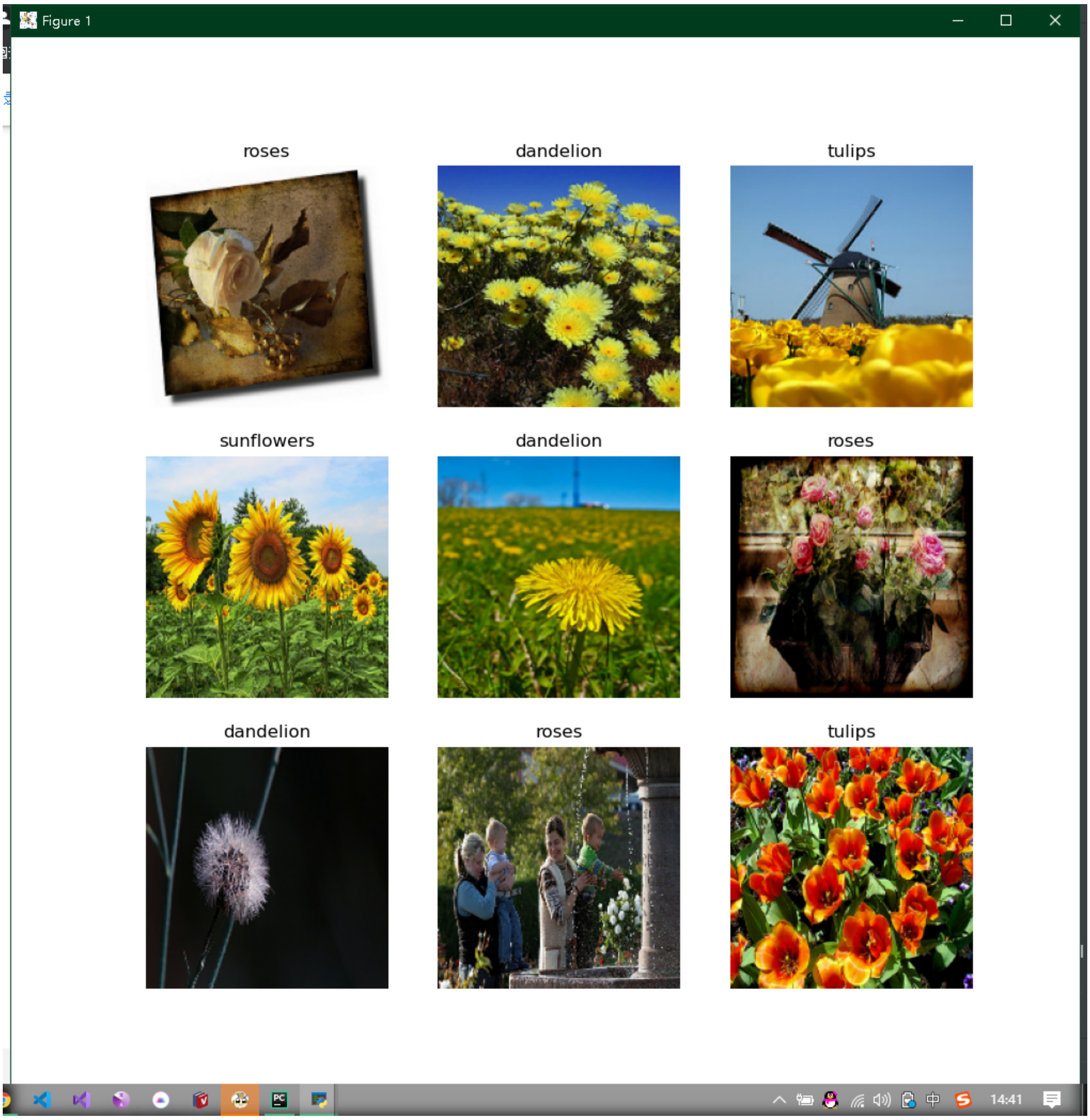
验证程序加载失败，请检查您的浏览器设置，例如广告拦截程序： https://blog.csdn.net/qq_25232685/article/details/119209208

1.4 具体代码及结论：

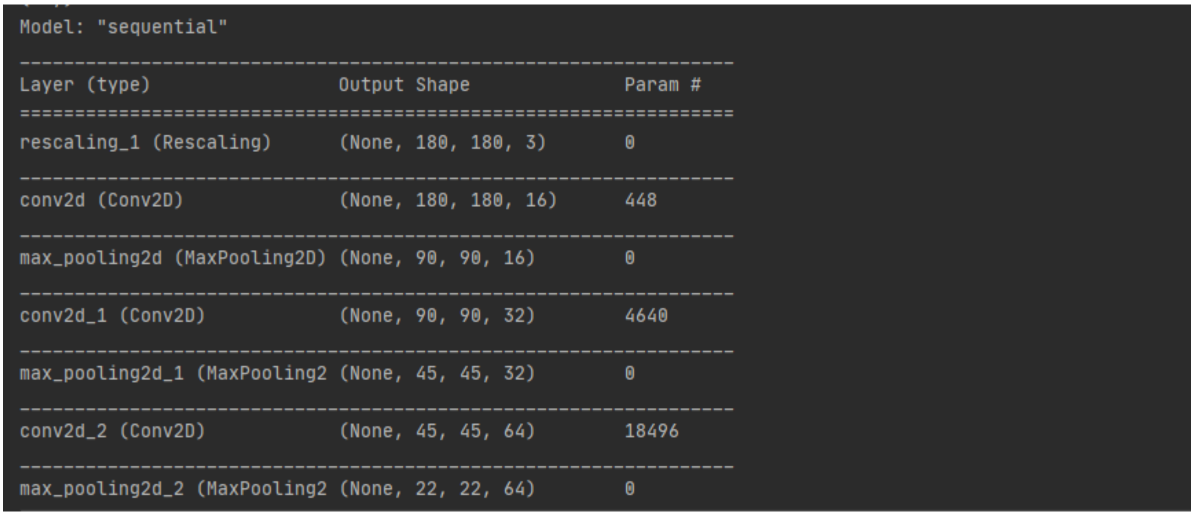
```
# 图像分类

# 卷积神经网络
# 图像增强

import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

# 1. Download and explore the dataset

import pathlib

```

```python
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file(
    'flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)

# 1.1 展示图片数量

image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)

# 输出 3670

# 1.2 查看图片

roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
PIL.Image.open(str(roses[1]))

# 2. Load using keras.preprocessing

# 2.1 创建数据集

batch_size = 32
img_height = 180
img_width = 180

# 80% of the images for training, and 20% for validation.

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

# Found 3670 files belonging to 5 classes.
# Using 2936 files for training.

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

# Found 3670 files belonging to 5 classes.
# Using 734 files for validation.

# 查找 class_names

class_names = train_ds.class_names
print(class_names)

# 输出 ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']

# 2.2 可视化数据
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")

# 2.3 查看测试数据

for image_batch, labels_batch in train_ds:
    print(image_batch.shape)
    print(labels_batch.shape)
    break

# 输出 :
# (32, 180, 180, 3)
# (32,)

# 这是一批 32 张形状为 180x180x3 的图像（最后一个维度是指颜色通道 RGB）。

# 2.4 缓冲预取数据
# Dataset.cache() 在第一个时期从磁盘加载图像后将图像保存在内存中。
# 这将确保数据集在训练模型时不会成为瓶颈。 如果您的数据集太大而无法放入内存，可以使用此方法来创建高性能的磁盘缓存。
#
# Dataset.prefetch() 在训练时重叠数据预处理和模型执行。

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# 2.5 标准化数据
normalization_layer = layers.experimental.preprocessing.Rescaling(1. / 255)

normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.

# print(np.min(first_image), np.max(first_image))

# 3. 创建模型 由三个卷积块组成，每个块都有一个最大池层。 有一个全连接层，上面有 128 个单元，由 relu 激活函数激活。

num_classes = 5

model = Sequential([
    layers.experimental.preprocessing.Rescaling(1. / 255, input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])

# 3.1 编译模型
# 选择 optimizers.Adam 优化器和 loss.SparseCategoricalCrossentropy 损失函数。

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
# 3.2 打印模型
# model.summary()

# 3.3 训练模型

epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
# 3.4 可视化结果
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

可视化图片：

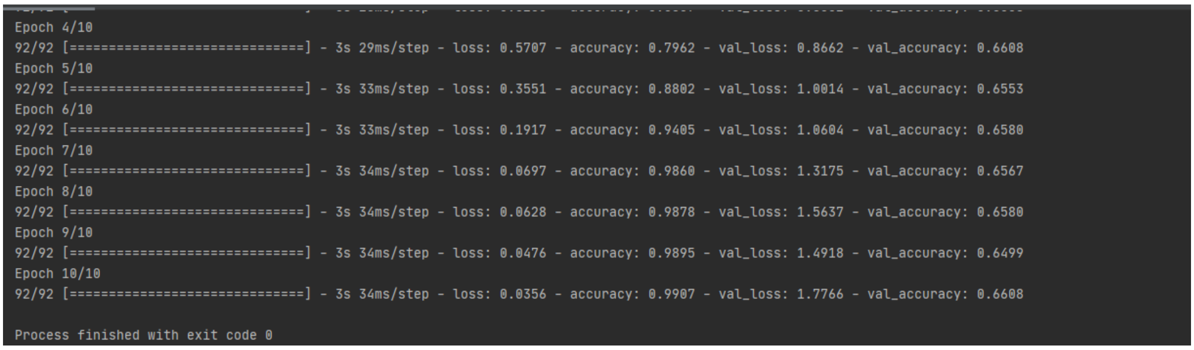


模型的详细数据；



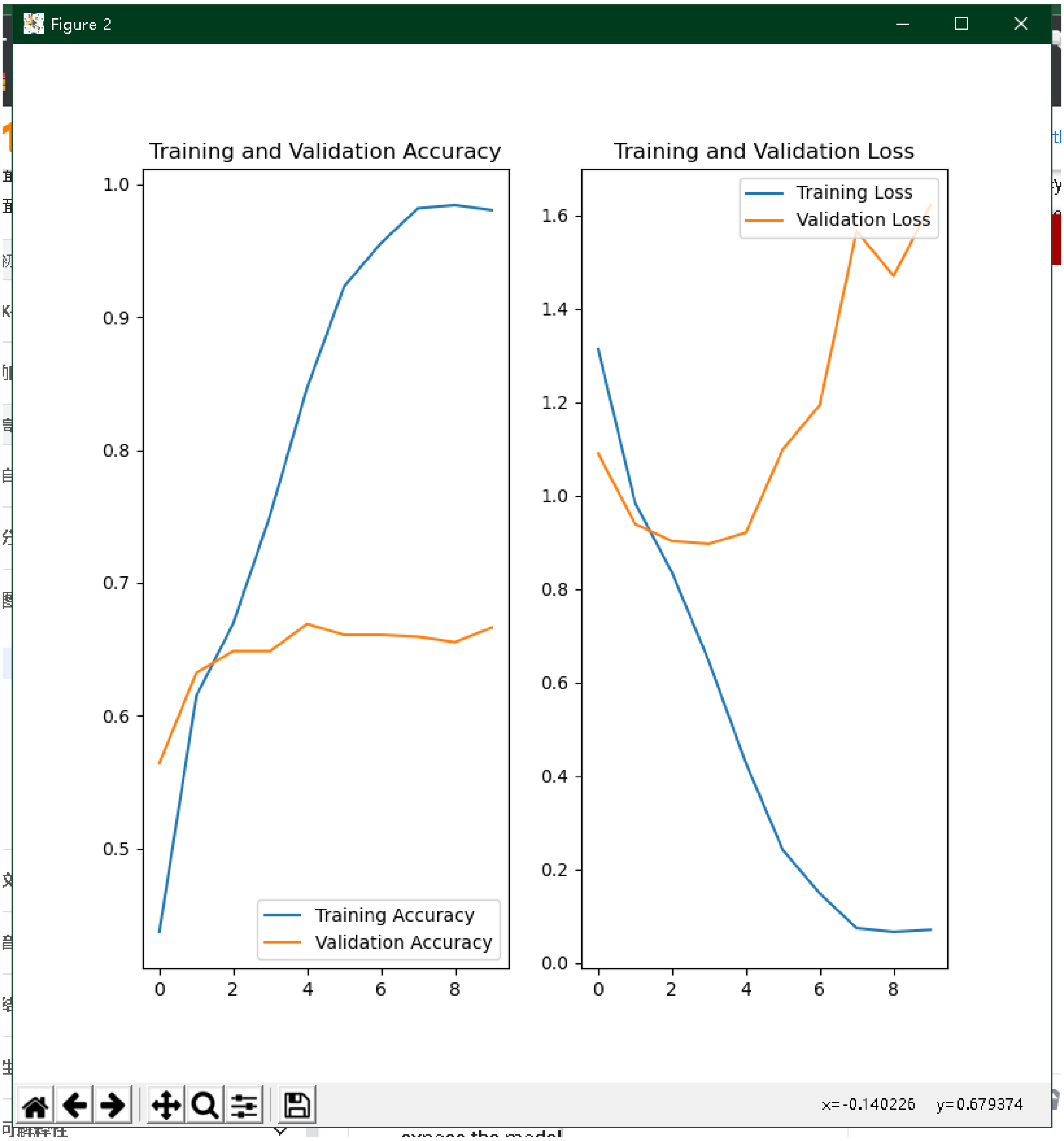
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 64)        0
```

训练结果：



```
Epoch 4/10
92/92 [==============================] - 3s 29ms/step - loss: 0.5707 - accuracy: 0.7962 - val_loss: 0.8662 - val_accuracy: 0.6608
Epoch 5/10
92/92 [==============================] - 3s 33ms/step - loss: 0.3551 - accuracy: 0.8802 - val_loss: 1.0014 - val_accuracy: 0.6553
Epoch 6/10
92/92 [==============================] - 3s 33ms/step - loss: 0.1917 - accuracy: 0.9405 - val_loss: 1.0604 - val_accuracy: 0.6580
Epoch 7/10
92/92 [==============================] - 3s 34ms/step - loss: 0.0697 - accuracy: 0.9860 - val_loss: 1.3175 - val_accuracy: 0.6567
Epoch 8/10
92/92 [==============================] - 3s 34ms/step - loss: 0.0628 - accuracy: 0.9878 - val_loss: 1.5637 - val_accuracy: 0.6580
Epoch 9/10
92/92 [==============================] - 3s 34ms/step - loss: 0.0476 - accuracy: 0.9895 - val_loss: 1.4918 - val_accuracy: 0.6499
Epoch 10/10
92/92 [==============================] - 3s 34ms/step - loss: 0.0356 - accuracy: 0.9907 - val_loss: 1.7766 - val_accuracy: 0.6608

Process finished with exit code 0
```

可视化训练结果：



分析原因： 过拟合，图片数量太少，在优化里面使用正则化和数据增强实现，见另外