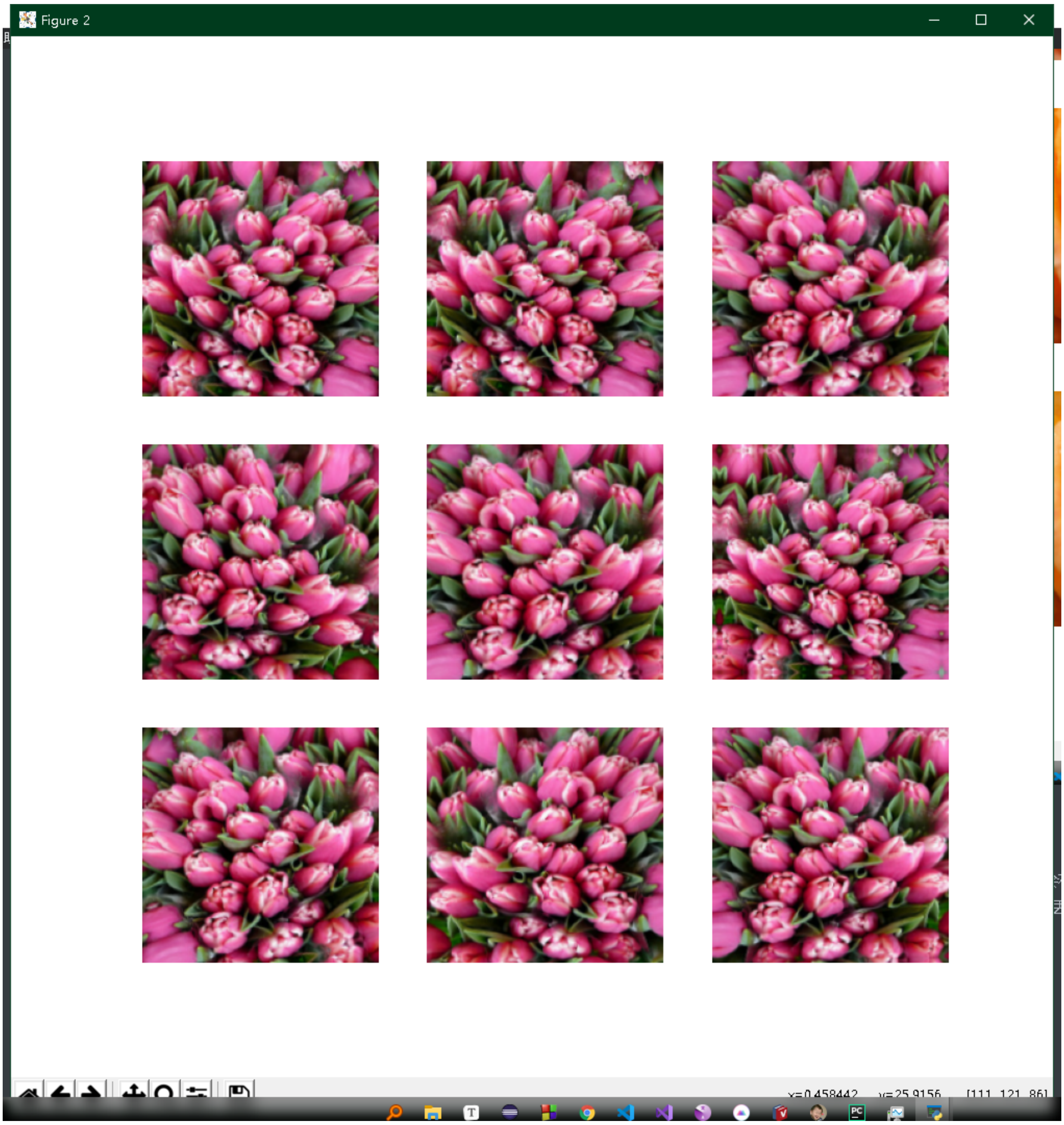
1. 优化

1.1 数据增强

```
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal",
                                                     input_shape=(img_height,
                                                                  img_width,
                                                                  3)),
        layers.experimental.preprocessing.RandomRotation(0.1),
        layers.experimental.preprocessing.RandomZoom(0.1),
    ]
)
```

结果展现:


Figure 2



1.2 Dropout

另一种减少过度拟合的技术是将 Dropout 引入网络，这是一种正则化形式。将 Dropout 应用于一个层时，它会在训练过程中从该层中随机删除（通过将激活设置为零）许多输出单元。 Dropout 将一个小数作为其输入值，形式为 0.1、0.2、0.4 等。这意味着从应用层中随机丢弃 10%、20% 或 40% 的输出单元。使layers.Dropout 创建一个新的神经网络，然后使用增强图像训练它。

1.3 具体代码

```
# 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
```

```python
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import os
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
# 1. Download and explore the dataset

import pathlib

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)

```

```python
# 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))

# 注意扩充 ： 数据增强 + 正则化
data_augmentation = keras.Sequential(
```

```python
    [
        layers.experimental.preprocessing.RandomFlip("horizontal",
                                                     input_shape=(img_height,
                                                                  img_width,
                                                                  3)),
        layers.experimental.preprocessing.RandomRotation(0.1),
        layers.experimental.preprocessing.RandomZoom(0.1),
    ]
)

plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
# 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([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  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(),
  # 正则化
  layers.Dropout(0.2),
  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 = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
# # 3.4 可视化结果
```

```python
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()
```

使用gpu后的训练结果：

```
Epoch 8/15
92/92 [==============================] - 3s 36ms/step - loss: 0.7087 - accuracy: 0.7378 - val_loss: 0.7910 - val_accuracy: 0.7139
Epoch 9/15
92/92 [==============================] - 4s 39ms/step - loss: 0.6690 - accuracy: 0.7444 - val_loss: 0.7738 - val_accuracy: 0.7030
Epoch 10/15
92/92 [==============================] - 6s 66ms/step - loss: 0.6335 - accuracy: 0.7485 - val_loss: 0.8200 - val_accuracy: 0.6798
Epoch 11/15
92/92 [==============================] - 12s 126ms/step - loss: 0.6219 - accuracy: 0.7609 - val_loss: 0.7185 - val_accuracy: 0.7153
Epoch 12/15
92/92 [==============================] - 23s 247ms/step - loss: 0.6060 - accuracy: 0.7708 - val_loss: 0.7789 - val_accuracy: 0.7084
Epoch 13/15
92/92 [==============================] - 23s 254ms/step - loss: 0.5567 - accuracy: 0.7978 - val_loss: 0.7257 - val_accuracy: 0.7262
Epoch 14/15
92/92 [==============================] - 23s 254ms/step - loss: 0.5589 - accuracy: 0.7861 - val_loss: 0.7313 - val_accuracy: 0.7139
Epoch 15/15
92/92 [==============================] - 23s 254ms/step - loss: 0.5173 - accuracy: 0.8082 - val_loss: 0.7252 - val_accuracy: 0.7398
```

可视化的结果：

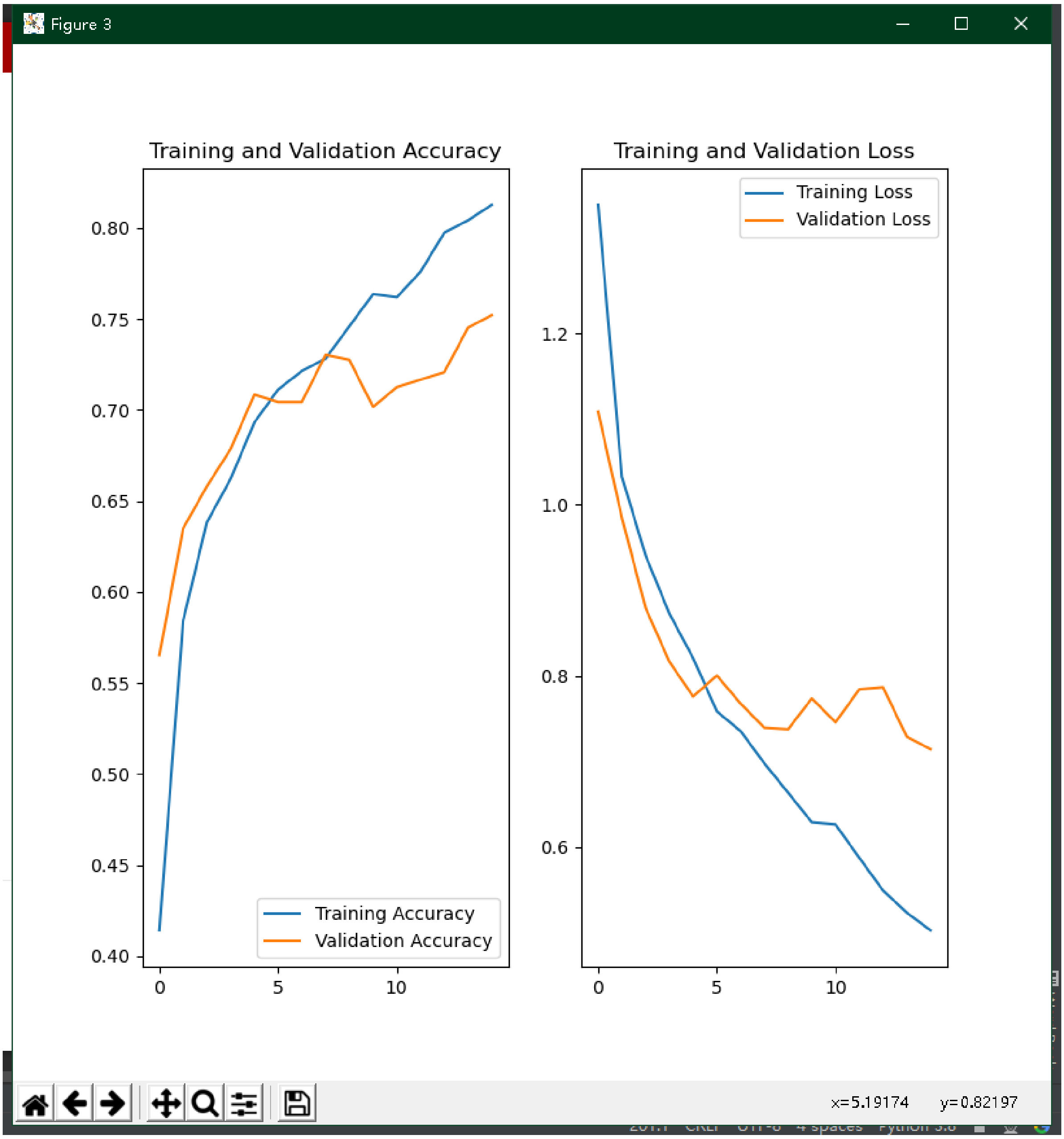


Figure 3
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Training and Validation Loss
Training Loss
Validation Loss
x=5.19174 y=0.82197