Project2 Report

Wenbo Du

Readme

The libraries and their corresponding version we introduced are:

keras 2.8.0
tensorflow 2.8.0
Pillow 9.1.0
numpy 1.20.3
pandas 1.4.2

The information about test data is in the file named flowers_test.csv, and notice that each label is preceded by a space, which means the string is not 'label' but ' label'.
All input images are in the folder named 'flowers'.
The backbone model that we used is Mobilenet.
The code named proj2.py is used to train the model.
The code named proj2.test.py is used to test the model performance.

Method

The algorithm we proposed contains three parts: pre-processing module, processing module and post-processing module.

a. Pre-processing module

First, we use tf.keras.preprocessing.image.ImageDataGenerator to import the training data, and in order to enlarge the data set and improve the robustness of the model, we need do some pre-processing to images.
Besides, we also do the normalization and split the data set into training set and validation set.

```
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1. / 255,
                                                                rotation_range=10,
                                                                width_shift_range=0.1,
                                                                height_shift_range=0.1,
                                                                shear_range=0.1,
                                                                zoom_range=0.1,
                                                                horizontal_flip=False,
                                                                fill_mode='nearest',
                                                                validation_split=0.1
```

Secondly, we can obtain the mapping relationship between label and one-hot code.

```
print(train_generator.class_indices)
```

And we use flow_from_directory to import those data.

```
train_generator = train_datagen.flow_from_directory(directory="flowers",
                                                    target_size=(img_size, img_size),
                                                    batch_size=32,
                                                    subset='training',
                                                    seed=1,
                                                    class_mode='categorical',
                                                    color_mode='rgb',
                                                    )
validation_generator = train_datagen.flow_from_directory(directory="flowers",
                                                         target_size=(img_size, img_size),
                                                         batch_size=32,
                                                         subset='validation',
                                                         seed=1,
                                                         class_mode='categorical',
                                                         color_mode='rgb',
                                                         )
```

Thirdly, we import the test images which are in the file named flowers_test according to the information in flowers_test.csv.

```
# Load test images
test_images, test_labels = get_images_labels(test_data, train_generator.class_indices, img_size)
```

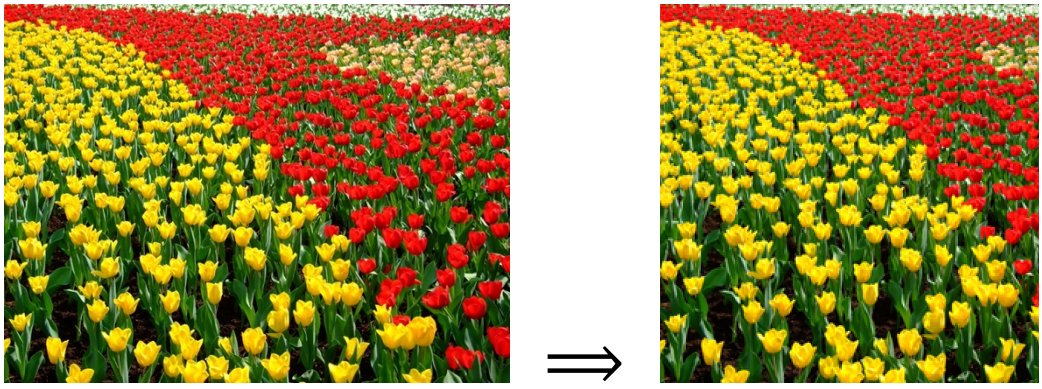
Then, we call the function named get_images_labels to process test images.

```
def get_images_labels(df, classes, img_size):
```

We calculate the size of the image, and if it is not 256X256, then we need to resize it. Besides, if the image is not square, we also need to crop the image.

```
frame = Image.open(df[i][0]).convert('RGB')
height, width = Image.open(df[i][0]).size
print(height, width)
if height != img_size or width != img_size:
    # frame = frame.resize((img_size, img_size), Image.ANTIALIAS)
    factor = img_size / min(height, width)
    frame = frame.resize((int(height * factor), int(width * factor)), Image.ANTIALIAS)
    print(frame.size)
    frame = frame.crop((0, 0, img_size, img_size))
    print(frame.size)
    frame = frame.resize((img_size, img_size), Image.ANTIALIAS)
    print(frame.size)
```

One example of resize and crop the image is shown below:



Next, we convert the image into array and do the normalization.

```
frame = np.array(frame) / 255  # Normalization
```

And we convert the label into one-hot code.

```
Y_test_temp = np.zeros(len(classes), dtype='float64')
if df[i][1][0] == ' ':
    p = classes.get(df[i][1][1:])
else:
    p = classes.get(df[i][1])
Y_test_temp[p] = 1
```

b. Processing module

The backbone model we choose is MobileNet.
First, we download the network and pre-trained weight.

```
# Load pre-trained backbone model
model = tf.keras.applications.MobileNet(
    input_shape=(img_size, img_size, 3),
    alpha=0.25,
    depth_multiplier=1,
    dropout=0.5,
    include_top=False,
    weights='imagenet',
    input_tensor=None,
    pooling=None,
    classes=1000,
    classifier_activation='softmax'
)
```

Then, we need to adjust the input size and the number of classes.

```
# Define the input size
inputs = tf.keras.layers.Input(shape=(img_size, img_size, 3))
x = model(inputs)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
# Change the number of classes from 1000 to 13
outputs = tf.keras.layers.Dense(13, activation='softmax', use_bias=True, name='Logits')(x)
model = tf.keras.models.Model(inputs=inputs, outputs=outputs)
```

After that, we define the optimization method, learning rate and loss function.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
lr = backend.get_value(model.optimizer.lr)
# print('lr is: ', lr)
backend.set_value(model.optimizer.lr, 0.0002)
```

Finally, we can start to train the model.

```
# Train the model
for i in range(100):
    test_images_copy = copy.deepcopy(test_images)
    test_labels_copy = copy.deepcopy(test_labels)
    checkpointer = keras.callbacks.ModelCheckpoint(filepath='my_model.h5', verbose=1, save_weights_only=False, save_freq=1)
    #model.fit_generator(train_generator, validation_data=validation_generator, steps_per_epoch=train_generator.samples/train_generator.b
    model.fit_generator(train_generator, steps_per_epoch=train_generator.samples / train_generator.batch_size, epochs=1, callbacks=None)
    loss, acc = model.evaluate(test_images_copy, test_labels_copy, batch_size=32, verbose=2)
    loss_0, acc_0 = model.evaluate_generator(validation_generator, verbose=2)
    print('Test model, accuracy: {:5.5f}%'.format(100 * acc))
    for j in range(len(test_labels)):
        X_test = np.zeros((1, img_size, img_size, 3), dtype='float32')
        X_test[0] = test_images_copy[j]
        print(model.predict(X_test))
        print(np.argmax(model.predict(X_test)))
        # im = Image.fromarray(255 * X_test[0])
        # im.show()
    # print(model.predict(test_images_copy))
    model.save('my_model.h5')
```

After each epoch, the model will output the training accuracy, validation accuracy and test accuracy.

c. Post-processing module

The code named proj2.test.py is used to test the model performance.

First, we load the weight from the file named “my_model.h5”.

```
# Load trained model
my_model = load_model(model)
```

Then we import the test data and predict results.

```
test_images_copy = copy.deepcopy(test_images)
test_labels_copy = copy.deepcopy(test_labels)
loss, acc = my_model.evaluate(test_images_copy, test_labels_copy, batch_size=32, verbose=2)
for i in range(len(test_labels)):
    X_test = np.zeros((1, img_size, img_size, 3), dtype='float64')
    X_test[0] = test_images_copy[i]
    # im = Image.fromarray(255 * X_test[0])  # numpy 转 image类
    # im.show()
    print(my_model.predict(X_test))
    print(np.argmax(my_model.predict(X_test)))
# print(my_model.predict(test_images_copy))
print('Test model, accuracy: {:5.5f}%'.format(100 * acc))
```

Experiment Results

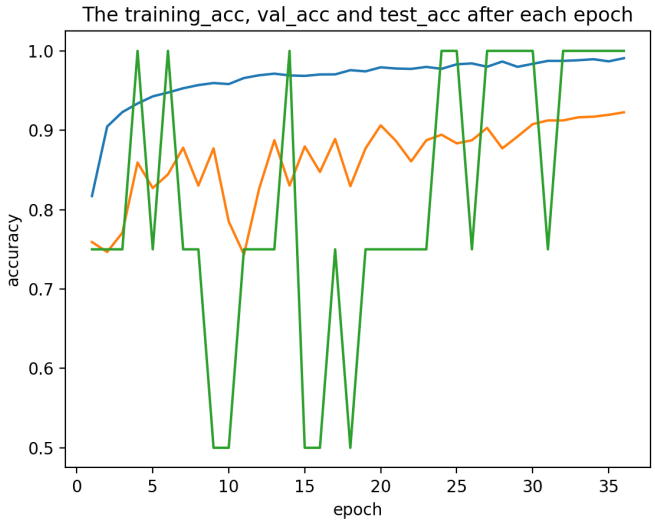
After load the trained model, the test results are:

```
common daisy
tulip
tulip
rose
Test model, accuracy: 100.00000%
```

The training accuracy, validation accuracy and test accuracy after each epoch are shown below:

train_acc = [0.8172, 0.9047, 0.9229, 0.9337, 0.9427, 0.9475, 0.9529, 0.9569, 0.9595,

0.9582, 0.9658, 0.9693, 0.9713, 0.9691, 0.9685, 0.9703, 0.9704, 0.9757, 0.9742, 0.9793, 0.9778, 0.9773, 0.9798, 0.9774, 0.9831, 0.9842, 0.9800, 0.9866, 0.9799, 0.9837, 0.9874, 0.9874, 0.9882, 0.9895, 0.9868, 0.9908]
validation_acc = [0.7592, 0.7467, 0.7709, 0.8593, 0.8272, 0.8444, 0.8780, 0.8303, 0.8772, 0.7850, 0.7435, 0.8264, 0.8874, 0.8303, 0.8796, 0.8475, 0.8890, 0.8296, 0.8772, 0.9062, 0.8866, 0.8608, 0.8874, 0.8944, 0.8835, 0.8874, 0.9030, 0.8772, 0.8921, 0.9077, 0.9124, 0.9124, 0.9163, 0.9171, 0.9195, 0.9226]
test_acc = [0.75, 0.75, 0.75, 1.00, 0.75, 1.00, 0.75, 0.75, 0.50, 0.50, 0.75, 0.75, 0.75, 1.00, 0.50, 0.50, 0.75, 0.50, 0.75, 0.75, 0.75, 0.75, 0.75, 1.00, 1.00, 0.75, 1.00, 1.00, 1.00, 1.00, 1.00, 0.75, 1.00, 1.00, 1.00, 1.00, 1.00]



The blue line is the train_acc , the orange line is the validation_acc and the green line is the test_acc.
In the final, the training accuracy is 0.9908, the validation accuracy is 0.9226 and the test accuracy is 1.00.