```
#4 pretrained model and transfer learning
```

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import PIL.Image
import tensorflow as tf
import tensorflow_datasets as tfds
import pathlib
```

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

```
batch_size = 32
img_height = 180
img_width = 180

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

validation_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    shuffle=True,
    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 2936 files for training.
Found 3670 files belonging to 5 classes.
Using 734 files for validation.
```

```
class_names = train_ds.class_names

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")
```


```
val_batches = tf.data.experimental.cardinality(validation_ds)
test_dataset = validation_ds.take(val_batches // 5)
validation_dataset = validation_ds.skip(val_batches // 5)

print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_dataset))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))
```

```
Number of validation batches: 19
Number of test batches: 4
```

```
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_ds.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2),
])


for image, _ in train_dataset.take(1):
    plt.figure(figsize=(10, 10))
    first_image = image[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
        plt.imshow(augmented_image[0] / 255)
        plt.axis('off')
```

```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
rescale = tf.keras.layers.Rescaling(1./127.5, offset=-1)
```

```
IMG_SIZE = (180, 180)
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE, include_top=False, weights='imagenet')
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_2
9406464/9406464 [==============================] - 0s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 6, 6, 1280)
```

```
base_model.trainable = False
```

```
base_model.summary()
```

```
Model: "mobilenetv2_1.00_224"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 180, 180, 3  0           []
                                )]

 Conv1 (Conv2D)                 (None, 90, 90, 32)   864         ['input_1[0][0]']

 bn_Conv1 (BatchNormalization)  (None, 90, 90, 32)   128         ['Conv1[0][0]']

 Conv1_relu (ReLU)              (None, 90, 90, 32)   0           ['bn_Conv1[0][0]']

 expanded_conv_depthwise (Depth  (None, 90, 90, 32)  288         ['Conv1_relu[0][0]']
 wiseConv2D)

 expanded_conv_depthwise_BN (Ba  (None, 90, 90, 32)  128         ['expanded_conv_depthwise[0][0]']
 tchNormalization)

 expanded_conv_depthwise_relu (  (None, 90, 90, 32)  0           ['expanded_conv_depthwise_BN[0][0
 ReLU)                                                           ]']

 expanded_conv_project (Conv2D)  (None, 90, 90, 16)  512         ['expanded_conv_depthwise_relu[0]
                                                                 [0]']

 expanded_conv_project_BN (Batc  (None, 90, 90, 16)  64          ['expanded_conv_project[0][0]']
 hNormalization)

 block_1_expand (Conv2D)        (None, 90, 90, 96)   1536        ['expanded_conv_project_BN[0][0]'
                                                                 ]

 block_1_expand_BN (BatchNormal  (None, 90, 90, 96)  384         ['block_1_expand[0][0]']
 ization)

 block_1_expand_relu (ReLU)     (None, 90, 90, 96)   0           ['block_1_expand_BN[0][0]']

 block_1_pad (ZeroPadding2D)    (None, 91, 91, 96)   0           ['block_1_expand_relu[0][0]']

 block_1_depthwise (DepthwiseCo  (None, 45, 45, 96)  864         ['block_1_pad[0][0]']
 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 45, 45, 96)  384         ['block_1_depthwise[0][0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 45, 45, 96)   0           ['block_1_depthwise_BN[0][0]']

 block_1_project (Conv2D)       (None, 45, 45, 24)   2304        ['block_1_depthwise_relu[0][0]']

 block_1_project_BN (BatchNorma  (None, 45, 45, 24)  96          ['block_1_project[0][0]']
 lization)

 block_2_expand (Conv2D)        (None, 45, 45, 144)  3456        ['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNormal  (None, 45, 45, 144)  576        ['block_2_expand[0][0]']
 ization)

 block_2_expand_relu (ReLU)     (None, 45, 45, 144)  0           ['block_2_expand_BN[0][0]']
```

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

```
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 1)
```

```
inputs = tf.keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
model.summary()
len(model.trainable_variables)
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 180, 180, 3)]     0         
                                                                 
 sequential (Sequential)     (None, None, None, 3)     0         
                                                                 
 tf.math.truediv_1 (TFOpLamb  (None, 180, 180, 3)      0         
 da)                                                             
                                                                 
 tf.math.subtract_1 (TFOpLam  (None, 180, 180, 3)      0         
 bda)                                                            
                                                                 
 mobilenetv2_1.00_224 (Funct  (None, 6, 6, 1280)       2257984   
 ional)                                                          
                                                                 
 global_average_pooling2d (G  (None, 1280)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout_1 (Dropout)         (None, 1280)              0         
                                                                 
 dense (Dense)               (None, 1)                 1281      
                                                                 
=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
2
```

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
19/19 [==============================] - 18s 784ms/step - loss: 2.0007 - accuracy: 0.1716
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 2.00
initial accuracy: 0.17
```

```
history = model.fit(train_dataset, epochs=initial_epochs, validation_data=validation_dataset)
```

```
Epoch 1/10
92/92 [==============================] - 101s 1s/step - loss: -0.8759 - accuracy: 0.2016 - val_loss: -3.9479 - val_accuracy: 0.2442
Epoch 2/10
```

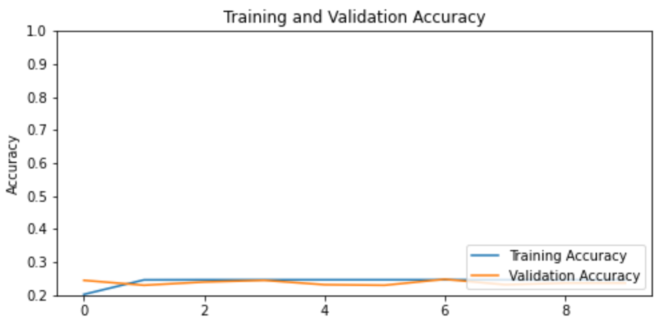
```
92/92 [==============================] - 96s 1s/step - loss: -6.0849 - accuracy: 0.2456 - val_loss: -8.2835 - val_accuracy: 0.2294
Epoch 3/10
92/92 [==============================] - 95s 1s/step - loss: -10.8943 - accuracy: 0.2459 - val_loss: -12.8895 - val_accuracy: 0.2393
Epoch 4/10
92/92 [==============================] - 95s 1s/step - loss: -15.8356 - accuracy: 0.2459 - val_loss: -17.5517 - val_accuracy: 0.2442
Epoch 5/10
92/92 [==============================] - 95s 1s/step - loss: -20.8170 - accuracy: 0.2459 - val_loss: -22.4271 - val_accuracy: 0.2310
Epoch 6/10
92/92 [==============================] - 95s 1s/step - loss: -26.0011 - accuracy: 0.2459 - val_loss: -27.9309 - val_accuracy: 0.2294
Epoch 7/10
92/92 [==============================] - 100s 1s/step - loss: -31.2669 - accuracy: 0.2459 - val_loss: -31.7254 - val_accuracy: 0.2475
Epoch 8/10
92/92 [==============================] - 101s 1s/step - loss: -36.3794 - accuracy: 0.2459 - val_loss: -37.5526 - val_accuracy: 0.2310
Epoch 9/10
92/92 [==============================] - 101s 1s/step - loss: -41.7960 - accuracy: 0.2459 - val_loss: -42.4957 - val_accuracy: 0.2360
Epoch 10/10
92/92 [==============================] - 109s 1s/step - loss: -46.9059 - accuracy: 0.2459 - val_loss: -47.5789 - val_accuracy: 0.2360
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')
```

```
Text(0.5, 1.0, 'Training and Validation Accuracy')
```




```
base_model.trainable = True

print("Number of layers in the base model: ", len(base_model.layers))
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

```
Number of layers in the base model:  154
```

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                            optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
                            metrics=['accuracy'])
model.summary()
len(model.trainable_variables)
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 180, 180, 3)]     0

 sequential (Sequential)     (None, None, None, 3)     0

 tf.math.truediv_1 (TFOpLamb  (None, 180, 180, 3)      0
 da)

 tf.math.subtract_1 (TFOpLam  (None, 180, 180, 3)      0
 bda)
```

```
 mobilenetv2_1.00_224 (Funct  (None, 6, 6, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,862,721
Non-trainable params: 396,544
_________________________________________________________________
56
```

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
```

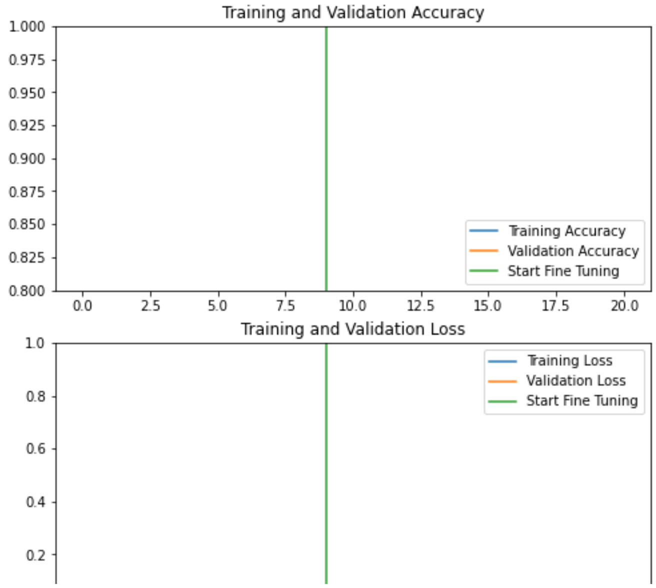
```
Epoch 10/20
92/92 [==============================] - 158s 2s/step - loss: -429.7906 - accuracy: 0.2459 - val_loss: -599.4150 - val_accuracy: 0.2310
Epoch 11/20
92/92 [==============================] - 150s 2s/step - loss: -606.5585 - accuracy: 0.2459 - val_loss: -644.9821 - val_accuracy: 0.2211
Epoch 12/20
92/92 [==============================] - 149s 2s/step - loss: -643.5132 - accuracy: 0.2459 - val_loss: -659.9099 - val_accuracy: 0.2409
Epoch 13/20
92/92 [==============================] - 148s 2s/step - loss: -664.0253 - accuracy: 0.2459 - val_loss: -677.9578 - val_accuracy: 0.2360
Epoch 14/20
92/92 [==============================] - 149s 2s/step - loss: -679.9046 - accuracy: 0.2493 - val_loss: -678.7469 - val_accuracy: 0.2541
Epoch 15/20
92/92 [==============================] - 149s 2s/step - loss: -690.2488 - accuracy: 0.2599 - val_loss: -702.7096 - val_accuracy: 0.2607
Epoch 16/20
92/92 [==============================] - 149s 2s/step - loss: -702.9858 - accuracy: 0.2701 - val_loss: -702.5030 - val_accuracy: 0.2574
Epoch 17/20
92/92 [==============================] - 149s 2s/step - loss: -711.9473 - accuracy: 0.2813 - val_loss: -736.5058 - val_accuracy: 0.2591
Epoch 18/20
92/92 [==============================] - 150s 2s/step - loss: -721.7292 - accuracy: 0.2871 - val_loss: -745.2521 - val_accuracy: 0.2739
Epoch 19/20
92/92 [==============================] - 151s 2s/step - loss: -731.8317 - accuracy: 0.2980 - val_loss: -743.1838 - val_accuracy: 0.2723
Epoch 20/20
92/92 [==============================] - 150s 2s/step - loss: -741.9691 - accuracy: 0.3028 - val_loss: -739.0104 - val_accuracy: 0.2888
```

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

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


plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```




```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
4/4 [==============================] - 3s 725ms/step - loss: -805.6907 - accuracy: 0.3125
Test accuracy : 0.3125
```

5 Analysis

On the full connection, CNN adds a convolution kernel to extract features, and then classifies the extracted features through the full connection, and selects the item with the highest score as the result of object recognition. From the analysis of the results, the results show an upward trend with the increase of epoch, indicating that we can set the epoch to be larger, and the obtained model will be more accurate. The shortcomings of CNN are obvious. First, there is only one item in the photo. CNN does not divide the photo. Secondly, the increase in the number of network layers will cause the gradient to disappear, making the entire network redundant.

The pre-trained model transfers the trained model parameters to the new model to help the new model training. The results from the pretrained model I was using were not very good. The first is that the new data set is too different from the original data set to perform poorly. The original data set I used was to judge cats and dogs, while the new data set was five different types of flowers. So it's not very ideal. One should try to freeze the initial layers (k layers) of the pretrained model and train the remaining (n-k) layers again. The similarity of the new dataset is low, so the predictions made with the pretrained model will not be very effective. It is therefore important to retrain higher layers on new datasets