电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

实验报告

Experimental Report

报告题目 机器视觉作业 2

学　　院 机械与电气工程学院

专　　业 机器人工程

学　　号 2021040902007

作者姓名 经彭宇

指导教师 王国泰

目 录

第一章 迁移学习

1.1 总体思路

使用 TensorFlow 进行图像分类的迁移学习是一种有效的方法，能够构建高性能的模型。首先，需要准备数据集。使用了一个包含猫和狗图像的数据集。这个数据集经过预处理，包括将图像调整为相同的尺寸（160x160 像素），并分为训练集和验证集。

接下来，我们使用 TensorFlow 中的 ImageDataGenerator 来进行数据增强，通过对训练图像进行随机变换来生成更多的训练样本。这有助于模型学习到更多的特征，并提高泛化能力。

接着，构建迁移学习模型。在这里使用了 MobileNet V2 作为基础模型，并在其顶部添加了全局平均池化层和密集层。通过迁移学习，可以利用 MobileNet V2 在 ImageNet 数据集上学到的特征来帮助我们的模型更好地理解图像。

然后，我们编译模型并进行训练。我们使用了 Adam 优化器和二元交叉熵损失函数，并监控模型的准确率。在初始训练之后，对模型进行了微调，通过解冻部分层并使用更小的学习率来进一步优化模型。

最后，评估模型的性能并进行预测。通过绘制训练和验证的准确率和损失曲线来分析模型的学习情况，并在测试集上进行评估来检验模型的泛化能力。同时，需要展示模型在测试集上的预测结果，以便更直观地了解模型的表现。

1.2 实现方法

（1）数据预处理：

下载数据集并解压，然后使用 tf.keras.utils.image_dataset_from_directory 效用函数创建一个 tf.data.Dataset 进行训练和验证。

```
16 train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
17                                                             shuffle=True,
18                                                             batch_size=BATCH_SIZE,
19                                                             image_size=IMG_SIZE)
20
```

图 1-1 创建 Dataset

为确保数据的准确性，显示训练集中的前九个图像和标签。如图 1-2 所示。

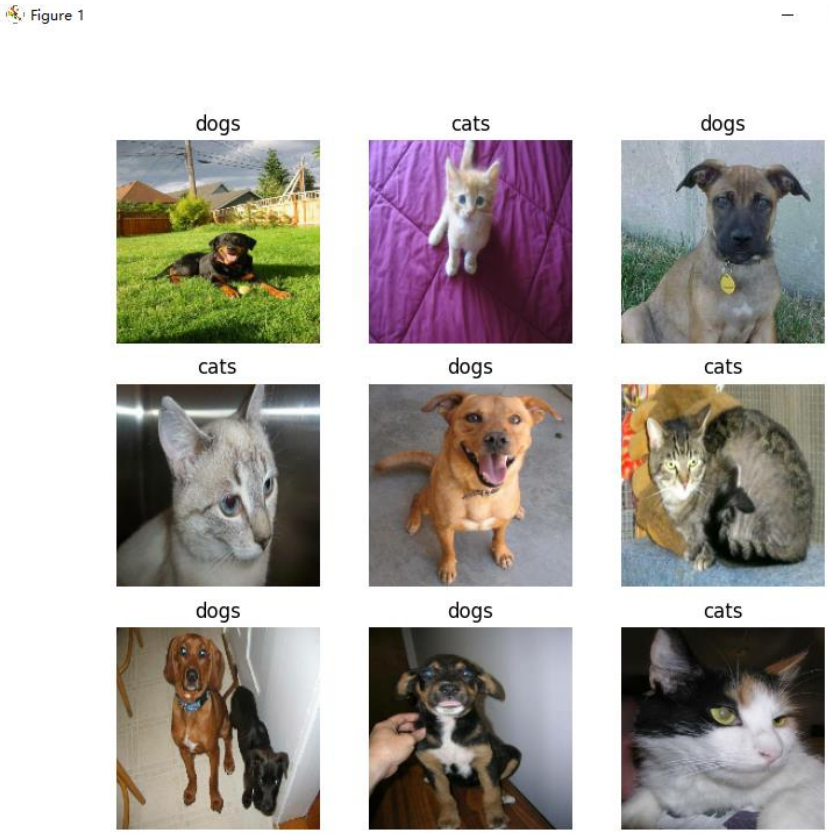

图 1-2 前九个图像和标签

但由于原始数据集不包含测试集，需要创建一个，使用 tf.data.experimental.cardinality 确定验证集中有多少批次的数据，然后将其中的 20% 移至测试集。

```
35 val_batches = tf.data.experimental.cardinality(validation_dataset)
36 test_dataset = validation_dataset.take(val_batches // 5)
37 validation_dataset = validation_dataset.skip(val_batches // 5)
38
39 print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_dataset))
40 print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))
```

图 1-3 创建测试集

```
Number of validation batches: 26
Number of test batches: 6
```

图 1-4 训练集和测试集的数量

因为没有较大的图像数据集，随意将随机但现实的转换应用于训练图像（例如旋转或水平翻转）来人为引入样本多样性。这有助于使模型暴露于训练数据的不同方面并减少过拟合，如图 1-5 所示。

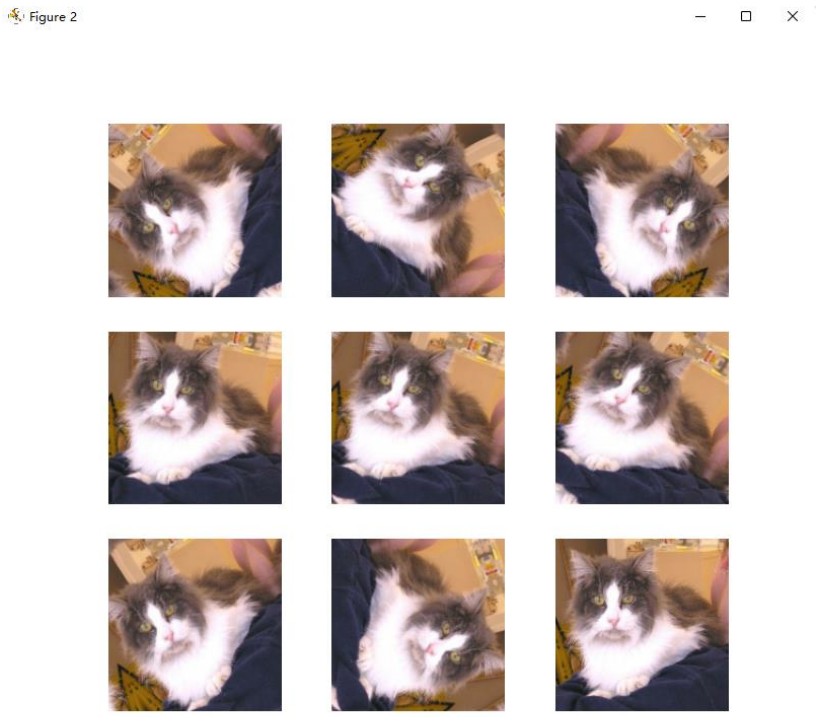

图 1-5 数据扩充

最后需要重新缩放像素值，因为下载的模型期望像素值处于 [-1, 1] 范围内，但此时，图像中的像素值处于 [0, 255] 范围内，需要重新缩放这些像素值。

```
62  preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
63
64  rescale = tf.keras.layers.Rescaling(1./127.5, offset=-1)
```

图 1-6 缩放像素值

（2）创建模型

根据 Google 开发的 **MobileNet V2** 模型来创建基础模型，包含全局平均池化层和密集预测层。此模型已基于 ImageNet 数据集进行预训练，ImageNet 数据集是一个包含 140 万个图像和 1000 个类的大型数据集。ImageNet 是一个研究训练数据集，具有各种各样的类别，例如 jackfruit 和 syringe。

冻结在上一步中创建的卷积基，并用作特征提取程序，冻结可避免在训练期间更新给定层中的权重。MobileNet V2 具有许多层，因此将整个模型的 trainable 标记设置为 False 会冻结所有这些层。

```
76  base_model.trainable = False
```

图 1-7 冻结卷积层

接下来需要添加分类头，使用 tf.keras.layers.GlobalAveragePooling2D 层在 5x5 空间位置内取平均值，从特征块生成预测，以将特征转换成每个图像一个向量（包含 1280 个元素）。应用 tf.keras.layers.Dense 层将这些特征转换成每个图像一个预测。因为此预测将被视为 logit 或原始预测值，所以在此处不需要激活函数。正数预测 1 类，负数预测 0 类。通过使用 Keras 函数式 API 将数据扩充、重新缩放、base_model 和特征提取程序层链接在一起来构建模型。

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

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

inputs = tf.keras.Input(shape=(160, 160, 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)
```

图 1-8 添加分类头

（3）训练模型

在训练模型前，需要先编译模型。由于存在两个类，并且模型提供线性输出，所以将 tf.keras.losses.BinaryCrossentropy 损失与 from_logits=True 结合使用。

Model: "functional_2"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 160, 160, 3)	0
sequential (Sequential)	(None, 160, 160, 3)	0
true_divide (TrueDivide)	(None, 160, 160, 3)	0
subtract (Subtract)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functional)	(None, 5, 5, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1,281

Total params: 2,259,265 (8.62 MB)
Trainable params: 1,281 (5.00 KB)
Non-trainable params: 2,257,984 (8.61 MB)

图 1-9 编译完的模型

在训练数据集上进行初始 epoch 训练，并在验证数据集上进行验证。经过 10 个周期的训练后，在验证集上看到约 92% 的准确率。

```
initial loss: 0.90
initial accuracy: 0.41
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.8978 - loss: 0.2379 - val_accuracy: 0.9443 - val_loss: 0.1336
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.9000 - loss: 0.2296 - val_accuracy: 0.9455 - val_loss: 0.1273
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9092 - loss: 0.2062 - val_accuracy: 0.9505 - val_loss: 0.1286
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.9104 - loss: 0.2081 - val_accuracy: 0.9530 - val_loss: 0.1159
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9116 - loss: 0.2058 - val_accuracy: 0.9554 - val_loss: 0.1119
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9129 - loss: 0.2041 - val_accuracy: 0.9592 - val_loss: 0.1005
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9136 - loss: 0.1824 - val_accuracy: 0.9592 - val_loss: 0.1002
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 161ms/step - accuracy: 0.9207 - loss: 0.1708 - val_accuracy: 0.9653 - val_loss: 0.1008
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9317 - loss: 0.1722 - val_accuracy: 0.9666 - val_loss: 0.0922
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.9228 - loss: 0.1746 - val_accuracy: 0.9703 - val_loss: 0.0892
```

图 1-10 模型训练

查看使用 MobileNet V2 基础模型作为固定特征提取程序时训练和验证准确率/损失的学习曲线。

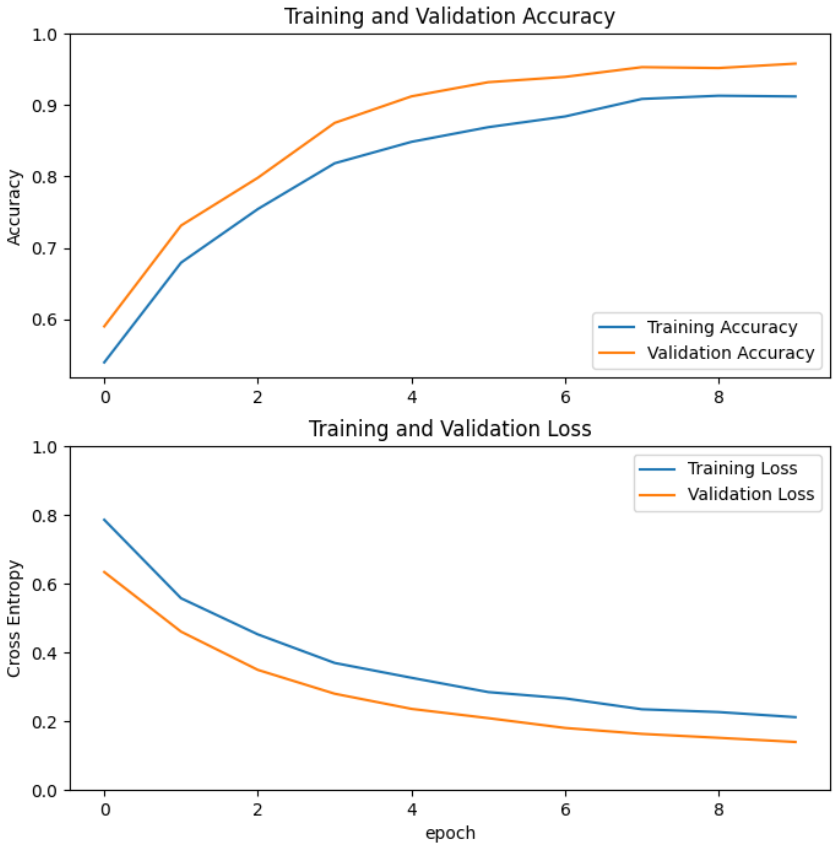

图 1-11 学习曲线

tf.keras.layers.BatchNormalization 和 tf.keras.layers.Dropout 等层会影响训练期间的准确率。在计算验证损失时，它们处于关闭状态，所以验证指标明显优于训练指标。训练指标报告的是某个周期的平均值，而验证指标则在经过该周期后才进行评估，因此验证指标会看到训练时间略长一些的模型。

（4）微调

在特征提取实验中，仅在 MobileNet V2 基础模型的顶部训练了一些层。预训练网络的权重在训练过程中未更新。因此微调少量顶层进一步提高模型的性能。

解冻 base_model 并将底层设置为不可训练。重新编译模型然后恢复训练。

```
143 base_model.trainable = True
144
145 print("Number of layers in the base model: ", len(base_model.layers))
```

图 1-12 解冻顶层

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

图 1-13 base_model 中的层数

随后编译模型，继续训练模型，并绘制学习曲线

```
Model: "functional_2"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 160, 160, 3)	0
sequential (Sequential)	(None, 160, 160, 3)	0
true_divide (TrueDivide)	(None, 160, 160, 3)	0
subtract (Subtract)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functional)	(None, 5, 5, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1,281

```
Total params: 2,259,265 (8.62 MB)
Trainable params: 1,281 (5.00 KB)
Non-trainable params: 2,257,984 (8.61 MB)
```

图 1-14 解冻后编译完的模型

```
Epoch 10/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 208ms/step - accuracy: 0.8339 - loss: 0.4328 - val_accuracy: 0.9715 - val_loss: 0.0937
Epoch 11/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 209ms/step - accuracy: 0.9034 - loss: 0.2684 - val_accuracy: 0.9802 - val_loss: 0.0827
Epoch 12/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 220ms/step - accuracy: 0.8953 - loss: 0.2472 - val_accuracy: 0.9814 - val_loss: 0.0605
Epoch 13/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 209ms/step - accuracy: 0.9240 - loss: 0.1867 - val_accuracy: 0.9827 - val_loss: 0.0564
Epoch 14/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 203ms/step - accuracy: 0.9245 - loss: 0.1754 - val_accuracy: 0.9839 - val_loss: 0.0554
Epoch 15/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 217ms/step - accuracy: 0.9417 - loss: 0.1536 - val_accuracy: 0.9827 - val_loss: 0.0501
Epoch 16/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 209ms/step - accuracy: 0.9395 - loss: 0.1284 - val_accuracy: 0.9814 - val_loss: 0.0477
Epoch 17/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 217ms/step - accuracy: 0.9510 - loss: 0.1359 - val_accuracy: 0.9839 - val_loss: 0.0470
Epoch 18/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 221ms/step - accuracy: 0.9479 - loss: 0.1450 - val_accuracy: 0.9839 - val_loss: 0.0441
Epoch 19/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 224ms/step - accuracy: 0.9527 - loss: 0.1150 - val_accuracy: 0.9814 - val_loss: 0.0423
Epoch 20/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 213ms/step - accuracy: 0.9517 - loss: 0.1146 - val_accuracy: 0.9827 - val_loss: 0.0438
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 105ms/step - accuracy: 0.9658 - loss: 0.0387
```

图 1-15 训练解冻后的模型

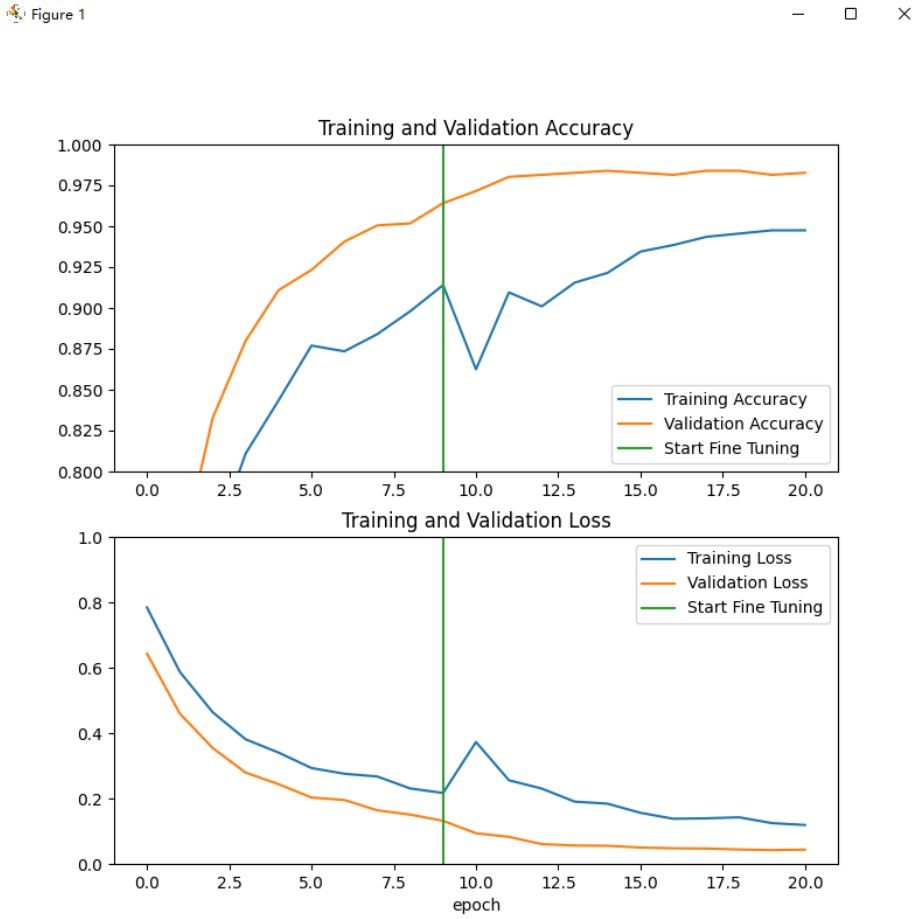

图 1-16 微调后的学习曲线

在微调 MobileNet V2 基础模型的最后几层并在这些层上训练分类器时，验证损失比训练损失高得多，因此可能存在一些过拟合。当新的训练集相对较小且与原始 MobileNet V2 数据集相似时，也可能存在一些过拟合。经过微调后，模型在验证集上的准确率几乎达到 95%。

（5）模型评估

使用测试集在新数据上验证模型的性能。

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

图 1-17 测试模型的性能

```
Test accuracy : 0.9791666865348816
```

图 1-18 模型的性能

1.3 效果展示

展示测试集中的图像网格，并显示模型预测的类别标签，以展示模型的预测结果。

```
Predictions:
 [1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0]
Labels:
 [1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0]
```

图 1-19 测试结果

1.4 思考和总结

使用预训练模型进行特征提取时，对于小型数据，常见做法是利用基于相同域中的较大数据集训练的模型所学习的特征，需要实例化预训练模型并在顶部添加一个全连接分类器。预训练模型处于“冻结状态”，训练过程中仅更新分类器的权重。在这种情况下，卷积基提取了与每个图像关联的所有特征。

为了进一步提高性能，需要通过微调将预训练模型的顶层重新用于新的数据集。在本例中，通过调整权重以使模型学习特定于数据集的高级特征。当训练数据集较大且与训练预训练模型所使用的原始数据集非常相似时，准确性达到 95%。

此模型中用了一个深度分类器，因此后面将使用一个简单的 SVM，看是否能达到同样的效果。

第二章 支持向量机

上一章通过迁移学习，我们利用预训练的 MobileNet V2 模型来构建图像分类模型，并对模型进行了微调。模型中用了一个深度分类器，本章将使用一个简单的 SVM 对特征进行提取和分类，看是否能达到同样的效果。

2.1 总体思路

从数据准备预处理、划分和增强，到构建学习模型并进行编译和训练和第一章的操作一致，需要做的是使用 SVM 模型对特征进行提取和分类，评估模型性能并绘制学习曲线。

同样需要对基础模型进行微调，并重新编译和训练模型。最后评估模型性能并进行预测，展示预测结果，并用混淆矩阵来辅助展示。

2.2 实现方法

数据预处理同上一章的操作，在特征提取的时候需要使用 SVM 进行提取和分类。

（1） SVM 特征提取

首先需要从未训练的模型 base_model 中获取 X，Y 的训练集和测试集

```
181 ##SVM
182 #冻结卷积基
183 base_model.trainable=False
184
185 #获取X,Y的训练集和测试集，注意这里获取的X以及testX因为被拉直成了一维，需要在后面进行reshape处
186 X,Y=predict(base_model,train_dataset)
187 testX,testY=predict(base_model,test_dataset)
188
189 #对X进行数据处理，获取训练集Xave&&测试集testXave
190 #由于X被拉直成一维，需要reshape还原成高维
191 X1=X.reshape(2000,5,5,1280)
192 #池化
193 Xave=global_average_layer(X1)
194
195 #同上
196 testX1=testX.reshape(192,5,5,1280)
197 testXave=global_average_layer(testX1)
```

图 2-1 获取 X，Y 的训练集和测试集

从 base_model 中获取训练集和测试集，用 predict 函数来实现此项操作，predict 函数如图 2-2 所示。

```python
def predict(model,db_test,batchz=None,verbose=True):
    '''
    自定义的model的预测方法，主要是实现配对返回预测值和真实值
    :param model:
    :param db_test:
    :param batchz:当db已经进行了batch，这里就不需要赋值
    :param verbose:
    :return:
    '''
    y_pre = np.array([])
    y_tru = np.array([])
    for elem in db_test.as_numpy_iterator():
        # 注意，这里的model要非训练模式
        batch_y_pre=model(elem[0],training=False).numpy().flatten()
        print(batch_y_pre.shape)
        print(len(batch_y_pre))
        batch_y_tru=elem[1].flatten()
        y_pre = np.insert(y_pre, len(y_pre), batch_y_pre)
        y_tru = np.insert(y_tru, len(y_tru), batch_y_tru)
    return y_pre,y_tru
```

图 2-2 predict 函数

（2） SVM 模型

创建了一个多项式核 SVM 分类器，并设置了正则化参数 C 为 1。这种设置通常适用于一般的二分类问题，并进行训练。

```python
#定义SVM模型
svm_model=SVC(kernel='poly',C=1)
svm_model.fit(Xave,Y)
```

图 2-3 SVM 模型创建并训练

评估模型性能，使用 svm_model.predict 函数实现，并通过混淆矩阵来辅助显示评估结果。

```python
#评估模型性能
y_pred = svm_model.predict(testXave)

#混淆矩阵
from sklearn.metrics import classification_report, confusion_matrix
print('Classification Report:\n', classification_report(testY, y_pred))
print('Confusion Matrix:\n', confusion_matrix(testY, y_pred))
```

图 2-4 模型评估

从图 2-5 和图 2-6 中可以看出，模型的准确率在 82%左右。

```
accuracy:
0.8229166666666666
recall:
0.8231144872490505
F1_score:
0.8228974498100923
```

图 2-5 模型评估

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      0.88      0.87       106
         1.0       0.84      0.81      0.83        86

    accuracy                           0.85       192
   macro avg       0.85      0.85      0.85       192
weighted avg       0.85      0.85      0.85       192

Confusion Matrix:
 [[93 13]
 [16 70]]
```

图 2-6 混淆矩阵

绘制学习曲线

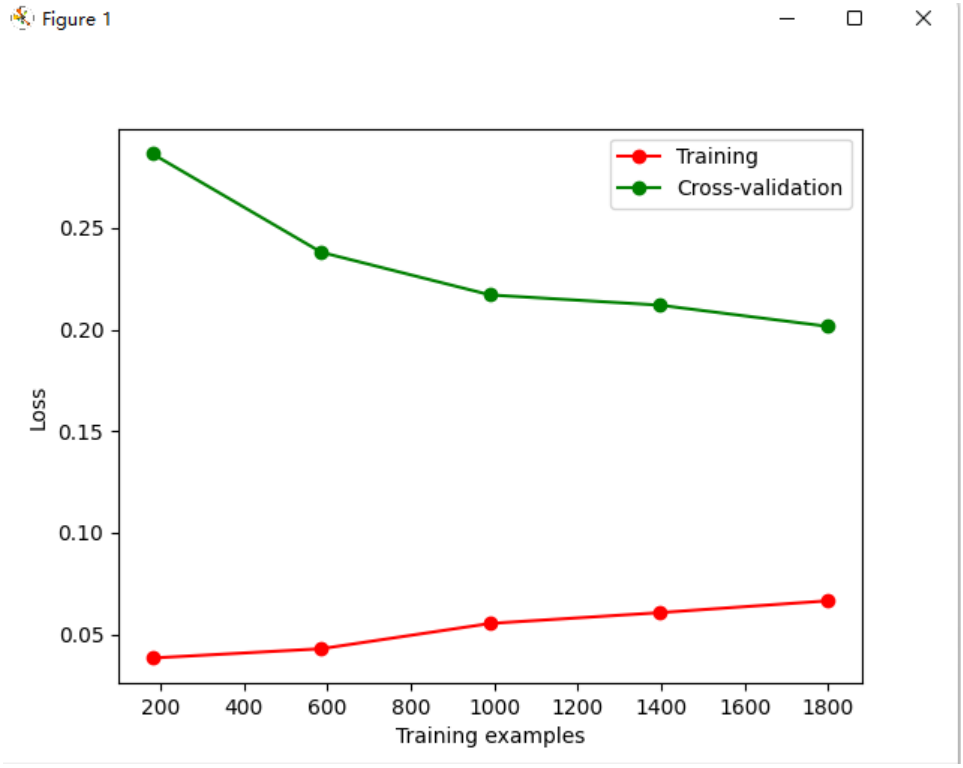

图 2-7 学习曲线

（3）微调

对基础模型进行微调，对微调后的模型再用 SVM 进行特征提取与分类，并重新编译和训练模型，对模型进行评估，结果如图 2-8 和图 2-9 所示。

```
accuracy:
0.984375
recall:
0.9833646512656129
```

图 2-8 模型评估结果

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99       109
         1.0       0.99      0.98      0.98        83

    accuracy                           0.98       192
   macro avg       0.98      0.98      0.98       192
weighted avg       0.98      0.98      0.98       192

Confusion Matrix:
 [[108   1]
 [  2  81]]
```

图 2-8 微调后模型的混淆矩阵

可以看到模型的准确率达到了将近 98%，绘制学习曲线。

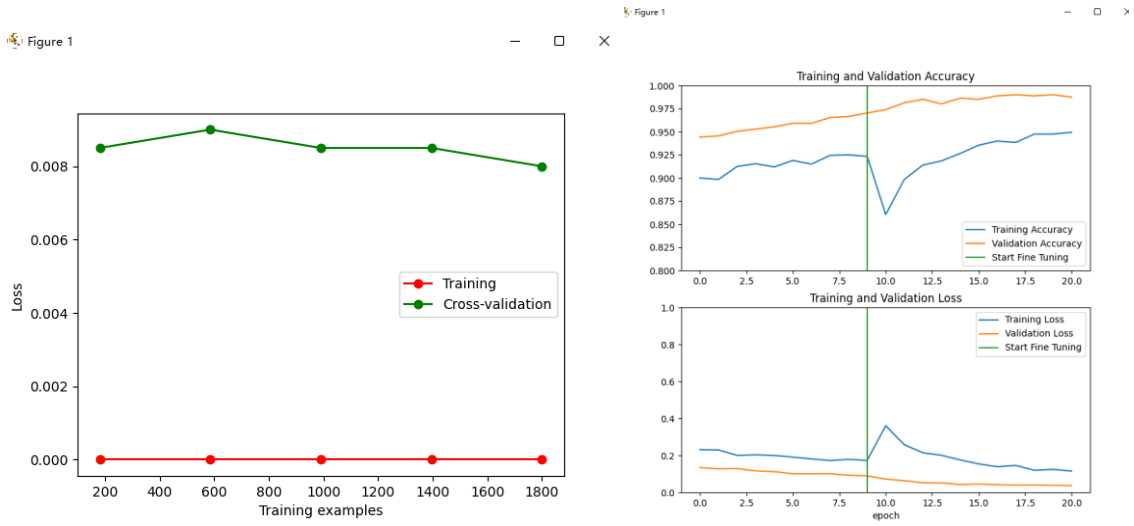

图 2-9 微调后的学习曲线

（4）预测

使用测试集在新数据上验证模型的性能。

```
362  ##评估与预测
363  loss, accuracy = model.evaluate(test_dataset)
364  print('Test accuracy :', accuracy)
```

图 2-10 验证性能

```
Test accuracy : 0.9947916865348816
```

图 2-11 预测结果

2.3 效果展示

展示测试集中的图像网格，并显示模型预测的类别标签（猫或狗），以展示模型的预测结果。

```
Predictions:
 [1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0]
Labels:
 [1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0]
```

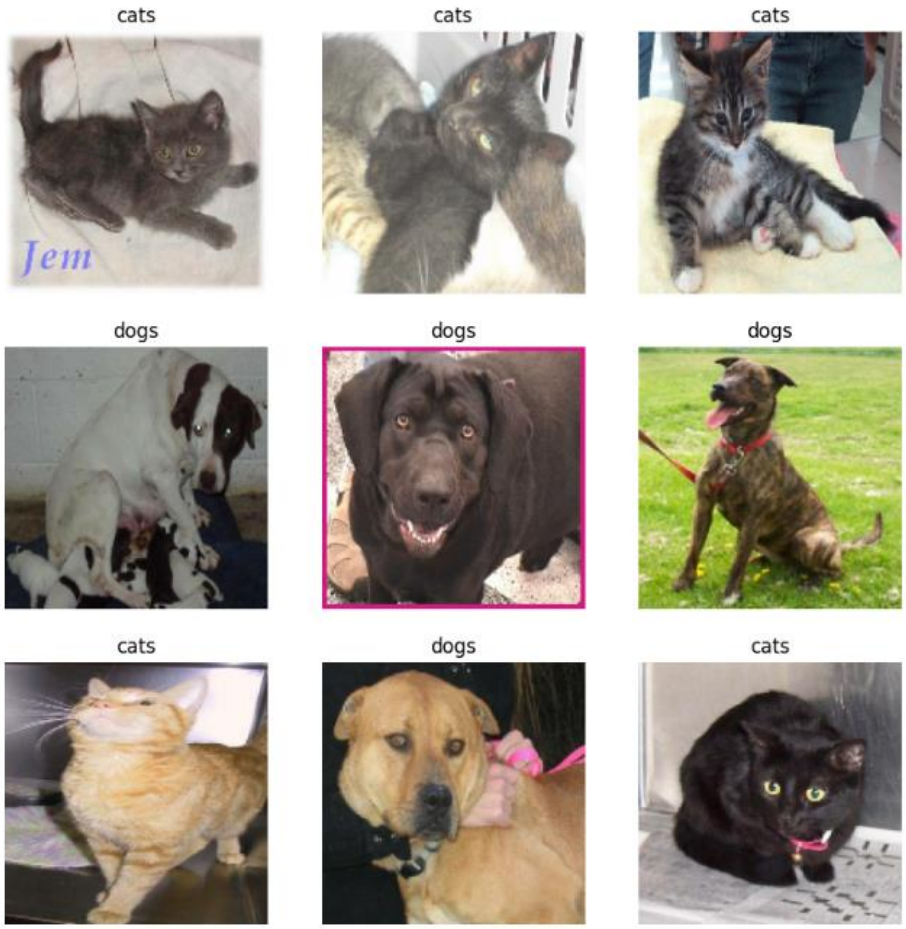

图 2-11 测试结果

2.4 思考和总结

在对未进行微调的模型进行 SVM 特征提取分类的时候，模型的精确度只有82%，远低于未使用 SVM 的微调前的模型。

但对微调后的模型进行 SVM 特征提取分类，模型的精确度达到了 98%左右，比未使用 SVM 的微调后的模型高了将近 3%。

在分析未使用 SVM 进行特征提取分类时精确度较低，而微调后的模型使用 SVM 特征提取分类精确度更高的原因时，考虑以下几个方面：

1. 特征的表达能力：微调后的模型经过更深层次的训练和调整，在提取特征时可能能够更好地捕捉数据的关键特征，使得特征的表达能力更强，进而提高了 SVM 模型的分类精确度。

2. 特征的维度：微调后的模型可能提取的特征维度更高，包含了更多的信息，这样在使用 SVM 进行特征提取分类时，可以更准确地划分不同类别之间的边界，从而提高了分类的精确度。

3. 模型的泛化能力：微调后的模型通常具有更好的泛化能力，能够更好地适应未见过的数据，这样在使用 SVM 进行分类时，可以在未见数据上取得更好的表现，从而提高了分类的精确度。

综合来看，微调后的模型在特征提取和 SVM 分类过程中能够更好地捕捉数据的关键特征，具有更好的泛化能力和参数调整，从而使得分类的精确度得到了提高。

第三章 总结与比较

3.1 总结

使用预训练的 MobileNet V2 模型作为基础，构建了图像分类模型，并通过编译和训练得到了模型。在训练集和验证集上达到了一定的准确率和较低的损失，但初始训练可能存在一定程度的过拟合。通过微调顶层模型，提高了模型的泛化能力和性能。在测试集上进行了评估和预测，得到了模型在未见数据上的表现。

利用迁移学习模型提取的特征，构建了 SVM 分类器，并对特征进行了训练和评估。通过混淆矩阵和其他指标评估了 SVM 模型在测试集上的性能。绘制了 SVM 模型的学习曲线，展示了模型的训练效果和泛化能力。

迁移学习模型利用了预训练模型的特征提取能力，适用于数据量较小的情况。模型结构简单清晰，易于理解和调整。但对于特定任务的表现可能不如从零开始训练的模型，因为预训练模型的特征可能不完全适用于目标任务。此外可能存在过拟合问题，特别是当训练数据集较小或者预训练模型特征不够泛化时。

SVM 是传统的机器学习模型，在特征提取和分类上表现稳定且性能良好。对于线性可分的数据或者合适的核函数选择，SVM 模型可以达到较高的分类准确率。对于小样本问题，SVM 模型能够有效避免过拟合，并且泛化能力强。但对于非线性可分的数据，需要合适的核函数选择，调参相对复杂。

3.2 比较

迁移学习模型具有较为复杂的模型结构，而 SVM 模型相对简单直观。迁移学习模型在初始训练时可能存在过拟合，需要通过微调等手段提高性能；而 SVM 模型在特定任务上表现稳定且性能良好。迁移学习模型对于图像分类任务具有一定的泛化能力，但需要调整模型结构和参数；SVM 模型对于小样本和线性可分的数据具有较好的泛化能力。

综合来看，迁移学习模型适用于需要更高泛化能力和复杂特征学习的任务，而 SVM 模型适用于对特征工程有较高要求和需要较稳定性能的任务。