实验四　猫狗分类

RT-AK 教育套件实验手册

WWW.RT-THREAD.ORG

Monday 22nd November, 2021

目录

第 1 章

实验介绍和目的

1.1 实验介绍

本实验是基于 Tensorflow 实现的第二个深度学习例程：猫狗图像分类。

作为计算机视觉和卷积神经网络的入门项目，猫狗图像分类的主要任务是：

随机输入一张猫/狗图像，通过 AI 模型推理最终得到图像所属分类的结果。

该实验是一个典型的二分类任务，相比第一节的手写数字识别实验，可以在实验过程中进一步提高对神经网络模型运行原理的认识，并初步了解卷积神经网络模型的结构。

图 **1.1:** *Cats.vs.Dogs*

1.2 实验目的

1. 进一步掌握 RT-AK 命令部署 AI 模型的使用
2. 学习卷积神经网络设计

第 2 章

实验器材

1. 上位机（电脑）
2. EgdeAI 实验板

第 3 章

实验步骤

确保环境安装无误后做再进行后续的实验，环境安装请参考以往实验教程

3.1 AI 模型训练

1. 打开本实验文件夹，该文件夹下包含Applications,Dataets等子文件夹，包含了运行实验项目所需的全部资料。查看主文件夹下的目录，确定cats_dogs_classfication.ipynb 在当前目录下，并在路径框下输入 jupyter notebook 命令用 Web 交互式计算笔记本打开猫狗图像分类训练文件。

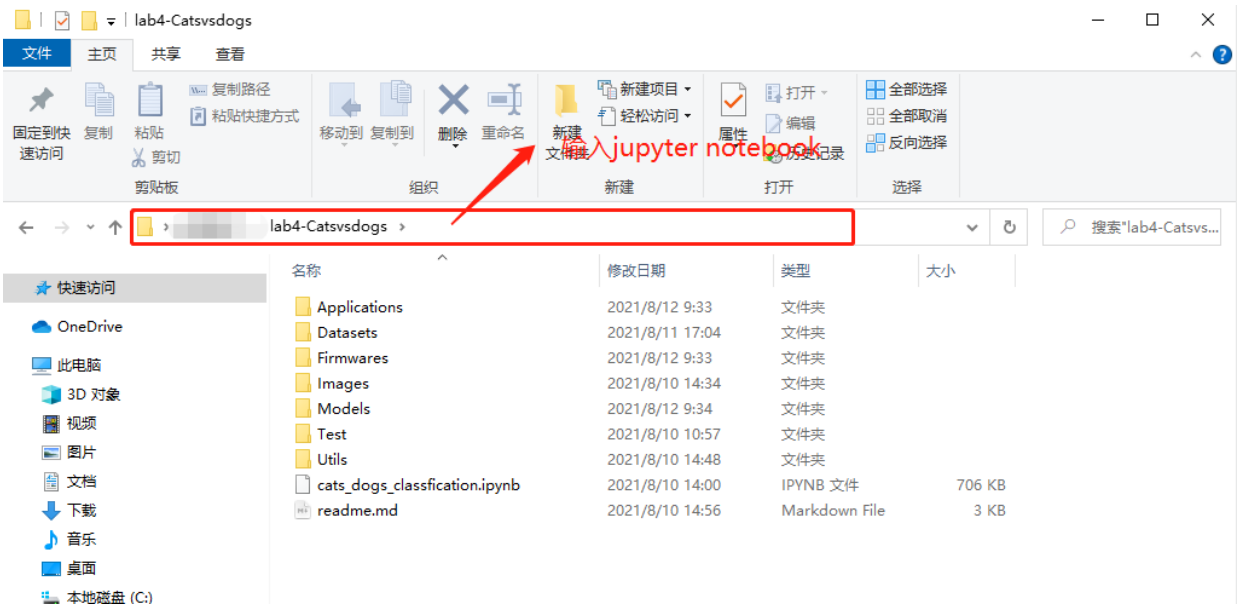

图 3.1: *命令行打开 jupyter notebook*

2. 在Jupyter notebook 中单击打开 cats_dogs_classfication.ipynb 文件，具体操作如下图所示

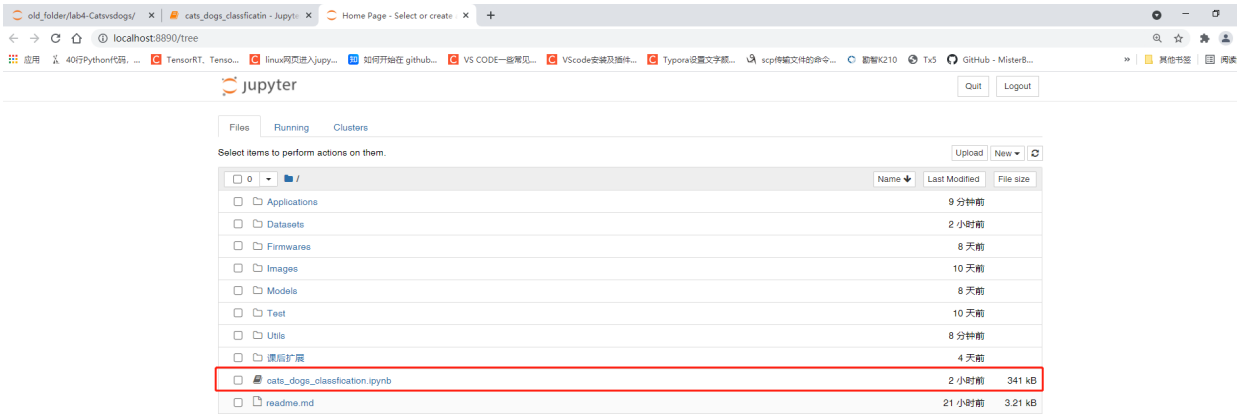

图 3.2: *ipynb 类型模型文件*

以下是猫狗图像分类模型训练代码的示例，内容摘自 cats_dogs_classfication.ipynb 文件

1. 导入库

```
####################################
#
#         导入库
#
####################################

import tensorflow as tf                                                    # 导
    入tensorflow
from tensorflow.keras import datasets, layers, models, backend, layers     #
    keras是tensorflow高级API，用来简化神经网络建模
import numpy as np                                                         #
    numpy是Python的一种开源的数值计算扩展
from matplotlib import pyplot as plt                                       #
    Matplotlib 是 Python 的绘图库
import os                                                                  # 导
    入标准库os 利用其中的API
os.environ["CUDA_VISIBLE_DEVICES"] = "0"                                   # 仅
    使用GPU训练时设置：设置当前使用的GPU设备仅为0号设备  设备名称为'/gpu:0'，
    也可以设置为 ="0,1,2,3..."
```

2. 准备数据集

```
####################################
#
#         准备数据集
#
####################################

# 根据实际情况进行设置batch size
BATCH_SIZE = 64

# 定义加载数据函数
```

```
# 这里需要注意img_path为你的路径，size为模型输入的尺寸，如果不符合该尺寸，将进行
    resize操作
def load_image(img_path,size = (240, 320)):
    # 自动设置标签
    label = tf.constant(1, tf.int8) if tf.strings.regex_full_match(img_path, ".*
        dogs.*") \
            else tf.constant(0, tf.int8)
    # 读取文件路径，将图片按顺序读入
    img = tf.io.read_file(img_path)
    # 注意此处为jpeg格式
    img = tf.image.decode_jpeg(img)
    # 对模型进行resize，并进行正则化
    img = tf.image.resize(img, size) / 255.0
    return(img, label)

# 使用并行化预处理num_parallel_calls 和预存数据prefetch来提升性能
# 读取训练和测试数据集，设置batch size
ds_train = tf.data.Dataset.list_files("./Datasets/cat_dog_dataset/train/*/*.jpg"
    ) \
            .map(load_image, num_parallel_calls = tf.data.experimental.AUTOTUNE)
                \
            .shuffle(buffer_size = 1000).batch(BATCH_SIZE) \
            .prefetch(tf.data.experimental.AUTOTUNE)

ds_test = tf.data.Dataset.list_files("./Datasets/cat_dog_dataset/validation/*/*.
    jpg") \
            .map(load_image, num_parallel_calls = tf.data.experimental.AUTOTUNE)
                \
            .batch(BATCH_SIZE) \
            .prefetch(tf.data.experimental.AUTOTUNE)

# 查看部分样本
plt.figure(figsize = (8, 8))
for i, (img, label) in enumerate(ds_train.unbatch().take(9)):
    ax = plt.subplot(3, 3, i+1)
    ax.imshow(img.numpy())
    ax.set_title("label = %d" %label)
    ax.set_xticks([])
    ax.set_yticks([])
plt.show()

for x, y in ds_train.take(1):
    print(x.shape, y.shape)
```

如果需要更换自己的数据集，需要更新ds_train和de_test对应的文件夹的相对路径，同时需要保证自己的数据集文件夹按照以下格式进行设置：

```
cat_dog_dataset
|
```

```
|-train
|+++-cats
|+++-dogs
|
|-validation
|+++-cats
|+++-dogs
```

使用plt.show()函数输出部分数据集的图片，同时显示图像具体的分类标签


```
(64, 240, 320, 3) (64,)
```

图 3.3: *显示部分测试集图片*

3. 搭建卷积神经网络猫狗图像分类模型

```
#####################################
#
#         创建模型
#
#####################################

model = tf.keras.models.Sequential([
                                                    # 利用sequential
    创建模型
    tf.keras.layers.Conv2D(16, 3, strides=(2, 2), padding='same', activation='
```

```
        relu',              # 进行卷积操作，模型输入层
                            input_shape=(240, 320, 3)),
                                                                    # 输入
                                图像的尺寸必须要和input_shape的结构相同。
    tf.keras.layers.MaxPooling2D(),
                                                                    #
        maxpooling层
    tf.keras.layers.BatchNormalization(),
                                                                #
        batchnormalization层
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
                                    # 第二层卷积
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
                                    # 第三层卷积
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
                                    # 第四层卷积
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, 1, padding='same', activation='relu'),
                                    # 第五层卷积
    tf.keras.layers.GlobalAveragePooling2D(),
                                                            # averagepooling
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
                                                                # dropout
        层
    tf.keras.layers.Flatten(),
                                                                    # 压平
        特征图
    tf.keras.layers.Dense(64, activation='relu'),
                                                        # relu激活函数
    tf.keras.layers.Dense(1, activation='sigmoid')
                                                    # sigmoid函数，二分类输出

])

# 选择ADAM优化器和二进制交叉熵损失函数。要查看每个训练时期的训练和验证准确性，请
    传递metrics参数。
model.compile(optimizer='adam',
                                                                    # 设置优化
    器
                loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=['accuracy'])

# 查看模型结构的具体信息
```

```
model.summary()
```

model.summary()函数输出模型的具体信息，如下图所示

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 120, 160, 16)      448
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 60, 80, 16)        0
_________________________________________________________________
batch_normalization_10 (Batc (None, 60, 80, 16)        64
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 60, 80, 32)        4640
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 30, 40, 32)        0
_________________________________________________________________
batch_normalization_11 (Batc (None, 30, 40, 32)        128
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 30, 40, 64)        18496
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 15, 20, 64)        0
_________________________________________________________________
batch_normalization_12 (Batc (None, 15, 20, 64)        256
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 15, 20, 64)        36928
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 7, 10, 64)         0
_________________________________________________________________
batch_normalization_13 (Batc (None, 7, 10, 64)         256
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 7, 10, 64)         4160
_________________________________________________________________
global_average_pooling2d_5 ( (None, 64)                0
_________________________________________________________________
batch_normalization_14 (Batc (None, 64)                256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
flatten_2 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 65
=================================================================
Total params: 69,857
Trainable params: 69,377
Non-trainable params: 480
_________________________________________________________________
```

图 3.4: *模型参数信息*

4. 模型训练

```
#######################################
#
#          训练模型
#
#######################################

# 当 accuracy 不再提升时，减小学习率
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='accuracy', factor=0.5,
    patience=4, min_lr=0.001,verbose=1)
# 当测试集准确率不再提高，停止训练
earlystop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=8,
    verbose=1)
# 指定优化器、损失函数和评价指标
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
# 根据参数训练模型
history = model.fit(ds_train,validation_data=-ds_test,verbose=2,epochs=50)
```

参数说明：

- `tf.keras.callbacks.ReduceLROnPlateau`：当 monitor 在 patience 个 epoch 中没出现明显变化，则会由因子 factor 来降低学习率，最小降到 min_lr 。
  - monitor：监控指标
  - factor：new_learning_rate = old_learning_rate * factor
  - patience: 监控指标没有明显提高的训练次数
  - min_lr：学习率的下界
  - verbose：是否打印出信息，1 为打印出信息

其他参数说明详见 Tensorflow API 文档 [链接地址：https://tensorflow.google.cn/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau?hl=zh-cn]

- `tf.keras.callbacks.EarlyStopping`: 当 monitor 在 patience 个 epoch 中没出现明显变化，则会停止训练。
  - monitor：监控指标
  - patience: 监控指标没有明显提高的训练次数
  - verbose：是否打印出信息，1 为打印出信息

其他参数说明详见 Tensorflow API 文档 [链接地址：https://tensorflow.google.cn/api_docs/python/tf/keras/callbacks/EarlyStopping?hl=zh-cn]

- `model.compile`: 指定优化器、损失函数和评价指标
  - optimizer：优化器。本实验中使用的是 SGD — 随机梯度下降，即在一批次中随机取一个样本的梯度作为整个批次的训练梯度
  - loss：损失函数。本实验中使用的是 spare_categorical_crossentropy—稀疏分类交叉熵，这是分类任务中最常用的是损失函数
  - metrics：评价指标。本实验中使用的是 accuracy，即正确率 = 预测正确样本数 / 总样本数
- `model.fit`: 根据参数训练模型
  - 第一个参数 x_y_train_ds 为训练集
  - validation_data：测试集
  - callbacks：训练过程中的回调
  - epochs：训练次数
  - 该方法返回值为 `history`，`history.history` 中包含了训练中每个 epoch 的各项指标值。
- `model.compile` 和 `model.fit` 其他参数说明详见 Tensorflow API 文档 [链接地址：https://tensorflow.google.cn/api_docs/python/tf/keras/Model?hl=zh-cn]

模型训练开始后，根据设置的 epochs 参数进行训练，训练过程如下图所示

```
Epoch 1/50
391/391 - 29s - loss: 0.6235 - accuracy: 0.6562 - val_loss: 0.9077 - val_accuracy: 0.5150
Epoch 2/50
391/391 - 25s - loss: 0.5272 - accuracy: 0.7380 - val_loss: 0.6099 - val_accuracy: 0.6920
Epoch 3/50
391/391 - 26s - loss: 0.4603 - accuracy: 0.7850 - val_loss: 0.4896 - val_accuracy: 0.7670
Epoch 4/50
391/391 - 26s - loss: 0.4019 - accuracy: 0.8219 - val_loss: 0.3779 - val_accuracy: 0.8380
Epoch 5/50
391/391 - 26s - loss: 0.3293 - accuracy: 0.8608 - val_loss: 0.3035 - val_accuracy: 0.8720
Epoch 6/50
391/391 - 27s - loss: 0.2710 - accuracy: 0.8876 - val_loss: 0.3210 - val_accuracy: 0.8560
Epoch 7/50
391/391 - 25s - loss: 0.2261 - accuracy: 0.9068 - val_loss: 0.3319 - val_accuracy: 0.8430
Epoch 8/50
391/391 - 26s - loss: 0.1964 - accuracy: 0.9193 - val_loss: 0.1474 - val_accuracy: 0.9350
Epoch 9/50
391/391 - 24s - loss: 0.1737 - accuracy: 0.9306 - val_loss: 0.7245 - val_accuracy: 0.7080
Epoch 10/50
391/391 - 27s - loss: 0.1484 - accuracy: 0.9398 - val_loss: 0.1213 - val_accuracy: 0.9570
Epoch 11/50
391/391 - 27s - loss: 0.1336 - accuracy: 0.9470 - val_loss: 0.1696 - val_accuracy: 0.9260
Epoch 12/50
391/391 - 27s - loss: 0.1243 - accuracy: 0.9513 - val_loss: 0.0835 - val_accuracy: 0.9640
Epoch 13/50
391/391 - 29s - loss: 0.1109 - accuracy: 0.9570 - val_loss: 0.1962 - val_accuracy: 0.9090
Epoch 14/50
391/391 - 27s - loss: 0.0968 - accuracy: 0.9617 - val_loss: 0.2168 - val_accuracy: 0.9140
Epoch 15/50
391/391 - 26s - loss: 0.0945 - accuracy: 0.9637 - val_loss: 0.0650 - val_accuracy: 0.9720
Epoch 16/50
391/391 - 26s - loss: 0.0810 - accuracy: 0.9682 - val_loss: 0.1036 - val_accuracy: 0.9550
```

图 3.5: *训练过程*

5. 模型保存

```
#####################################
#
#         模型保存
#
#####################################

# 保存为keras的.h5模型
keras_model = model.save('keras_model.h5')

# 保存为tflite的.tflite模型
tflite_model = tf.keras.models.load_model('keras_model.h5')
converter = tf.lite.TFLiteConverter.from_keras_model(tflite_model)
tflite_save = converter.convert()
open("cats_and_dogs_model.tflite", "wb").write(tflite_save)
```

当所有的代码执行完成之后，会在当前文件夹下得到两个模型文件：

文件	描述
keras_model.h5 后缀文件	使用 Tensorflow 中 Keras API 训练所得的模型文件
cats_and_dogs_model.tflite 后缀文件	RT-AK 所支持的模型文件格式

其中，两个模型的内部保存的都是网络结构和权重数据，将生成的文件手动保存到 Models 文件夹中。

6. 图片预测

```
#####################################
#
#         图片预测
#
#####################################

import cv2                                                    # 导入
    opencv，用于显示导入图片，显示图片
from pathlib import Path                                      # 导入模
    型路径
from PIL import Image                                         # 导入
    pillow

def predict_cat_dog(img_path, model):                         # 定义预
    测函数
    img = cv2.imread(str(img_path))                           # 读取文
        件路径
    img = cv2.resize(img, (320, 240))                         # 将读入
        的图片resize
    img = img.astype(np.float32)                              # 强制转
        换为float32
    img /= 255                                                # 归一化
    img = np.expand_dims(img, axis=0)                         # 扩充维
        度
    print(img.shape)                                          # 检查维
        度扩充前后差异
    pred = model.predict(img)                                 # 预测推
        片
    pred1 = model.predict_classes(img)                        # 预测分
        类
    print(pred, pred1)                                        # 打印预
        测结果

model = tf.keras.models.load_model('./Models/keras_model.h5')  # 加载模
    型
for img in Path("Test").glob("*.jpg"):                        # for循
    环，加载"test"文件夹中的.jpg文件
    predict_cat_dog(img, model)                               # 模型预
        测
```

下图为推理测试得到的结果，第一行显示为输入参数的维度，第二行则显示推理的结果和取整后的结果，与 Test 文件夹中的图片一一对应

```
(1, 240, 320, 3)
[[7.140072e-06]] [[0]]
(1, 240, 320, 3)
[[9.451146e-05]] [[0]]
(1, 240, 320, 3)
[[1.6176652e-11]] [[0]]
(1, 240, 320, 3)
[[0.00057435]] [[0]]
(1, 240, 320, 3)
[[2.7431593e-06]] [[0]]
(1, 240, 320, 3)
[[0.99958694]] [[1]]
(1, 240, 320, 3)
[[1.]] [[1]]
(1, 240, 320, 3)
[[0.99999356]] [[1]]
(1, 240, 320, 3)
[[1.]] [[1]]
(1, 240, 320, 3)
[[0.9996325]] [[1]]
```

图 3.6: *测试结果*

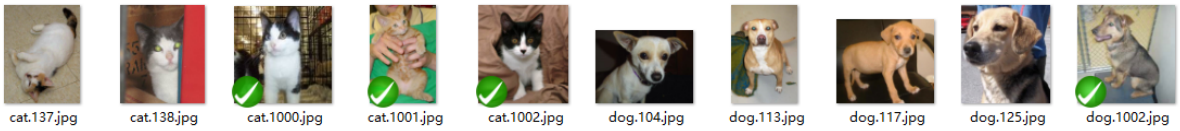

图 3.7: *测试图片*

3.2 模型信息

已训练好的模型放置在文件夹 `Models`中，后缀 h5 的为 keras 模型文件，后缀 tflite 的为 tflite 模型文件，**后者更适合部署**。模型训练时，所有数据都是 float32 类型，关于模型其他信息，我们需要说明以下几点：

- 本实验训练的猫狗分类模型结构简单，总参数量为 69857，tflite 模型文件大小为 280KB，适合部署至 K210
- 模型输入格式：NHWC，N=1，C=3，H=240，W=320, 类型为 float32
- 模型输出格式：N*1，N=1，类型为 float32

3.3 模型部署

在 `RT-AK/rt_ai_tools` 路径下打开 Windows 命令行终端（确保目标文件夹下面包含`aitools.py`文件）

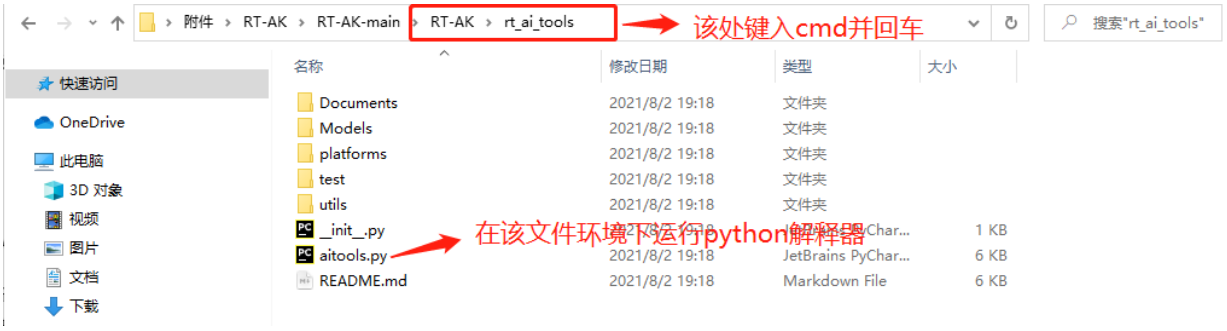

图 3.8: *运行 rt-ak*

在 Windows 输入以下命令：

```
# 量化为 uint8，使用 KPU 加速，量化数据集为图片
$ python aitools.py --project=<your_project_path> --model=<your_model_path> --
    model_name=cd --platform=k210 --dataset=<your_val_dataset>
```

--model_name=cd（cd 是 catsvsdogs 的简写）为了匹配实验中的示例代码，不要随意更改。下面是自己构建工程时使用的示例，自己搭建实验的过程中需要注意自己文件所在的文件夹和文件名

```
# 示例(量化模型，图片数据集)
$ python aitools.py --project="xxx\lab4-Catsvsdogs\xxx" --model="xxx\lab4-Catsvsdogs
    \Models\cats_and_dogs_model.tflite" --model_name=cd --platform=k210 --dataset="
    xxx\lab4-Catsvsdogs\Datasets\quantize_data"
```

其中，--project 是目标工程路径，--model 是目标模型路径，--model_name 是转化模型的名字，--platform 是指定插件支持的目标平台为 k210，--dataset 是模型量化所需要用到的数据集。

关于 RT-AK 工具的应用，更多详细的参数信息请看文档: RT-AK\rt_ai_tools\platforms\plugin_k210\README.md。

当部署成功之后，目标工程夹 xxx 会生成以下几个文件：

文件	描述
rt_ai_lib/	RT-AK Libs，模型推理库
applications/cd_kmodel.c	kmodel 的十六进制储存
applications/rt_ai_cd_model.c	与目标平台相关的信息
applications/rt_ai_cd_model.h	模型相关信息

同时，在 RT-AK\rt_ai_tools\platforms\plugin_k210 路径下会生成两个文件

文件	描述
cd.kmodel	k210 所支持的模型格式
convert_report.txt	tflite 模型转成 kmodel 格式的缓存信息

如果不想生成上述两文件，可以在模型部署的时候在命令行参数末尾加上：`--clear`，则在部署过程中即可删除相关中间文件

注意：

1、RT-AK 部署成功后不会产生应用代码，比如模型推理代码，需要在嵌入式项目工程中手工编写相关代码，详见 “3. 嵌入式 AI 模型应用”

2、在应用开发过程中，请遵守 RT-Thread 的编程规范以及 API 使用标准

3.4 嵌入式 AI 模型应用

使用 RT-AK 将训练好的 tflite 模型成功部署到工程之后，我们就可以开始着手编写应用层代码来使用该模型。本节的所有代码详见文件夹 `Applications`内部的文件

3.4.1 代码流程

系统内部初始化：

- kpu 时钟初始化

RT-AK Lib 模型加载并运行：

- 注册模型（代码自动注册，无需修改）
- 查找注册模型
- 初始化模型，挂载模型信息，准备运行环境
- 运行（推理）模型
- 获取输出结果
- 模型后处理

猫狗分类业务逻辑层：

- 输出最大值的索引

3.4.2 核心代码说明

```
/* read一帧图像,显示与AI图像会按顺序连续排放在buffer中 */
  display_image = rt_malloc((240 * 320 * 2)+(240 * 320 * 3));
  kpu_image = display_image + (240 * 320 * 2); //接着显示之后存放,注意地址偏移

  rt_device_t camera_dev = rt_device_find(CAMERA); //查找摄像头设备,CAMERA="ov2640
      " OR "ov5640" OR "gc0308"
  if(!camera_dev) {
      rt_kprintf("find camera err!\n");
      return -1;
  };
```

```
    rt_device_init(camera_dev); //初始化摄像头
    rt_device_open(camera_dev,RT_DEVICE_OFLAG_RDWR); //打开摄像头,读写模式
    rt_device_set_rx_indicate(camera_dev,camera_cb); //设置read回调函数

    mymodel = rt_ai_find(MY_MODEL_NAME);                        //查找模型
    if(!mymodel){
        rt_kprintf("\nmodel find error\n");
        while (1) {};
    }

    if (rt_ai_init(mymodel,(rt_ai_buffer_t*)kpu_image) != 0)      //模型初始化
    {
        rt_kprintf("\nmodel init error\n");
        while (1) {};
    }

    /* 使能系统全局中断 */
    sysctl_enable_irq();

    rt_kprintf("rt_ai_init complete..\n");


    //至此，所有运行前准备工作完成。

    while (1){
        g_dvp_finish_flag = 0;
        rt_device_read(camera_dev,0,display_image,0);
        while (g_dvp_finish_flag == 0) {};

        g_ai_done_flag= 0;
        if(rt_ai_run(mymodel,ai_done,&g_ai_done_flag) != 0){
            rt_kprintf("rtak run fail!\n");
            while (1);
        }
        while(!g_ai_done_flag);
        int pred = 0;
        float *output;
        output = rt_ai_output(mymodel,0);                          //输出kpu推理结果
        if (output[0] >= 0.5)                                                  //输出最
            大值作为推理结果
                pred = 1;
            else
                pred = 0;

        //lcd_clear(BLACK);
        rt_kprintf("The prediction is : %s\n",label[pred]);

        lcd_draw_picture(0, 0, 320, 240, (uint32_t *) display_image);
        lcd_draw_string(320/2 - 16, 20, label[pred], GREEN);
```

```
            ;
        }
    }
```

k210 推理时输入图片的说明：

- k210 图片数据输入格式是 CHW，chanel: c，heigh: H，width: W; 模型的输入是 3*240*320，C=3，H=240，W=320，图像数据类型为 uint8

第 4 章

编译和运行

4.1 编译

参考 lab2-env 教程中 Studio 使用方法。新建->RT-Thread项目->基于开发板->K210-RT-DRACO 输入工程目录和工程名，新建基于开发板的模板工程。将实验代码复制到 application 文件中替换原文件代码。点击 编译。会在你的工程根目录下生成一个 rtthread.bin 文件，然后参考下面的 烧录方法。其中 rtthread.bin 需要烧写到设备中进行运行。

4.2 烧录

连接好串口，点击 Studio 中的下载图标进行下载，详细可参考lab-env2

或者

使用 K-Flash 工具进行烧写 bin 文件。

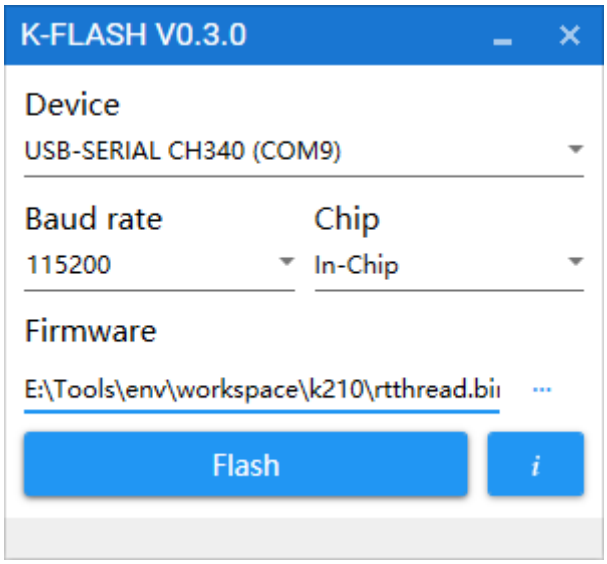

图 4.1: *K-Flash*

第 5 章

实验现象

如果编译 & 烧写无误，K-Flash 会自动打开 Windows 终端，自动连接实验板，系统启动后，我们的程序会自动运行。

- 本实验中，首先打印 Hello, RT-Thread。
- 其次，查找并初始化 AI 模型，并输出 rt_ai_init complete。
- 启用摄像头获取图像
- 将预测的结果显示在 LCD 上

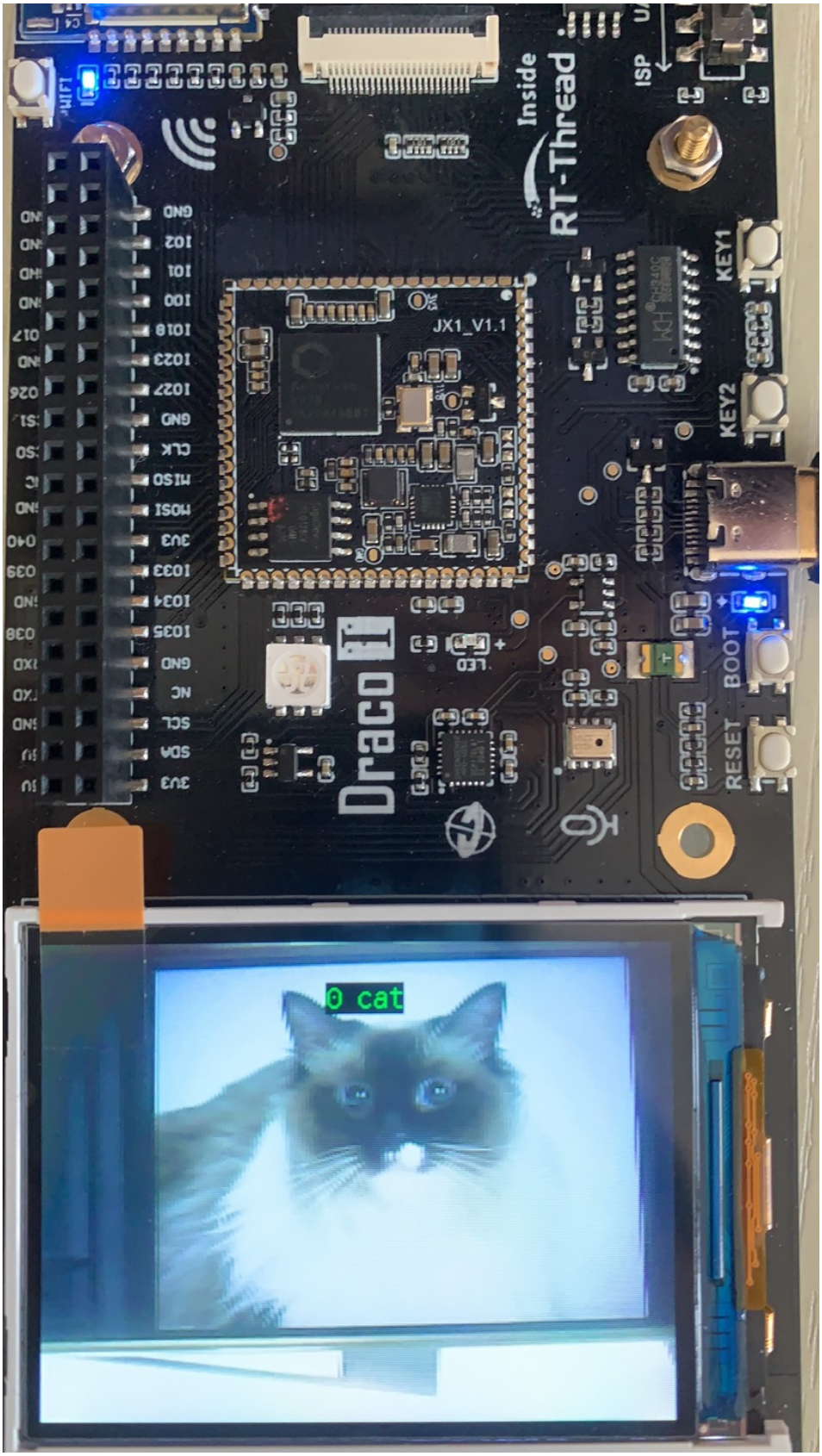

图 **5.1:** *结果显示*

第 6 章

API 使用说明

6.1 嵌入式 AI 开发 API 文档

```
rt_ai_t rt_ai_find(const char *name);
```

Paramaters	Description
name	注册的模型名
Return	–
rt_ai_t	已注册模型句柄
NULL	未发现模型

描述: 查找已注册模型

```
rt_err_t rt_ai_init(rt_ai_t ai, rt_aibuffer_t* work_buf);
```

Paramaters	Description
ai	rt_ai_t 句柄
work_buf	运行时计算所用内存
Return	–
0	初始化成功
非 0	初始化失败

描述: 初始化模型句柄, 挂载模型信息, 准备运行环境.

```
rt_err_t rt_ai_run(rt_ai_t ai, void (*callback)(void * arg), void *arg);
```

Paramaters	Description
ai	rt_ai_t 模型句柄

Paramaters	Description
callback	运行完成回调函数
arg	运行完成回调函数参数
Return	–
0	成功
非 0	失败

描述: 模型推理计算

```
rt_aibuffer_t rt_ai_output(rt_ai_t aihandle,rt_uint32_t index);
```

Paramaters	Description
ai	rt_ai_t 模型句柄
index	结果索引
Return	–
NOT NULL	结果存放地址
NULL	获取结果失败

描述: 获取模型运行的结果, 结果获取后.

`rt_ai_libs/readme.md` 文件中有详细说明

6.2 LCD API 说明手册

```
/**
 * @fn void lcd_init(void);
 * @brief LCD初始化
*/
void lcd_init(void);
/**
 * @fn void lcd_clear(uint16_t color);
 * @brief 清屏
 * @param color 清屏时屏幕填充色
*/
void lcd_clear(uint16_t color);
/**
 * @fn void lcd_set_direction(lcd_dir_t dir);
 * @brief 设置LCD显示方向
 * @param dir 显示方向参数
*/
void lcd_set_direction(lcd_dir_t dir);
```

```
/**
 * @fn void lcd_set_area(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
 * @brief 设置LCD显示区域
 * @param x1 左上角横坐标
 * @param y1 左上角纵坐标
 * @param x2 右下角横坐标
 * @param y2 右下角纵坐标
*/
void lcd_set_area(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
/**
 * @fn void lcd_draw_point(uint16_t x, uint16_t y, uint16_t color);
 * @brief 画点
 * @param x 横坐标
 * @param y 纵坐标
 * @param color 颜色
*/
void lcd_draw_point(uint16_t x, uint16_t y, uint16_t color);

/**
 * @fn void lcd_draw_string(uint16_t x, uint16_t y, char *str, uint16_t color);
 * @brief 显示字符串
 * @param x 显示位置横坐标
 * @param y 显示位置纵坐标
 * @param str 字符串
 * @param color 字符串颜色
*/
void lcd_draw_string(uint16_t x, uint16_t y, char *str, uint16_t color);
/**
 * @fn void lcd_draw_picture(uint16_t x1, uint16_t y1, uint16_t width, uint16_t
     height, uint32_t *ptr);
 * @brief 显示图片
 * @param x1 左上角横坐标
 * @param y1 左上角纵坐标
 * @param width 图片宽
 * @param height 图片高
 * @param ptr 图片地址
*/
void lcd_draw_picture(uint16_t x1, uint16_t y1, uint16_t width, uint16_t height,
    uint32_t *ptr);
/**
 * @fn void lcd_draw_rectangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,
     uint16_t width, uint16_t color);
 * @brief 画矩形
 * @param x1 左上角横坐标
 * @param y1 左上角纵坐标
 * @param x2 右下角横坐标
 * @param y2 右下角纵坐标
 * @param width 线条宽度(当前无此功能)
 * @param color 线条颜色
```

```
*/
```

6.3 Camera 使用

Camera 已对接 RT-Thread 设备驱动框架，可通过 RT-Thread Device 标准接口进行调用。RT-Thread 设备框架文档：https://www.rt-thread.org/document/site/#/rt-thread-version/rt-thread-standard/README

下面将使用到的 API 进行简要说明，引用自 RTT 官方文档：

查找设备

应用程序根据设备名称获取设备句柄，进而可以操作设备。查找设备函数如下所示：

```
rt_device_t rt_device_find(const char* name);
```

参数	描述
name	设备名称
返回	——
设备句柄	查找到对应设备将返回相应的设备句柄
RT_NULL	没有找到相应的设备对象

初始化设备

获得设备句柄后，应用程序可使用如下函数对设备进行初始化操作：

```
rt_err_t rt_device_init(rt_device_t dev);
```

参数	描述
dev	设备句柄
返回	——
RT_EOK	设备初始化成功
错误码	设备初始化失败

[!NOTE] 注：当一个设备已经初始化成功后，调用这个接口将不再重复做初始化 0。

打开和关闭设备

通过设备句柄，应用程序可以打开和关闭设备，打开设备时，会检测设备是否已经初始化，没有初始化则会默认调用初始化接口初始化设备。通过如下函数打开设备：

```
rt_err_t rt_device_open(rt_device_t dev, rt_uint16_t oflags);
```

参数	描述
dev	设备句柄
oflags	设备打开模式标志
返回	——
RT_EOK	设备打开成功
-RT_EBUSY	如果设备注册时指定的参数中包括 RT_DEVICE_FLAG_STANDALONE 参数，此设备将不允许重复打开
其他错误码	设备打开失败

oflags 支持以下的参数：

```
#define RT_DEVICE_OFLAG_CLOSE 0x000   /* 设备已经关闭（内部使用）*/
#define RT_DEVICE_OFLAG_RDONLY 0x001  /* 以只读方式打开设备 */
#define RT_DEVICE_OFLAG_WRONLY 0x002  /* 以只写方式打开设备 */
#define RT_DEVICE_OFLAG_RDWR 0x003    /* 以读写方式打开设备 */
#define RT_DEVICE_OFLAG_OPEN 0x008    /* 设备已经打开（内部使用）*/
#define RT_DEVICE_FLAG_STREAM 0x040   /* 设备以流模式打开 */
#define RT_DEVICE_FLAG_INT_RX 0x100   /* 设备以中断接收模式打开 */
#define RT_DEVICE_FLAG_DMA_RX 0x200   /* 设备以 DMA 接收模式打开 */
#define RT_DEVICE_FLAG_INT_TX 0x400   /* 设备以中断发送模式打开 */
#define RT_DEVICE_FLAG_DMA_TX 0x800   /* 设备以 DMA 发送模式打开 */
```

应用程序打开设备完成读写等操作后，如果不需要再对设备进行操作则可以关闭设备，通过如下函数完成：

```
rt_err_t rt_device_close(rt_device_t dev);
```

参数	描述
dev	设备句柄
返回	——
RT_EOK	关闭设备成功
-RT_ERROR	设备已经完全关闭，不能重复关闭设备
其他错误码	关闭设备失败

读写设备

应用程序从设备中读取数据可以通过如下函数完成（摄像头设备中pos 和size参数无作用）：

```
rt_size_t rt_device_read(rt_device_t dev, rt_off_t pos,void* buffer, rt_size_t size)
    ;
```

参数	描述
dev	设备句柄
pos	读取数据偏移量
buffer	内存缓冲区指针，读取的数据将会被保存在缓冲区中
size	读取数据的大小
返回	——
读到数据的实际大小	如果是字符设备，返回大小以字节为单位，如果是块设备，返回的大小以块为单位
0	需要读取当前线程的 errno 来判断错误状态

数据收发回调

当硬件设备收到数据时，可以通过如下函数回调另一个函数来设置数据接收指示，通知上层应用线程有数据到达：

```
rt_err_t rt_device_set_rx_indicate(rt_device_t dev, rt_err_t (*rx_ind)(rt_device_t
    dev,rt_size_t size));
```

参数	描述
dev	设备句柄
rx_ind	回调函数指针
返回	——
RT_EOK	设置成功