**飞行器图像分类**

李卓彧 詹铠铭 郑安玮

**摘要：**经过一学期的人工智能程序设计课程的学习，我们选定了“飞行器图像分类”这个项目作为大作业进行开展，以求更加熟练地掌握Python语言的实际运用。我们查阅了网络上现存的关于“图像分类”的各种实例，选定了主流的CNN与SVM两种分类器，通过比较实验观察在“飞行器图像分类”这一课题下两种分类器的表现，最终得出CNN算法在大数据集下的图像分类性能高于SVM算法。

目录

[1. 介绍 2](#_Toc126514400)

[1.1 概述 2](#_Toc126514401)

[1.2 项目分工 2](#_Toc126514402)

[2. 实验方法与算法分析 2](#_Toc126514403)

[2.1 图像分类算法 2](#_Toc126514404)

[3. 实验结果分析 7](#_Toc126514405)

[4. 代码演示 8](#_Toc126514406)

[4.1 SVM分类器 8](#_Toc126514407)

[4.2 CNN分类器 11](#_Toc126514408)

# 1. 介绍

### 1.1 概述

图像分类，即根据各自在图像信息中所反映的不同特征，把不同类别的目标区分开来的图像处理方法。它利用计算机对图像进行定量分析，把图像或图像中的每个像元或区域划归为若干个类别中的某一种，以代替人的视觉判读。

本课题针对“飞行器种类”开展，最终实现将数据集分类为6个类别，分别对应着6种飞行器，截至目前，我们已经通过CNN与SVM两种主流分类器成功运行并测试了项目，并分别得到了两种分类器在此课题下的准确度表现。

从实验结果上看，CNN算法在此课题上的表现要优于SVM算法，但两种分类器的准确度反馈均低于0.50，根据对样本的人工分析和算法结构的讨论，我们总结出了两种算法出现准确度低下的一些原因。

### 1.2 项目分工

* 李卓彧：分析问题并制定CNN分类器计划；
* 郑安玮：项目规划与实验性能对比分析；
* 詹铠铭：预处理数据并制定SVM分类器计划；

# 2. 实验方法与算法分析

### 2.1 图像分类算法

图像分类算法是指将图像数据预处理后，训练不同的分类器算法，以达到能够为不同种类的图片打上对应标签的目的。目前主流的分类器算法包括CNN，KNN，SVM，BP等，其中CNN（卷积神经网络）为代表的传统分类器算法在特征量较高的图像识别领域表现出众。

为了达到图像分类的目的，图像分类项目的流程基本如下：

测试结果数据

运行分类器

训练分类器

预处理

##### 2.1.1 预处理

首先我们知道，图像的预处理基本完全与数据本身有关，从数据采集的瞬间开始，我们就需要考虑预处理的事项。因此，进行数据的预处理是有一定的前提的，我们将其总结为三点：

1. 全部样本的尺寸是一致的（同时，全部样本的通道数是一致的）；
2. 图像最终以Tensor形式被输入卷积网络；
3. 图像被恰当地归一化；

只有达到以上三个要求，对图像数据的预处理才是最有效的。

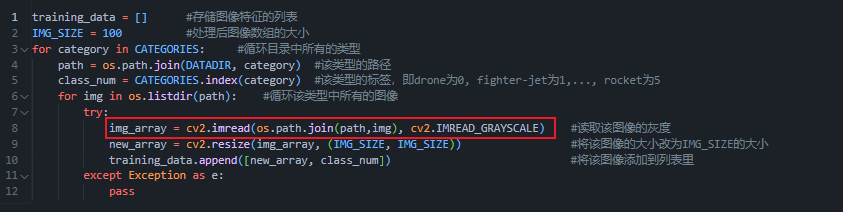
与分类器算法类似，样本数据的预处理方法也存在主流的几种，例如灰度转化，RGB三通道分离，直方图灰度体现等等。本次课题采用了灰度转化和RGB三通道分离两种预处理方法，分别应用于SVM算法与CNN算法。

###### 2.1.1.1 灰度转化

将彩色图像转化成为灰度图像的过程称为图像的**灰度转化**。灰度图像是R、G、B三个分量相同的一种特殊的彩色图像，其一个像素点的变化范围为255种，所以在数字图像处理种一般先将各种格式的图像转变成灰度图像以使后续的图像的计算量变得少一些。而灰度图像的描述与彩色图像一样仍然反映了整幅图像的整体和局部的色度和亮度等级的分布和特征。

本次飞行器分类数据集的类别比较多，每一个类别里面包含大量图像，并且飞行器的形状对颜色没有太大的影响，所以初步考虑用灰度处理会比较有效，同时简化模型的训练过程，在创建cnn模型时也能增加更多层。

当使用cv2.imread函数读取图像的时候，第二参数直接传入cv2.IMREAD\_GRAYSCALE就能实现读取图像的灰度。最后用cv2.resize函数将图像修改为相同大小，就基本完成了图像的预处理。如下图所示。



###### 2.1.1.2 RGB三通道分离

相较于灰度转化，RGB三通道保留了原始数据的更多特征，有利于分类器更好地识别图像，但同时增加了训练难度和实验耗时。

使用opencv和os遍历图片，将图片数据，类标，路径分开储存，并将数据/255归一化处理。（注释中本想使用PIL库读取图片解决png无法正常读取问题，后发现RGB通道问题及Convolution2D读取格式问题，选择忽视数据中的png图片）如下图所示。



##### 2.1.2分类器的训练与运行测试

如开篇所说，本次课题采用了CNN与SVM两种主流的分类器算法，但是这两类算法在处理数据时衍生出了许多的模型，不同的模型在具体实例中的表现也是不一样的。常见的处理模型有SVC，随机森林，决策树，简单神经网络，AlexNet等。我们在测试代码时一一采用了，这里仅列出表现较好的几种模型。

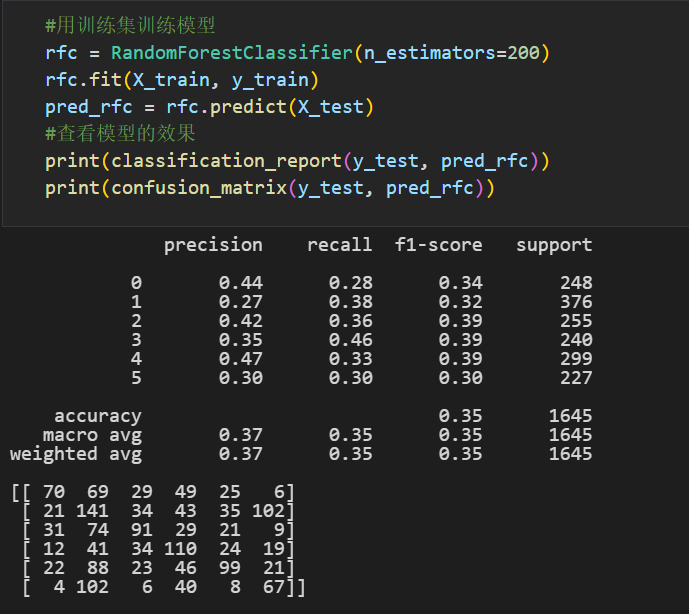
###### 2.1.2.1 SVC模型

**Accuracy：0.35**

**Macro avg：0.34**

**Weighted avg：0.34**

###### 2.1.2.2 随机森林模型



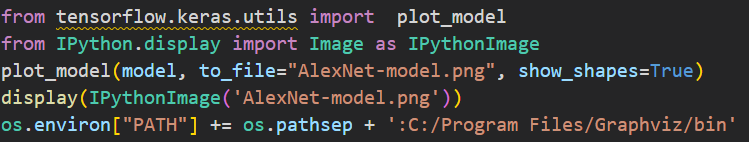
**Accuracy：0.35**

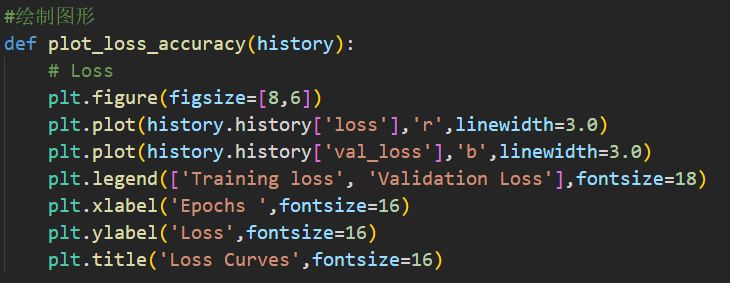
**Macro avg：0.35**

**Weighted avg：0.35**

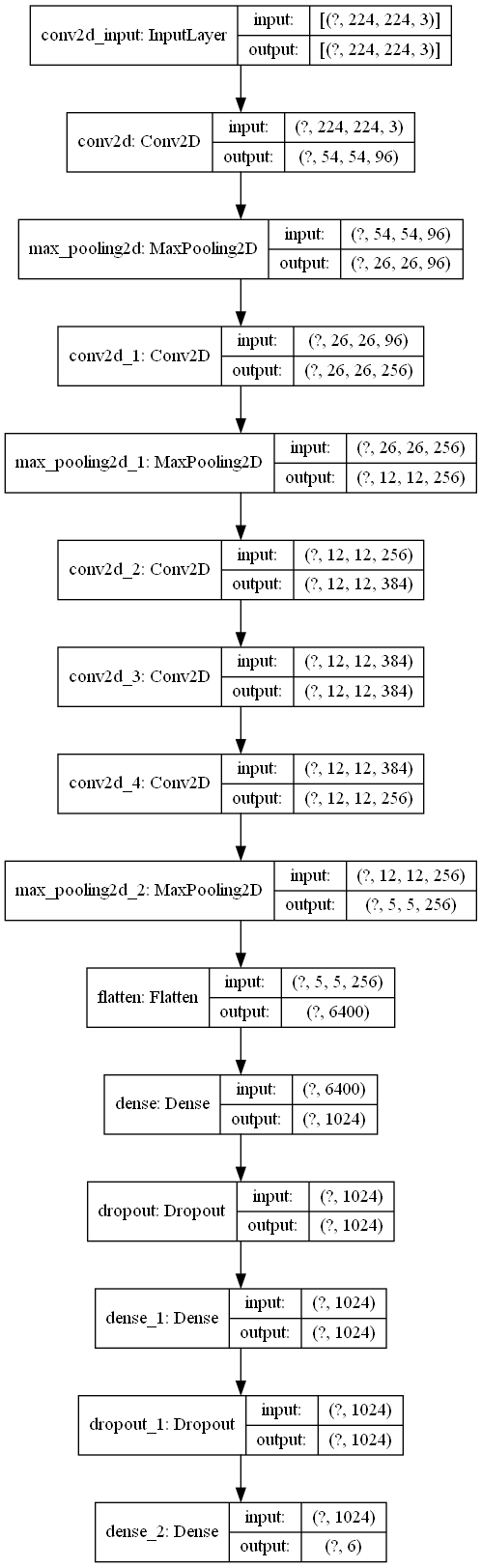
###### 2.1.2.3 AlexNet模型

代码实现：

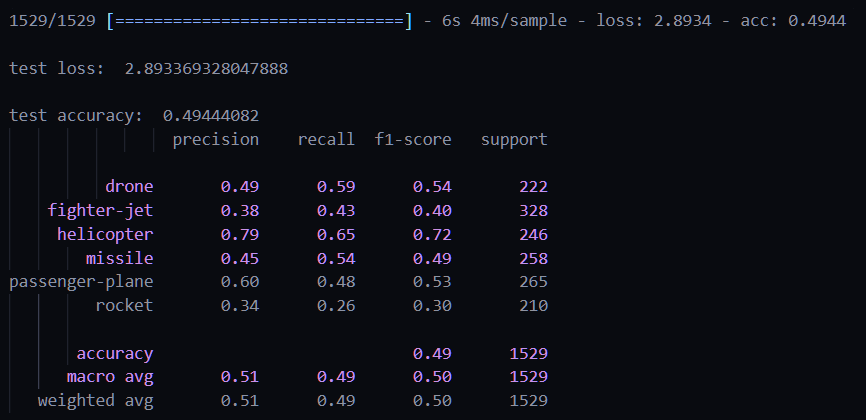




利用Matplotlib对模型结构进行可视化：



测试结果：



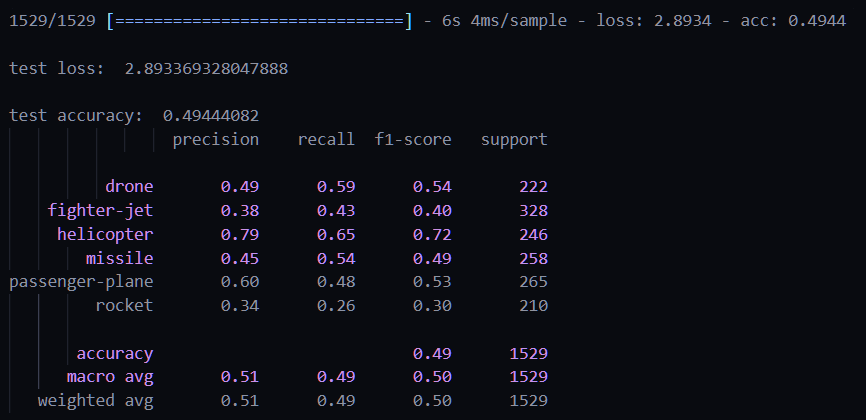
**Accuracy：0.49**

**Macro avg：0.50**

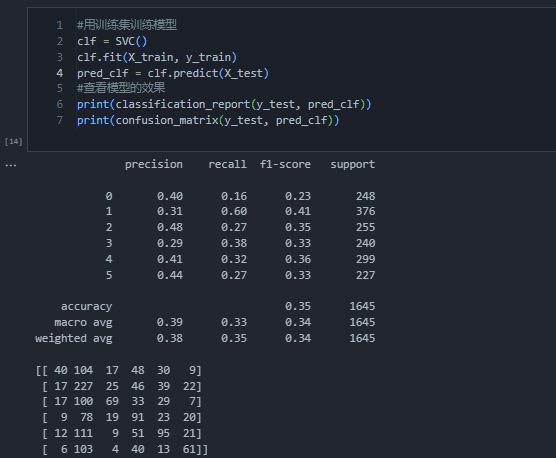
**Weighted avg：0.50**

我们将SVC和随机森林模型应用到了SVM分类器上，将AlexNet应用到了CNN分类器上，实验结果显著。

# 实验结果分析

根据测试结果，我们发现CNN处理器在Acccuracy，macro avg和weighted avg三个指标上都优于SVM分类器，对此我们进行了一定程度的探讨。

CNN分类器



SVM分类器

SVM分类器

从结论上来讲，SVM算法相较于传统的CNN算法，更擅长于轻量化的数据挖掘，但是在对特征识别要求较高的图像分类上，CNN（卷积神经网络）的优势则体现了出来。

CNN算法是一种通过提取图像特征，获取图像的信息，并通过减小图像的尺寸，减少输入参数的深度学习算法，对于图像处理有出色表现。而SVM是一个面向数据的分类算法，基本模型是在特征空间上找到最佳的分离超平面使得训练集上正负样本间隔最大。

对于CNN算法，它拥有以下特点：

* 实现简单；
* 分类时计算量简单小，存储资源低，可解释性强，比较适合处理有缺失属性值的样本，能够处理不相关的特征；
* 分类精度高；
* 对噪声数据有很好的健壮性；
* 容易过拟合（故采用了AlexNet模型）

而对于SVM算法：

* 分类思想很简单，就是将样本与决策面的间隔最大化；
* 在中小量样本规模的时候容易得到数据和特征之间的非线性关系；
* 对大规模数据训练比较困难；
* 对缺失数据敏感，对非线性问题没有通用的解决方案，核函数的正确选择不容易；

对于“图像分类”这个应用而言，在预处理过程中尽可能地保留图像特征无疑能够提高最终拟合的准确度。但是观察样本数据，数据集中存在着大量色彩噪音浓厚的个体，这对模型训练造成的阻碍比较大。

本次作业数据集包含6个大类，数据集容量超过7000份，直接导致了CNN算法运行测试使用了大量时间，以及SVM算法测试精度低下。

另外，在我们使用cv2去处理数据集时，对png与wepg等类型文件出现了报错，导致我们不得不舍弃掉一部分数据，这也是本次作业中两大模型准确度偏低的原因之一。

# 代码演示

### SVM分类器

##### Training

1. #读取X.pickle和y.pickle文件，分别传递到特征项X和目标项y变量
2. import pickle
3. X = pickle.load(open("X.pickle", "rb"))
4. y = pickle.load(open("y.pickle", "rb"))
5. #导入相关的包
6. from sklearn.neighbors import KNeighborsClassifier
7. from sklearn.ensemble import RandomForestClassifier
8. from sklearn.svm import SVC
9. from sklearn.tree import DecisionTreeClassifier
10. from sklearn.preprocessing import StandardScaler
11. from sklearn.model\_selection import train\_test\_split
12. from sklearn.metrics import classification\_report, confusion\_matrix
13. from sklearn.feature\_extraction import image
14. X\_new = X.reshape(-1, X.shape[1] \* X.shape[2])  #将特征项X的所有特征展平为一维
15. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_new, y, test\_size=0.2, random\_state=42)   #将特征项X和目标相y拆分为训练集和测试集
16. #用训练集训练随机森林模型
17. rfc = RandomForestClassifier(n\_estimators=200)
18. rfc.fit(X\_train, y\_train)
19. pred\_rfc = rfc.predict(X\_test)
20. #查看模型的效果
21. print(classification\_report(y\_test, pred\_rfc))
22. print(confusion\_matrix(y\_test, pred\_rfc))
23. #用训练集训练SVC模型
24. clf = SVC()
25. clf.fit(X\_train, y\_train)
26. pred\_clf = clf.predict(X\_test)
27. #查看模型的效果
28. print(classification\_report(y\_test, pred\_clf))
29. print(confusion\_matrix(y\_test, pred\_clf))

##### 4.1.2 Preprocess

#导入相关的包

import numpy as np

import cv2

import os

import random

from matplotlib import pyplot as plt

import pickle

import imghdr

DATADIR = "./飞行器分类数据集/data/planes/"

CATEGORIES = os.listdir(DATADIR)

training\_data = []      #存储图像特征的列表

IMG\_SIZE = 150          #处理后图像数组的大小

image\_exts = ['jpeg','jpg', 'bmp', 'png']

for category in CATEGORIES:     #循环目录中所有的类型

    path = os.path.join(DATADIR, category)  #该类型的路径

    class\_num = CATEGORIES.index(category)  #该类型的标签，即drone为0, fighter-jet为1,..., rocket为5

    for img in os.listdir(path):    #循环该类型中所有的图像

        image\_path = os.path.join(path,img) #该图像的路径

        try:

            tip = imghdr.what(image\_path)   #查看该图像的文件类型

            if tip in image\_exts:           #若该图像的文件类型存在于image\_exts列表

                img\_array = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)    #读取该图像的灰度

                new\_array = cv2.resize(img\_array, (IMG\_SIZE, IMG\_SIZE))                 #将该图像的大小改为IMG\_SIZE的大小

                training\_data.append([new\_array, class\_num])                            #将该图像添加到列表里

            else:

                print('Image not in ext list {}'.format(image\_path))

        except Exception as e:

            print('Issue with image {}'.format(image\_path))

#查看被存储的图像是否正确

plt.imshow(training\_data[-1][0], cmap = 'gray')

plt.show()

random.shuffle(training\_data)   #随机打乱列表中图像的顺序

#training\_data拆分为特征项X和目标项y

X = []

y = []

for features, label in training\_data:

    X.append(features)

    y.append(label)

y = np.array(y)

X = np.array(X)

# 把特征项除以255，缩放到0至1的范围内。

X\_scaled = X / 255.0

#将特征项X和目标相y分别存储到X.pickle和y.pickle文件，方便后续使用

pickle\_out = open("X.pickle", "wb")

pickle.dump(X\_scaled, pickle\_out)

pickle\_out.close()

pickle\_out = open("y.pickle", "wb")

pickle.dump(y, pickle\_out)

pickle\_out.close()

### 4.2 CNN分类器

import os

import glob

import cv2

import random

import numpy as np

import tensorflow as tf

from PIL import Image

from tensorflow.python.keras.layers.convolutional import MaxPooling2D, Convolution2D

from tensorflow.python.keras.optimizers import SGD

from sklearn.preprocessing import LabelBinarizer

from tensorflow.python.keras.layers.core import Dense, Dropout, Activation, Flatten

from tensorflow.python.keras.models import Sequential

import argparse

from keras import regularizers

from matplotlib import pyplot as plt

from sklearn.metrics import classification\_report

path = "./data/planes/"

#读取图像

def read\_img(path):

    cate = [path + x for x in os.listdir(path) if os.path.isdir(path + x)]

    imgs = []

    labels = []

    fpath = []

    for idx, folder in enumerate(cate):

        # 遍历整个目录判断每个文件是不是符合

        for im in glob.glob(folder + '/\*.jpg'):

            IMG\_SIZE = 224

            # print('reading the images:%s' % (im))

            try:

             img = cv2.imread(im)

             # img = Image.open(im).convert("RGB")  # 调用PIL库读取该图像

             # # 将rgb通道转换为bgr

             # img = cv2.cvtColor(np.asarray(img), cv2.COLOR\_RGB2BGR)

             # # print(img.shape)

             img = cv2.resize(img, (IMG\_SIZE, IMG\_SIZE),interpolation=cv2.INTER\_CUBIC)  # 图像像素大小一致

             # print(img.shape)

             imgs.append(img)  # 图像数据

             labels.append(idx)  # 图像类标

             fpath.append(path + im)  # 图像路径名

             # print(path+im, idx)

            except:

                continue

    return np.asarray(fpath, np.string\_), np.asarray(imgs, np.float32)/255, np.asarray(labels, np.int32)

# 读取图像

fpaths, data, label = read\_img(path)

print(data.shape)  # (7641, 28, 28, 3)

# data=np.transpose(data,(0,3,1,2))

# 计算有多少类图片

num\_classes = len(set(label))

print(num\_classes)

# 生成等差数列随机调整图像顺序

num\_example = data.shape[0]

# print(num\_example)

arr = np.arange(num\_example)

np.random.shuffle(arr)

data = data[arr]

label = label[arr]

fpaths = fpaths[arr]

# 拆分训练集和测试集 80%训练集 20%测试集

ratio = 0.8

s = np.int(num\_example \* ratio)

x\_train = data[:s]

y\_train = label[:s]

fpaths\_train = fpaths[:s]

x\_val = data[s:]

y\_val = label[s:]

fpaths\_test = fpaths[s:]

# 6824 6824 1706 1706

print(len(x\_train),len(y\_train),len(x\_val),len(y\_val))

lb=LabelBinarizer()

y\_train=lb.fit\_transform(y\_train)

y\_val=lb.fit\_transform(y\_val)

print(y\_train)

#GPU加速

import os

import tensorflow as tf

os.environ["CUDA\_DEVICES\_ORDER"] = "PCI\_BUS\_IS"

os.environ["CUDA\_VISIBLE\_DEVICES"] = "0"

gpu\_options = tf.GPUOptions(per\_process\_gpu\_memory\_fraction=0.8)

sess = tf.Session(config=tf.ConfigProto(gpu\_options=gpu\_options))

# # 建立基础的神经网络

# model=Sequential()

# model.add(Convolution2D(filters = 32,kernel\_size = 5,strides = 1,padding = 'same',input\_shape = (28,28,3)))

# # 增加神经网络激活函数

# model.add(Activation('relu'))

# model.add(tf.keras.layers.Dropout(0.5))

# # 池化层利用MaxPooling2D

# model.add(MaxPooling2D(

#     pool\_size = (2, 2),             # 向下取样

#     strides = (2,2),                # 取样跳2个

#     padding='same',                 # Padding method

# ))

# model.add(tf.keras.layers.Dropout(0.5))  # 加上DropOut防止过拟合

# model.add(Convolution2D(64, 5, 5, padding='same'))

# model.add(Activation('relu'))

# model.add(MaxPooling2D(pool\_size = (2, 2), padding='same'))

# model.add(Flatten())                  # 将三维层拉直

# model.add(Dense(64))                # 全连接层

# model.add(Activation('relu'))         # 激励函数

# model.add(Dense(6))                  # 输出6个单位

# model.add(Activation('softmax'))      # 分类激励函数

# LR=0.001

# EPOCHS=300

# # 定义优化器 学习效率设置为0.001

# opt =SGD(lr=LR)

#建立AlexNet模型

def create\_model(optimizer='adam', kernel\_initializer='he\_normal', activation='relu'):

    #第一层卷积

    #卷积核数量96 尺寸11\*11 步长4 激活函数relu

    #最大池化 尺寸3\*3，步长2

    model = Sequential()

    model.add(Convolution2D(filters=96,

                     kernel\_size=11,

                     strides=4,

                     input\_shape=(224, 224, 3),

                     activation=activation))

    model.add(MaxPooling2D(pool\_size=3, strides=2))

    #第二层卷积

    #卷积核数量256 尺寸5\*5 激活函数relu same卷积

    #最大池化 尺寸3\*3，步长2

    model.add(Convolution2D(filters=256,

                     kernel\_size=5,

                     padding='same',

                     activation=activation))

    model.add(MaxPooling2D(pool\_size=3, strides=2))

    #第三层卷积

    #卷积核数量384 尺寸3 激活函数relu same卷积

    model.add(Convolution2D(filters=384,

                     kernel\_size=3,

                     padding='same',

                     activation=activation))

    #第四层卷积

    #卷积核数量384 尺寸3 激活函数relu same卷积

    model.add(Convolution2D(filters=384,

                     kernel\_size=3,

                     padding='same',

                     activation=activation))

    #第五层卷积

    #卷积核数量256 尺寸3 激活函数relu same卷积

    #最大池化 尺寸3\*3，步长2

    model.add(Convolution2D(filters=256,

                     kernel\_size=3,

                     padding='same',

                     activation=activation))

    model.add(MaxPooling2D(pool\_size=3, strides=2))

    #展平特征图

    model.add(Flatten())

    #第一个全连接 1024神经元 relu

    model.add(Dense(1024, activation='relu'))

    model.add(Dropout(0.5))

    #第二个全连接 4096神经元 relu

    model.add(Dense(1024, activation='relu'))

    model.add(Dropout(0.5))

    #第二个全连接 输出6类结果

    model.add(Dense(6, activation='softmax'))

    # 定义损失函数-

    model.compile(loss="categorical\_crossentropy",optimizer=optimizer,metrics=["accuracy"])

    return model

#创建模型

optimizer =tf.train.AdamOptimizer(learning\_rate=1e-4)

model = create\_model(optimizer,

                     kernel\_initializer='uniform',

                     activation='relu')

from tensorflow.keras.utils import  plot\_model

from IPython.display import Image as IPythonImage

plot\_model(model, to\_file="AlexNet-model.png", show\_shapes=True)

display(IPythonImage('AlexNet-model.png'))

os.environ["PATH"] += os.pathsep + ':C:/Program Files/Graphviz/bin'

#绘制图形

def plot\_loss\_accuracy(history):

    # Loss

    plt.figure(figsize=[8,6])

    plt.plot(history.history['loss'],'r',linewidth=3.0)

    plt.plot(history.history['val\_loss'],'b',linewidth=3.0)

    plt.legend(['Training loss', 'Validation Loss'],fontsize=18)

    plt.xlabel('Epochs ',fontsize=16)

    plt.ylabel('Loss',fontsize=16)

    plt.title('Loss Curves',fontsize=16)

# 模型训练

h=model.fit(x\_train,y\_train,validation\_data=(x\_val,y\_val),epochs=30,batch\_size=64)

# 模型测试

predicts=model.predict(x\_val,batch\_size=64)

loss, accuracy = model.evaluate(x\_val, y\_val)

print('\ntest loss: ', loss)

print('\ntest accuracy: ', accuracy)

print(classification\_report(y\_val.argmax(axis=1),

                                predicts.argmax(axis=1), target\_names=["drone","fighter-jet","helicopter","missile","passenger-plane","rocket"]))