**暨南大学本科实验报告专用纸**

课程名称 人工智能原理 成绩评定

实验项目名称 深度学习 指导教师 张佳

实验项目编号 实验项目类型 综合 实验地点 线上

学生姓名 陈宇 学号 2020101642

学院 信息科学技术学院 系 计算机系 专业 软件工程

实验时间2022 年 11月25日下午～11月25日 下午 温度 ℃湿度

# 实验内容

本次实验学习和体会深度学习的神经网络算法，利用python的tensorflow实现对猫狗图片的识别。在实现对任务一的完成后，通过增加训练集，验证集的方法，观察模型训练的结果。完成任务2后，对神经网络模型的结构进行更改，观察在变化网络层数，卷积核数，卷积核大小后模型的变化。

# 核心代码

训练模型：train.py

import os

import tensorflow as tf

from keras.optimizers import RMSprop

from keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

# 数据集地址，如果出现地址错误，则考虑使用绝对地址

base\_dir = 'dataset/cats\_and\_dogs'

# 指定每一种数据的位置

train\_dir = os.path.join(base\_dir, 'train')

validation\_dir = os.path.join(base\_dir, 'validation')

# Directory with our training cat/dog pictures

train\_cats\_dir = os.path.join(train\_dir, 'cats')

train\_dogs\_dir = os.path.join(train\_dir, 'dogs')

# Directory with our validation cat/dog pictures

validation\_cats\_dir = os.path.join(validation\_dir, 'cats')

validation\_dogs\_dir = os.path.join(validation\_dir, 'dogs')

# 模型处添加了 dropout 随机失效，也就是说有时候可能不用到某些神经元，失效率为0.5

# 下面是模型的整体结构，可以观察到每一层卷积之后，都会使用一个最大池化层对提的数据进行降维，减少计算量，后续实验修改网络结构主要修改下面部分

# 设计模型

model = tf.keras.models.Sequential([

 # 我们的数据是 150x150 而且是三通道的，所以我们的输入应该设置为这样的格式。

 tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)),

 tf.keras.layers.MaxPooling2D(2, 2), # 最大池化层

 tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

 tf.keras.layers.MaxPooling2D(2, 2),

 tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),

 tf.keras.layers.MaxPooling2D(2, 2),

 tf.keras.layers.Dropout(0.5), # dropout 层通通过忽略一般数量特征，可以减少过拟合现象

 tf.keras.layers.Flatten(), # 全链接层，将多维输入一维化

 tf.keras.layers.Dense(512, activation='relu'),

 # 二分类只需要一个输出

 tf.keras.layers.Dense(1, activation='sigmoid')

])

# 进行优化方法选择和一些超参数设置

# 因为只有两个分类。所以用 2 分类的交叉熵，使用 RMSprop，学习率为 0.0001.优化指标为 accuracy

model.compile(loss='binary\_crossentropy',  # 损失函数使用交叉熵

  optimizer=RMSprop(lr=1e-4),  # 优化器，学习率设置为 0.0001

  metrics=['acc'])

# 数据处理

# 把每个数据都放缩到 0 到 1 范围内

# 这里的代码进行了更新，原来这里只进行归一化处理，现在要进行数据增强。

train\_datagen = ImageDataGenerator(

 rescale=1. / 255,

 rotation\_range=40,

 width\_shift\_range=0.2,

 height\_shift\_range=0.2,

 shear\_range=0.2,

 zoom\_range=0.2,

 horizontal\_flip=True,

 fill\_mode='nearest')

test\_datagen = ImageDataGenerator(rescale=1. / 255)

# 生成训练集的带标签的数据

train\_generator = train\_datagen.flow\_from\_directory(train\_dir, # 训练图片的位置

 batch\_size=20, # 每一个投入多少张图片训练

 class\_mode='binary', # 设置我们需要的标签类型

 target\_size=(150, 150)) # 将图片统一大小

# 生成验证集带标签的数据

validation\_generator = test\_datagen.flow\_from\_directory(validation\_dir, # 验证图片的位置

 batch\_size=20, # 每一个投入多少张图片训练

 class\_mode='binary', # 设置我们需要的标签类型

 target\_size=(150, 150)) # 将图片统一大小

# 进行训练

history = model.fit\_generator(

 train\_generator, # 训练集数据

 steps\_per\_epoch=20, # 每个 epoch 训练多少次

 epochs=10, # 训练轮数，建议在[10,50]如果电脑训练速度快，可以大于 50

 validation\_data=validation\_generator, # 验证集数据

 validation\_steps=10,

 verbose=1) # 训练进度显示方式，可取值 0，1（显示训练进度条），2（一个 epoch输出一条信息）

# 保存训练的模型到当前目录

model.save('model.h5')

# 得到精度和损失值

acc = history.history['acc'] # train\_acc

val\_acc = history.history['val\_acc'] # val\_acc

loss = history.history['loss'] # train\_loss

val\_loss = history.history['val\_loss'] # val\_loss

epochs = range(len(acc)) # 得到迭代次数

# 绘制精度曲线

plt.plot(epochs, acc)

plt.plot(epochs, val\_acc)

plt.title('Training and validation accuracy')

plt.legend(('Training accuracy', 'validation accuracy'))

plt.figure()

# 绘制损失曲线

plt.plot(epochs, loss)

plt.plot(epochs, val\_loss)

plt.legend(('Training loss', 'validation loss'))

plt.title('Training and validation loss')

plt.show()

模型预测：predict.py

# 预测

from tensorflow.python.keras.models import load\_model

import numpy as np

from keras.utils import image\_utils

# 测试图片地址

path = 'dataset/predicted/dog1.jpeg'

# 加载模型

model = load\_model('model.h5')

# 将图片转换成 150\*150 的格式，与模型训练的输入保持一致

img = image\_utils.load\_img(path, target\_size=(150, 150))

x = image\_utils.img\_to\_array(img) / 255.0

# 在第 0 维添加维度变为 1x150x150x3，和我们模型的输入数据一样

x = np.expand\_dims(x, axis=0)

# np.vstack:按垂直方向（行顺序）堆叠数组构成一个新的数组，我们一次只有一个数据所以不这样也可以

images = np.vstack([x])

# batch\_size 批量大小，程序会分批次地预测测试数据，这样比每次预测一个样本会快

classes = model.predict(images, batch\_size=1)

# classes[0]表示分类概率，大于 0.5 表示分类为 dog，小于 0.5 表示分类为 cat

print(classes[0])

if classes[0] > 0.5:

    print("It is a dog")

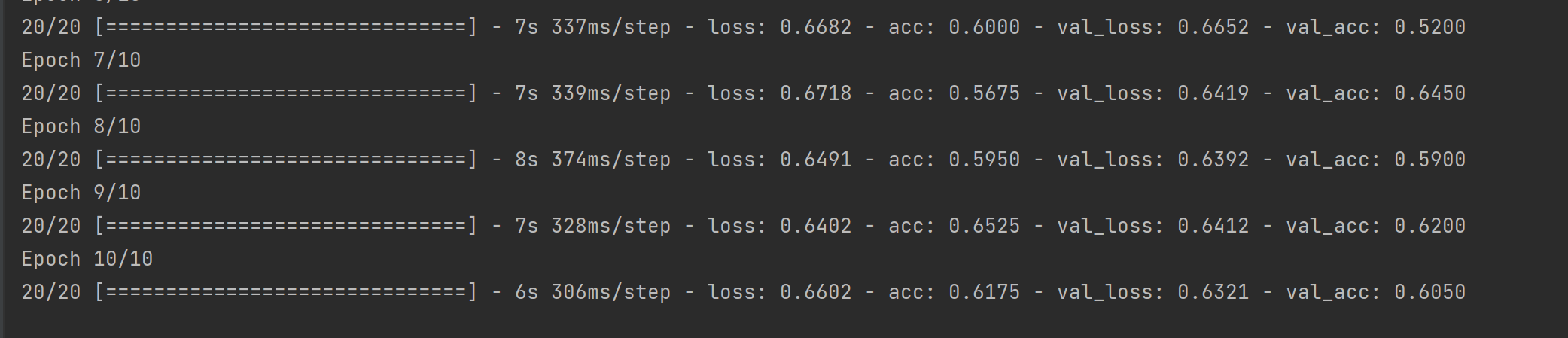
else:

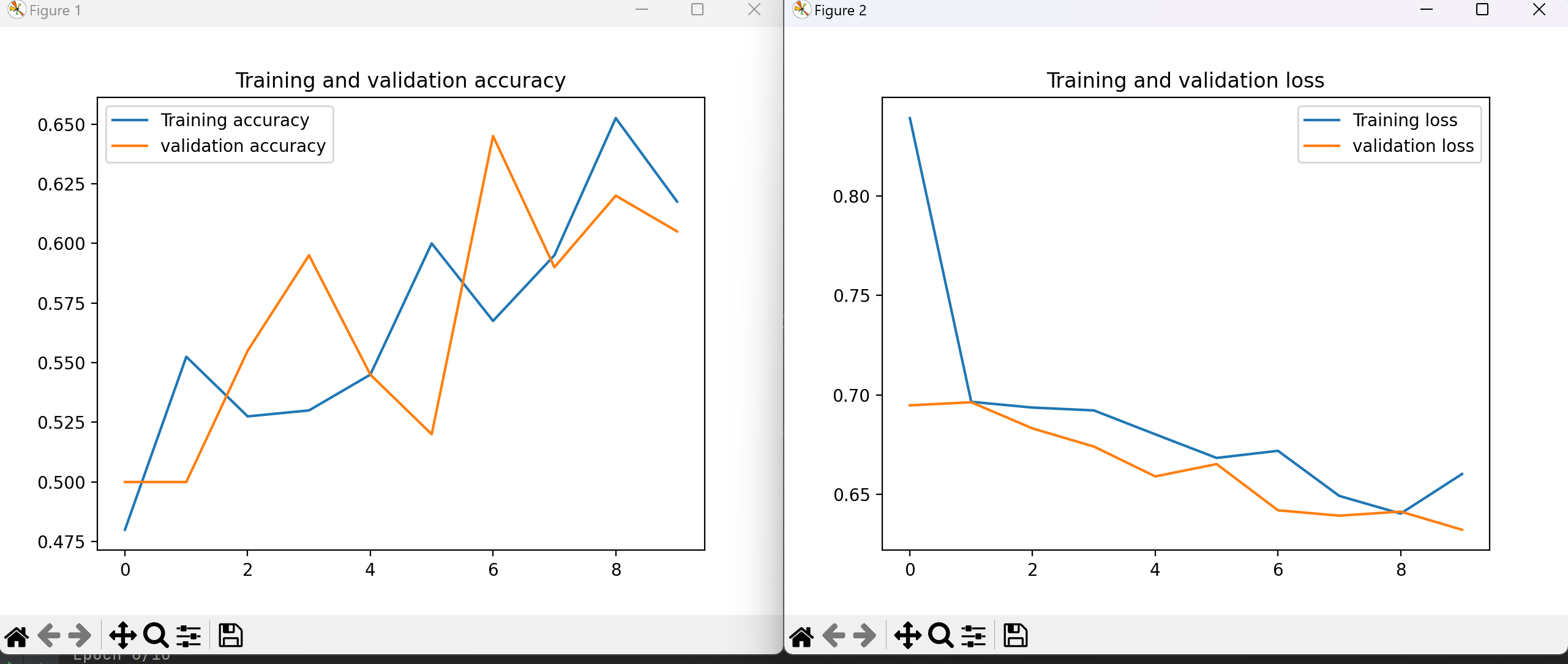
    print("It is a cat")

# 实验结果

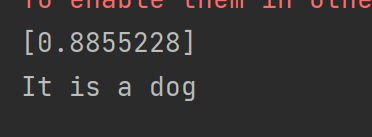
任务一：

模型数据：



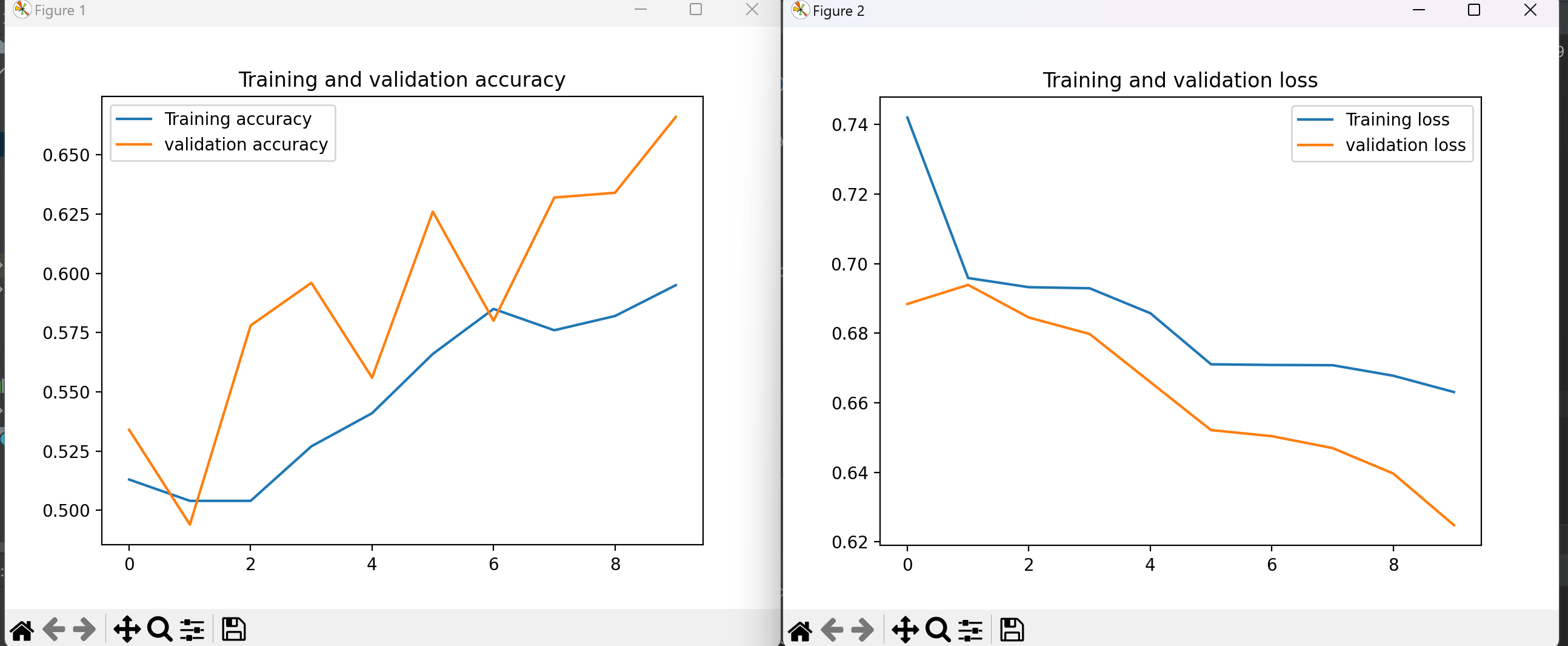


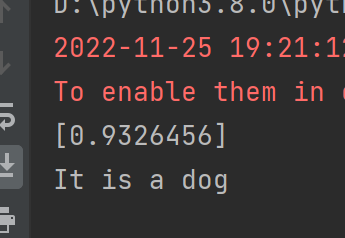
模型预测：



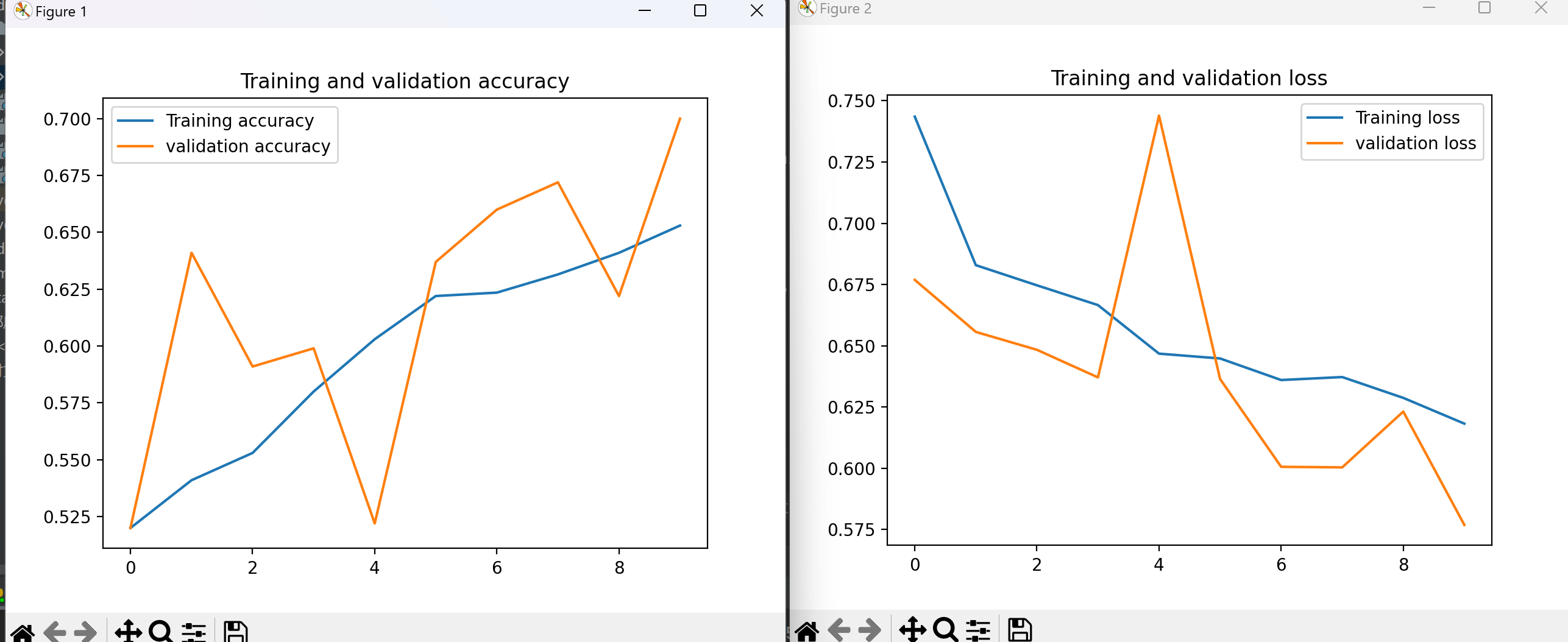
任务二：

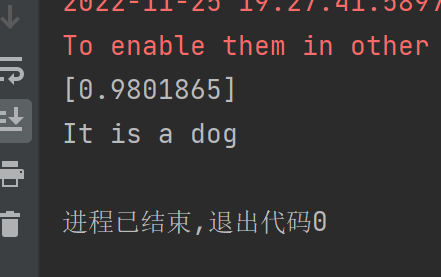
训练集，验证集大小：50





训练集，验证集大小：100



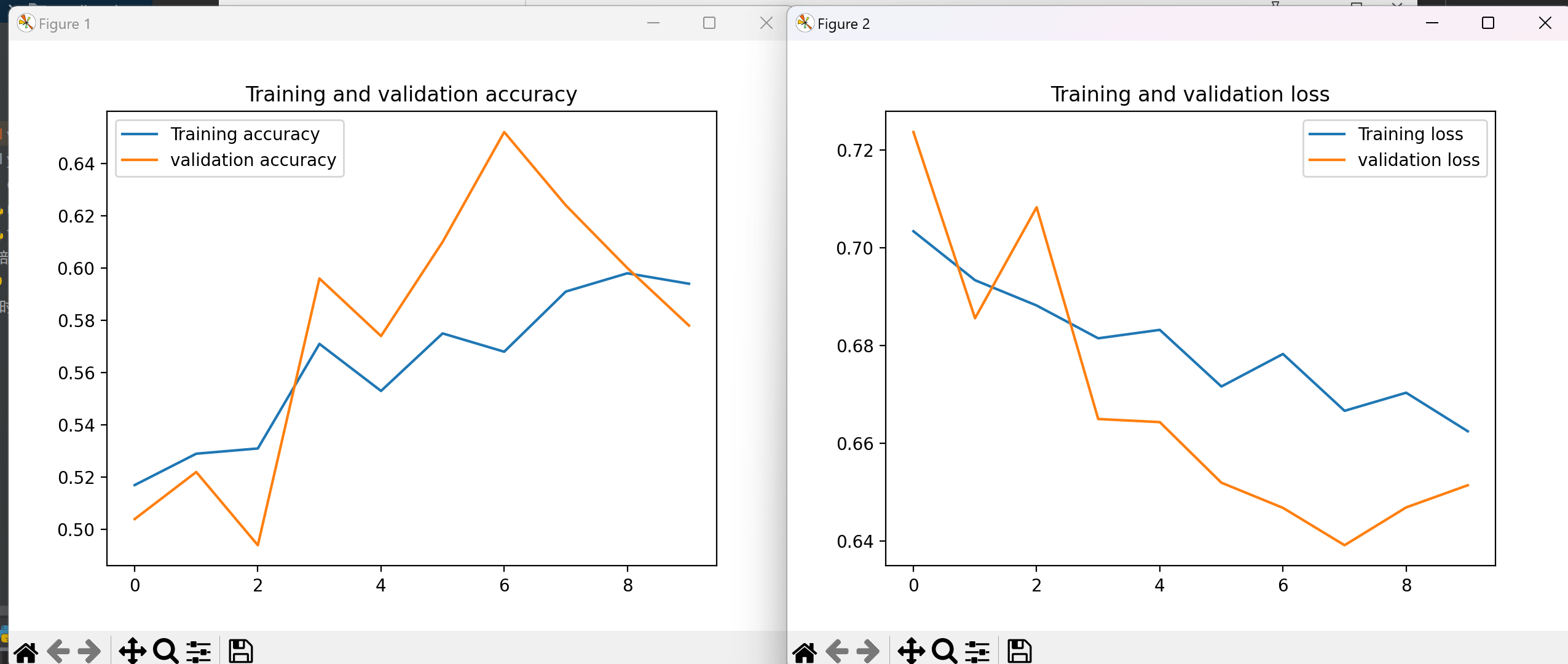


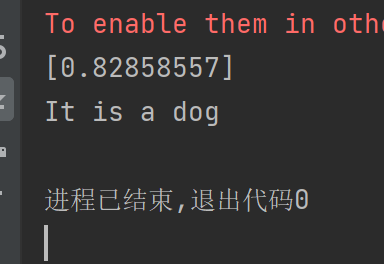
由不同数量的训练集和验证集可以得知，随着训练轮次的增加，训练验证的准确率都上升，损失值都下降，随着训练集和验证集的数量增加，准确率进一步上升，损失值进一步下降：（注意到纵坐标的数值变化），对预测图片的分类概率也越来越大（偏向正确答案狗）

任务三：

增加网络层数：

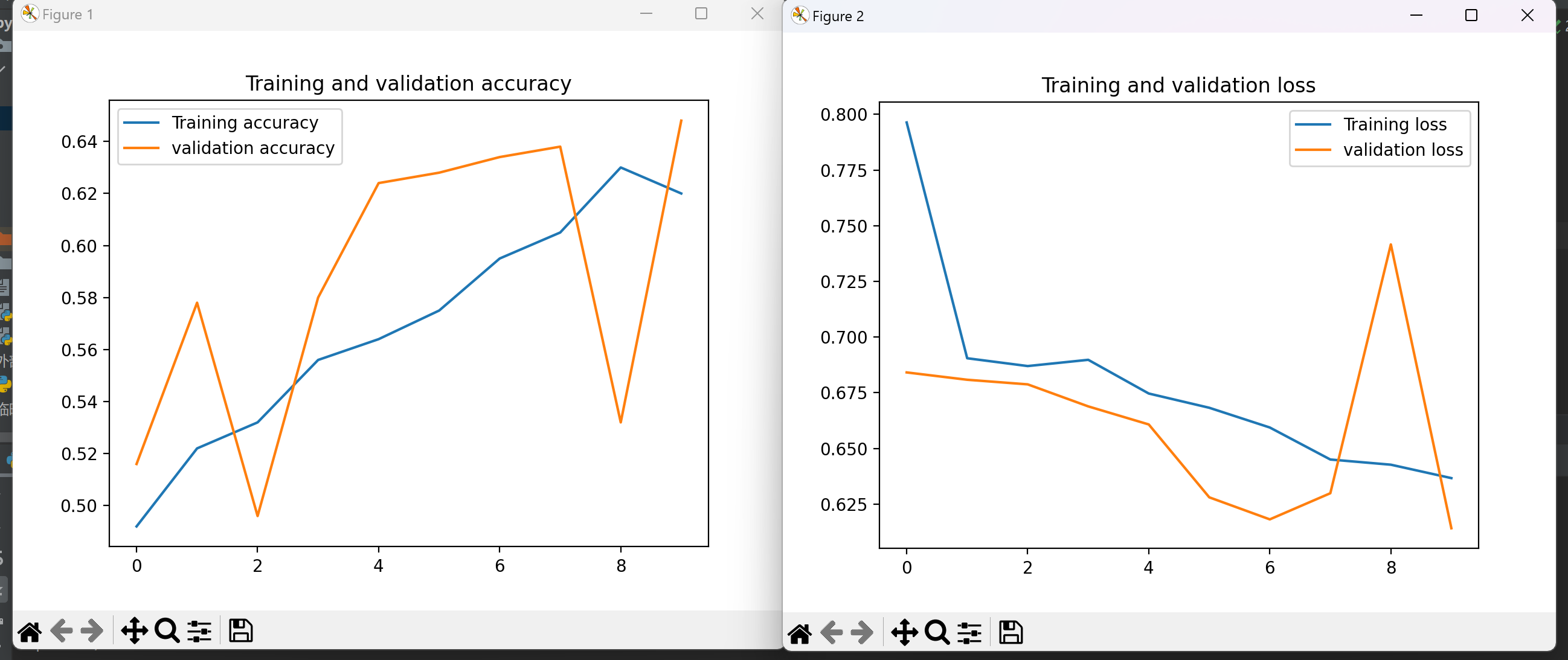


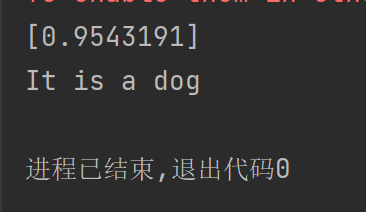




修改卷积核数量：

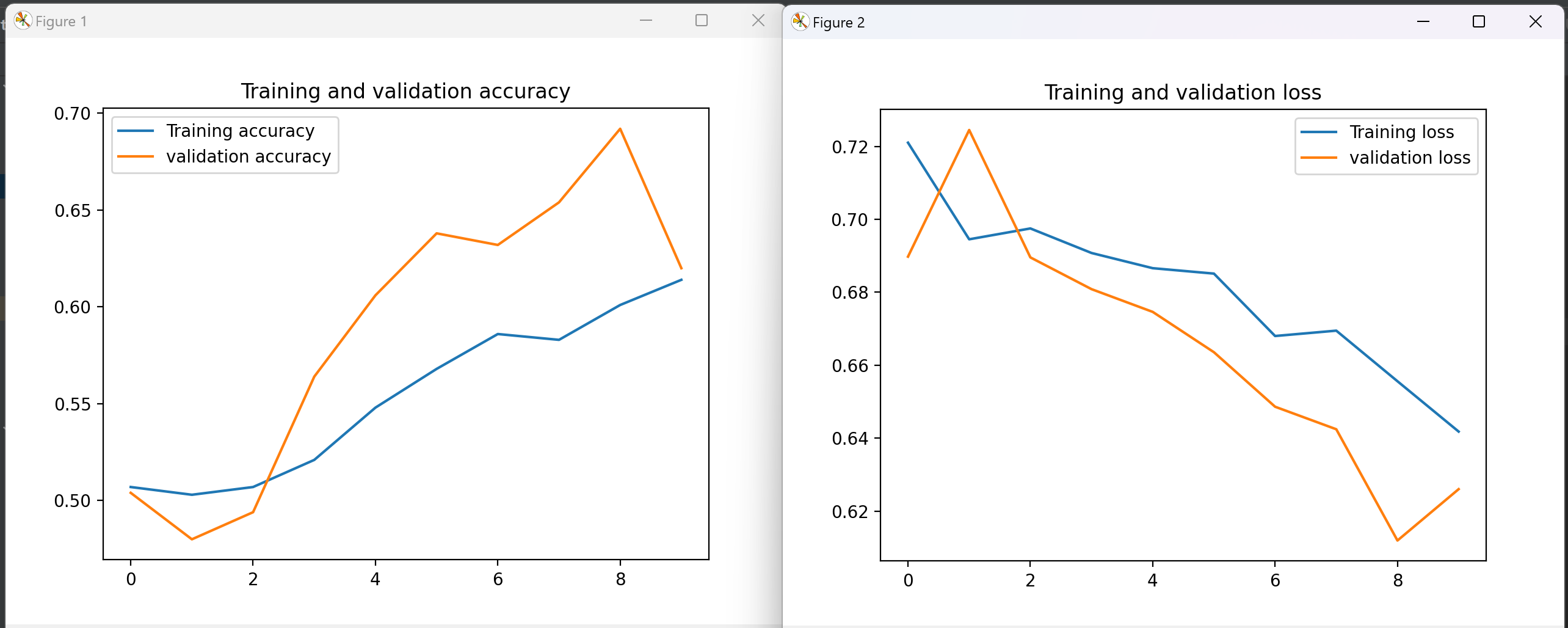


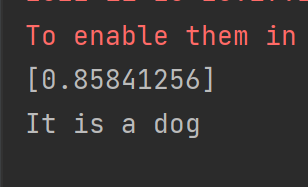




修改卷积核大小：







结论：

网络层数增加一层，对实验结果没有产生明显变化；增加两层，实验结果出现错误。

卷积核数量、修改卷积核大小的增加，模型的准确率少量上升，损失值少量下降。