《智能音视频课程设计》

课程设计报告书

**课程名称** 智能音视频课程设计

**设计题目** 猫狗分类

**专业班级** 人工智能191班

**学 号** 1520193079

**学生姓名** 何学长

**指导教师** 王碧

**起止日期** 2022年6月13日 至6月24日

**信息工程学院**

1. **课题（或系统）概述**（现状分析，系统目标等）

图像分类是计算机视觉领域的热门研究方向之一，也是实现物体检测、人脸识别、姿态估计等应用的重要基础，因此图像分类技术有很高的学术研究和科技应用价值。图像分类，即给定一幅输入图像，通过某种分类算法来判断该图像所属类别。图像分类的主要过程包括图像预处理、特征提取和分类器设计。图像预处理包括图像滤波，如中值滤波[7]、均值滤波、高斯滤波[9]以及图像归一化等操作，其主要作用是过滤图像中的一些无关信息，在简化数据的前提下最大限度地保留有用信息，增强特征提取的可靠性。特征提取是图像分类任务中最为关键的一部分，其将输入图像按照一定的规则变换生成另一种具有某些特性的特征表示，新的特征往往具有低维度、低冗余、低噪声、结构化等优点，从而降低了对分类器复杂度的要求，提高了模型性能。最后通过训练分类器对提取的特征进行分类，从而实现图像的分类。

本次课题主要聚焦在猫狗图像分类问题。大部分的猫狗分类问题仅是一个二分类问题，而这次做的工作是猫狗了软多分类问题，即不仅要是试别出是猫还是狗，还要准确知道是什么品种。这次用到的是在图像分类中常用的ResNet模型，并进行Web可视化展示。

**2.系统分析设计部分**（必需，重点）

(1)系统的功能

将训练好的模型部署到Web端口，并在Web上上传测试图片。当完成检 测后，输出软分类结果。

(2)可行性分析

目前神经网络模型已经很好的完成了图像分类问题，在ImageNet数据分 类问题上，甚至超过了人类的分辨能力。目前工业上也有大量的图像分类应 用：安全帽检测，口罩检测等待。

(3)需求分析

1、系统的功能性需求就是，可以正确的检测出图片中的物体是猫还是狗， 在猫狗同窗情况下也要准确试别区分出来。

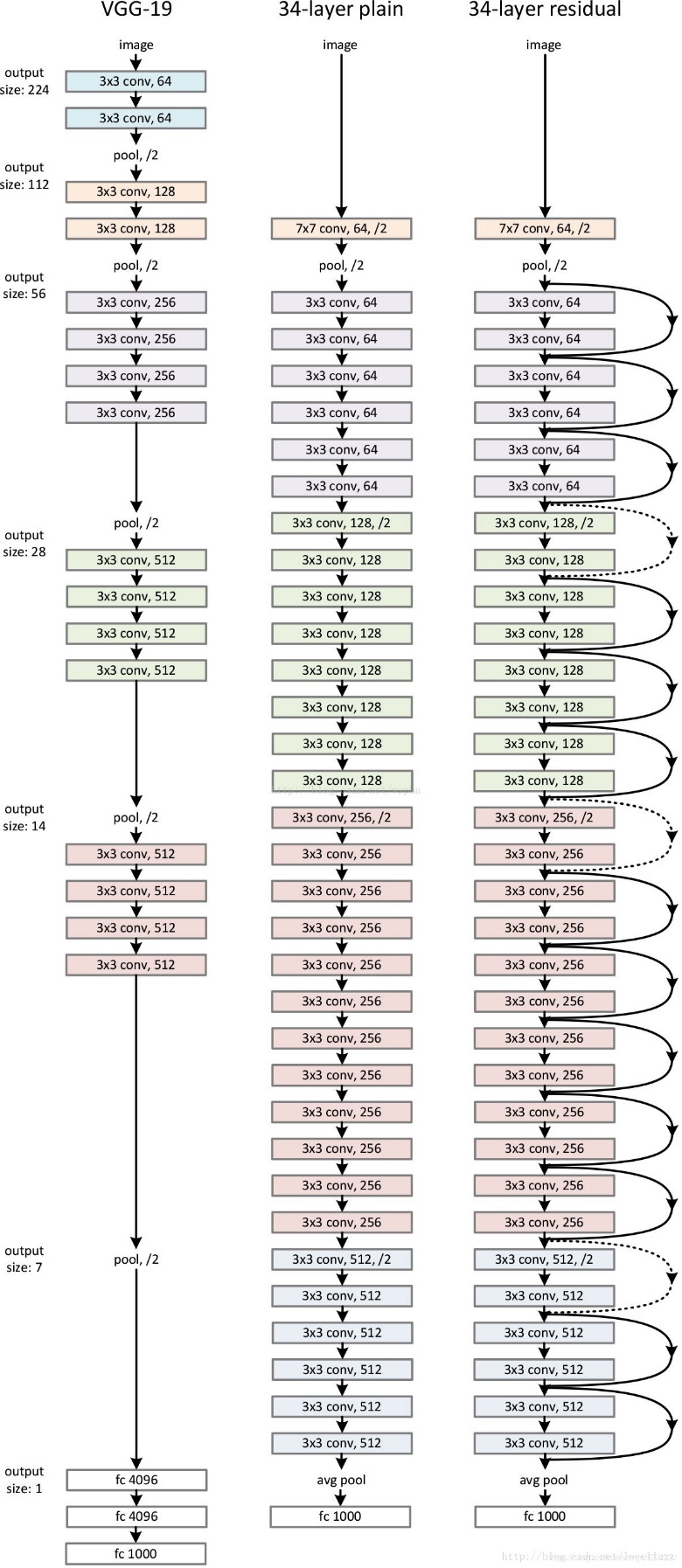
2、系统的性能需求。要求模型的检测既能保证准确率，又能保证速度。 要 求能在服务器上部署，且不占用网络资源。

3、系统的可移植性。系统要求预测模块能和系统中其他的模块很好地融 合在一起，那么算法就应该具备高可移植性。

4、系统的鲁棒性。考虑到实际中信息的复杂性，以及室外天气多变，这 些都是阻碍系统正常检测的因素。这就要求系统有一定的鲁棒性，能够在不 同场景和天气条件下正常检测。

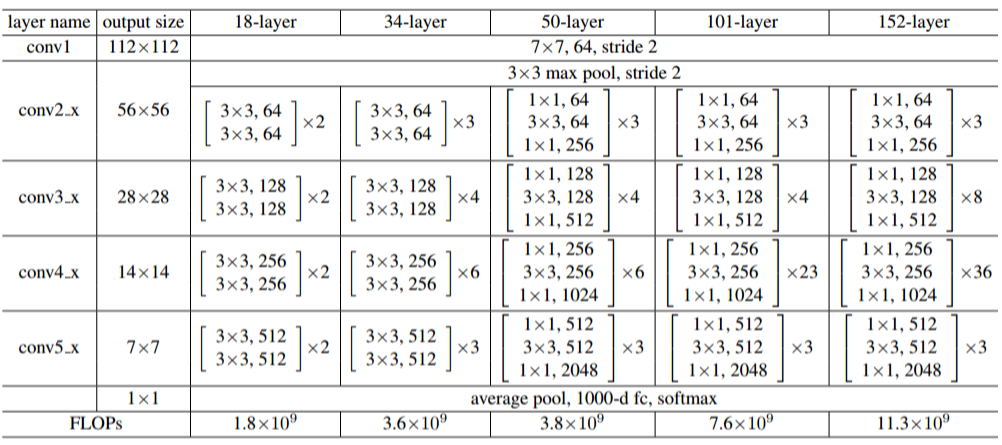
(4)概要设计，要求（重点）

①模型的网络结构和损失函数

这次猫狗分类任务使用了ResNet50模型，其网络结构为：

ResNet为多个Residual Block的串联，下面直观看一下ResNet-34与34-layer plain net和VGG的对比，以及堆叠不同数量Residual Block得到的不同ResNet。

ResNet结构非常容易修改和扩展，通过调整block内的channel数量以及堆叠的block数量，就可以很容易地调整网络的宽度和深度，来得到不同表达能力的网络，而不用过多地担心网络的“退化”问题，只要训练数据足够，逐步加深网络，就可以获得更好的性能表现。



从上表中可以看到50-layer和101layer。拿101-layer那列，我们先看看101-layer是不是真的是101层网络，首先有个输入7x7x64的卷积，然后经过3 + 4 + 23 + 3 = 33个building block，每个block为3层，所以有33 x 3 = 99层，最后有个fc层(用于分类)，所以1 + 99 + 1 = 101层，确实有101层网络；看到50-layer和101-layer这两列，可以发现，它们唯一的不同在于conv4\_x，ResNet50有6个block，而ResNet101有23个block，差了17个block，也就是17 x 3 = 51层。

Loss函数：

交叉熵损失函数.

②给出系统模块划分

本系统划分为图像分类与Web端展示两块.

③各功能模块的算法处理流程图及相关说明

图像分类中：主要介绍残差算法：

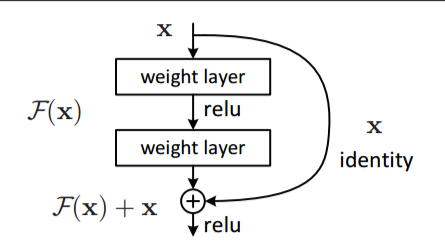
ResNet的主要功能

对浅层网络逐渐叠加layers，模型在训练集和测试集上的性能会变好，因为模型复杂度更高了，表达能力更强了，可以对潜在的映射关系拟合得更好。但是随着网络越来越深，效果并不会一直提升甚至会出现下降。这种在深层网络出现的问题叫做“退化”，其本质是网络的梯度随着网络越来越深会出现消失或者爆炸现象。

针对这样的问题有两种解决的思路：一种是调整求解方法，比如更好的初始化、更好的梯度下降算法等；另一种是调整模型结构，让模型更易于优化——改变模型结构实际上是改变了损失函数空间的形态。

ResNet的作者从后者入手，探求更好的模型结构。将堆叠的几层layer称之为一个block，对于某个block，其可以拟合的函数为F(x)，如果期望的潜在映射为H(x)，与其让F(x) 直接学习潜在的映射，不如去学习残差H(x)−x，即F(x):=H(x)−x，这样原本的前向路径上就变成了F(x)+x，用F(x)+x来拟合H(x)。作者认为这样可能更易于优化，因为相比于让F(x)学习成恒等映射，让F(x)学习成0要更加容易——后者通过L2正则就可以轻松实现。这样，对于冗余的block，只需F(x)→0就可以得到恒等映射，性能不减。

那么这个F(x)+x被设计成了如下的形式：



F(x)+x 构成的block称之为Residual Block，即残差块。多个相似的Residual Block串联构成ResNet。

个残差块有2条路径F(x)和x，F(x)路径拟合残差，不妨称之为残差路径，x路径为identity mapping恒等映射，称之为“shortcut”。图中的⊕为element-wise addition，要求参与运算的F(x)和x的尺寸要相同。

普通的平原网络与深度残差网络的最大区别在于，深度残差网络有很多旁路的支线将输入直接连到后面的层，使得后面的层可以直接学习残差，这些支路就叫做shortcut。传统的卷积层或全连接层在信息传递时，或多或少会存在信息丢失、损耗等问题。ResNet 在某种程度上解决了这个问题，通过直接将输入信息绕道传到输出，保护信息的完整性，整个网络则只需要学习输入、输出差别的那一部分，简化学习目标和难度。

shortcut连接被实验和研究了很久。Highway networks也使用了带有门函数的shortcut。但是这些门函数需要参数，而ResNet的shortcut不需要参数。而且当Highway networks的门函数的shortcut关闭时，相当于没有了残差函数，但是ResNet的shortcut一直保证学习残差函数。而且，当Highway networks的层数急剧增加时，没有表现出准确率的上升了。总之，ResNet可以看成是High

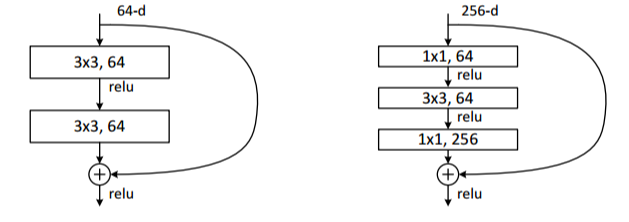
way networks的特例，但是从效果上来看，要比Highway networks好。

Residual Block的设计

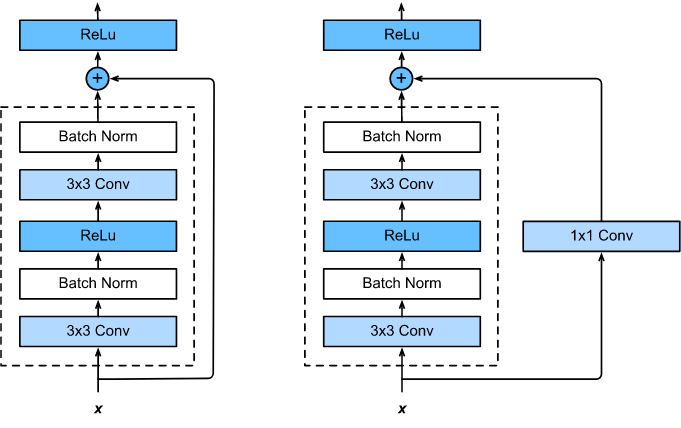
随之而来的问题是：1）残差路径如何设计？2）shortcut路径如何设计？3）Residual Block之间怎么连接？

在原论文中，残差路径可以大致分成2种，一种有bottleneck结构，即下图右中的1×1 卷积层，用于先降维再升维，主要出于降低计算复杂度的现实考虑，称之为“bottleneck block”，另一种没有bottleneck结构，如下图左所示，称之为“basic block”。basic block由2个3×3卷积层构成，bottleneck block由1×1。

如下图：



shortcut路径大致也可以分成2种，取决于残差路径是否改变了feature map数量和尺寸，一种是将输入x原封不动地输出，另一种则需要经过1×1卷积来升维 or/and 降采样，主要作用是将输出与F(x)路径的输出保持shape一致，对网络性能的提升并不明显，两种结构如下图所示：



残差网络的背后原理

残差块一个更通用的表示方式是：

现在我们先不考虑升维或者降维的情况，那么在上式子1中， 是直接映射， 是激活函数，一般使用ReLU。我们首先给出两个假设：假设1： [公式] 是直接映射；假设2： [公式] 是直接映射。

那么这时候残差块可以表示为：

对于一个更深的层，其与层的关系可以表示为:

这个公式反应了残差网络的两个属性：

1）层可以表示为任意一个比它浅的层和他们之间的残差部分之和；

2），，是各个残差块特征的单位累和，而MLP是特征矩阵的累积。

根据BP中使用的导数的链式法则，损失函数关于的梯度可以表示为:

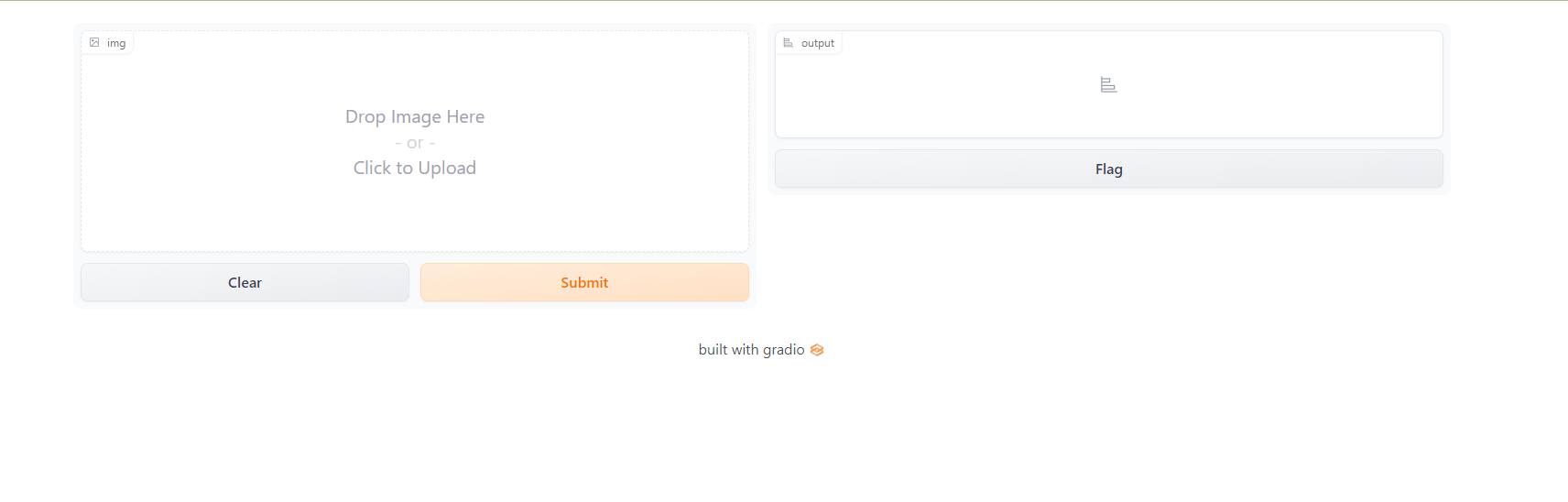
上面公式反映了残差网络的两个属性：

1. 在整个训练过程中，不可能一直为-1，也就是说残差网络中不会出现梯度消失问题。
2. 表示层的的梯度可以直接传递到任何一个比它浅的层。

Web端部署

本次课程项目的部署通过使用Gradio库来完成，直接将训练好的网络权重在本地实例化，然后写好输入测试的图像尺寸和最后输出类别的个数即可完成网络部署。

Web端的界面为：



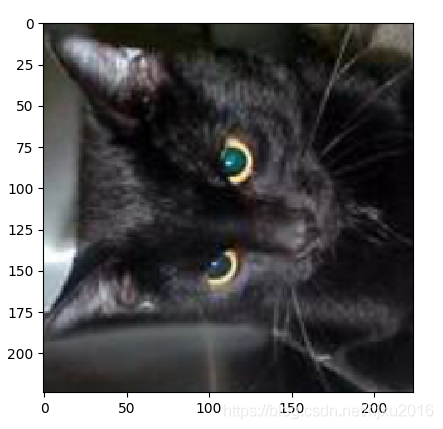
左边点击选择要输入的图片，然后点击Submit即可。如果等待过久或者选择其他图片测试点击Clear按键即可。右边是输出的分类结果。

数据集

本次训练使用的数据集为ImageNet上预训练过的ResNet50的网络权重，然后在自己收集的数据集上在训练。数据集为：猫狗大战。从微软官方下载获得。

介绍

以著名的猫狗大战数据集为例,实战多种分类网络。数据分布如下,在data文件夹下,分成 train和validation两个文件夹。数据样例展示：



(5)详细设计，列出算法的网络结构和核心代码，代码如果比较长，可以附录在后面

(6)数据库设计部分（非必需，如果使用了数据库，则必须给出，如果没有使用数据库，则此部分可以不要）

**3.系统测试部分**（必需）

根据已知的输入，预测应该输出的结果，并实际运行，对比分析，找出错误所在，并改正。

能够正确根据已知的输入，遇到到应该出现的结果，无出现的错误。结果在系统操作演示部分展示。

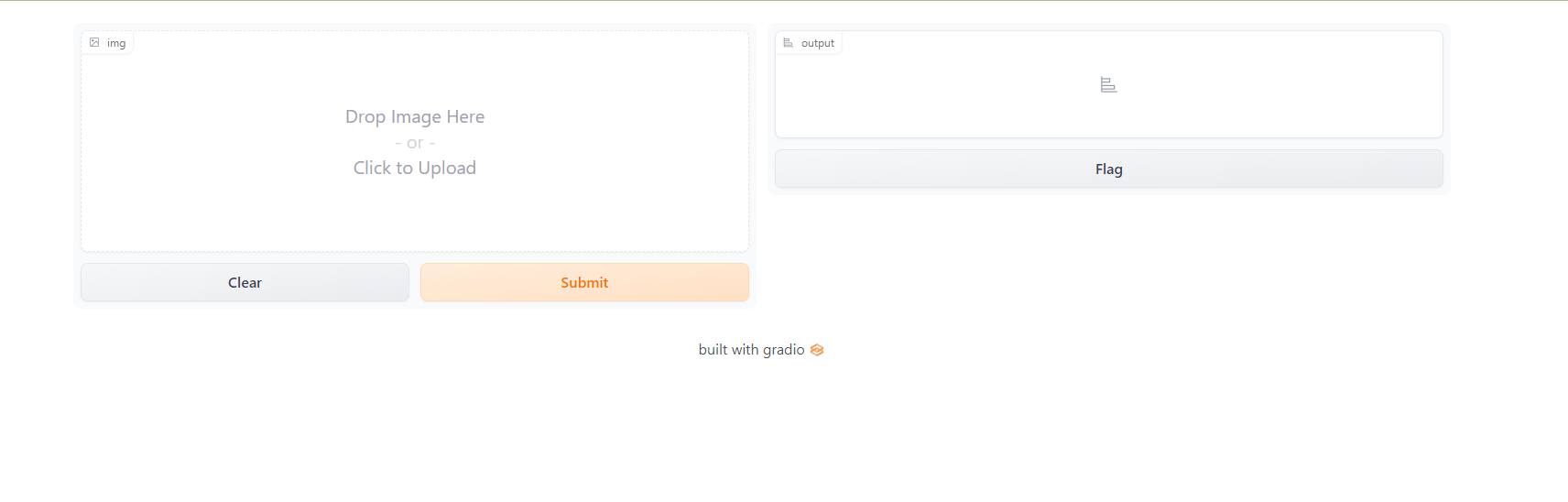
**4.系统操作演示部分**（必需）

(1)数据的输入操作

打开Web界面直接使用，当网络情况或者图片较为复杂时，需要一些时间等待。输入的图片尺寸没有限制。

首先打开Web端网站——https://34989.gradio.app/。

进去可以看到如下界面:

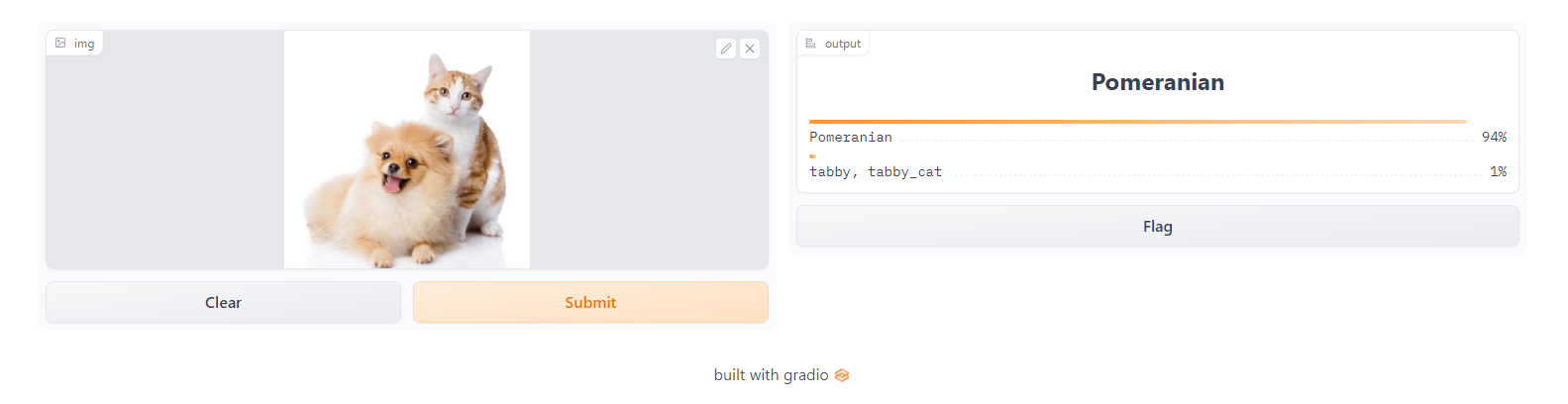


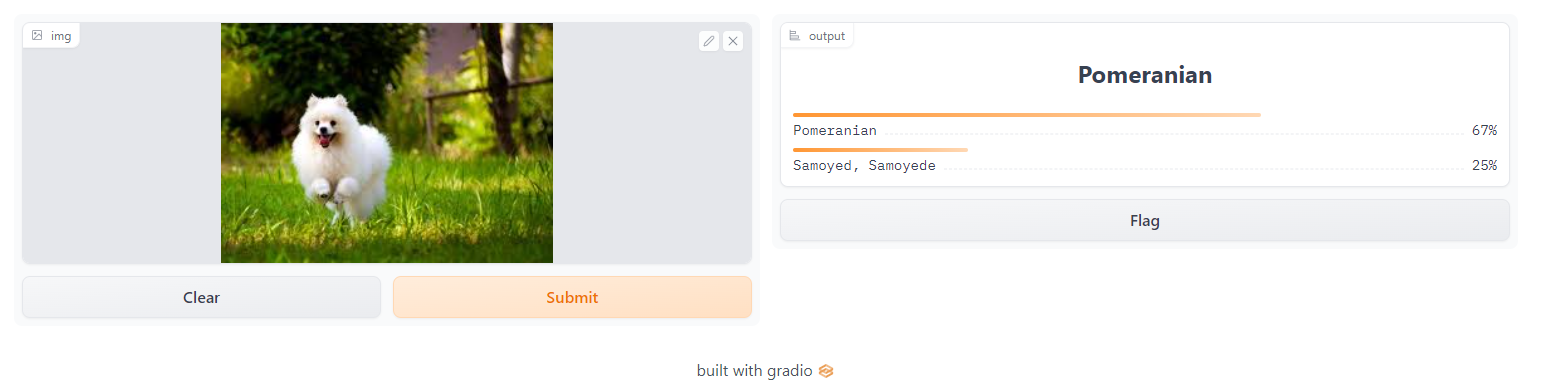
测试数据为：



(2)数据的输出操作（可视化效果）







(3)结果分析

通过上面的测试结果可以知道，模型可以分辨出猫和狗，尽管在同窗的情况下也可以识别出。但是模型对于具体是哪一类猫或狗上效果还是有点欠缺。

**5.系统小结**（必需）

(1)程序设计过程中遇到的问题及解决过程

在制作web展示的时候因为会写前端代码，没办法展示。不过直接用Gradio迅速解决并完成了Web部署。

(2)课程设计心得及体会

这次课程设计，独立的完成了一次从任务选题，到任务分析的过程。在这期间提高了我对于本专业知识的理解。在这次Web端网络部署上，学到了新的知识，提高了自己的业务能力。

**6.附源程序**

RsetNet50：

import torch.optim as optim

import torch

import torch.nn as nn

import torch.nn.parallel

import torch.optim

import torch.utils.data

import torch.utils.data.distributed

import torchvision.transforms as transforms

import torchvision.datasets as datasets

import torchvision.models

from effnetv2 import effnetv2\_s

from torch.autograd import Variable

# 设置超参数

BATCH\_SIZE = 16

EPOCHS = 10

DEVICE = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

# 数据预处理

transform = transforms.Compose([

transforms.Resize((128, 128)),

# transforms.RandomVerticalFlip(),

# transforms.RandomCrop(50),

# transforms.ColorJitter(brightness=0.5, contrast=0.5, hue=0.5),

transforms.ToTensor(),

transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])

])

transform\_test = transforms.Compose([

transforms.Resize((128, 128)),

transforms.ToTensor(),

transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])

])

# 读取数据

dataset\_train = datasets.ImageFolder('data/train', transform)

print(dataset\_train.imgs)

# 对应文件夹的label

print(dataset\_train.class\_to\_idx)

dataset\_test = datasets.ImageFolder('data/val', transform\_test)

# 对应文件夹的label

print(dataset\_test.class\_to\_idx)

# 导入数据

train\_loader = torch.utils.data.DataLoader(dataset\_train, batch\_size=BATCH\_SIZE, shuffle=True)

test\_loader = torch.utils.data.DataLoader(dataset\_test, batch\_size=BATCH\_SIZE, shuffle=False)

modellr = 1e-4

# 实例化模型并且移动到GPU

criterion = nn.CrossEntropyLoss()

# model = effnetv2\_s()

# num\_ftrs = model.classifier.in\_features

# model.classifier = nn.Linear(num\_ftrs, 2)

model = torchvision.models.resnet18(pretrained=False)

num\_ftrs = model.fc.in\_features

model.fc = nn.Linear(num\_ftrs, 2)

model.to(DEVICE)

# 选择简单暴力的Adam优化器，学习率调低

optimizer = optim.Adam(model.parameters(), lr=modellr)

def adjust\_learning\_rate(optimizer, epoch):

"""Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""

modellrnew = modellr \* (0.1 \*\* (epoch // 50))

print("lr:", modellrnew)

for param\_group in optimizer.param\_groups:

param\_group['lr'] = modellrnew

# 定义训练过程

def train(model, device, train\_loader, optimizer, epoch):

model.train()

sum\_loss = 0

total\_num = len(train\_loader.dataset)

print(total\_num, len(train\_loader))

for batch\_idx, (data, target) in enumerate(train\_loader):

data, target = Variable(data).to(device), Variable(target).to(device)

output = model(data)

loss = criterion(output, target)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

print\_loss = loss.data.item()

sum\_loss += print\_loss

if (batch\_idx + 1) % 50 == 0:

print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

epoch, (batch\_idx + 1) \* len(data), len(train\_loader.dataset),

100. \* (batch\_idx + 1) / len(train\_loader), loss.item()))

ave\_loss = sum\_loss / len(train\_loader)

print('epoch:{},loss:{}'.format(epoch, ave\_loss))

def val(model, device, test\_loader):

model.eval()

test\_loss = 0

correct = 0

total\_num = len(test\_loader.dataset)

print(total\_num, len(test\_loader))

with torch.no\_grad():

for data, target in test\_loader:

data, target = Variable(data).to(device), Variable(target).to(device)

output = model(data)

loss = criterion(output, target)

\_, pred = torch.max(output.data, 1)

correct += torch.sum(pred == target)

print\_loss = loss.data.item()

test\_loss += print\_loss

correct = correct.data.item()

acc = correct / total\_num

avgloss = test\_loss / len(test\_loader)

print('\nVal set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(

avgloss, correct, len(test\_loader.dataset), 100 \* acc))

# 训练

for epoch in range(1, EPOCHS + 1):

adjust\_learning\_rate(optimizer, epoch)

train(model, DEVICE, train\_loader, optimizer, epoch)

val(model, DEVICE, test\_loader)

torch.save(model, 'model.pth')

Web端部署：

import requests

import gradio as gr

import torch

import torchvision

from timm import create\_model

from timm.data import resolve\_data\_config

from timm.data.transforms\_factory import create\_transform

= "https://storage.googleapis.com/bit\_models/ilsvrc2012\_wordnet\_lemmas.txt"

LABELS = requests.get(IMAGENET\_1k\_URL).text.strip().split('\n')

model = create\_model('resnet50', pretrained=True)

transform = create\_transform(

\*\*resolve\_data\_config({}, model=model)

)

model.eval()

def predict\_fn(img):

img = img.convert('RGB')

img = transform(img).unsqueeze(0)

with torch.no\_grad():

out = model(img)

probabilites = torch.nn.functional.softmax(out[0], dim=0)

values, indices = torch.topk(probabilites, k=2)