基于CNN的地形识别

一、实验内容

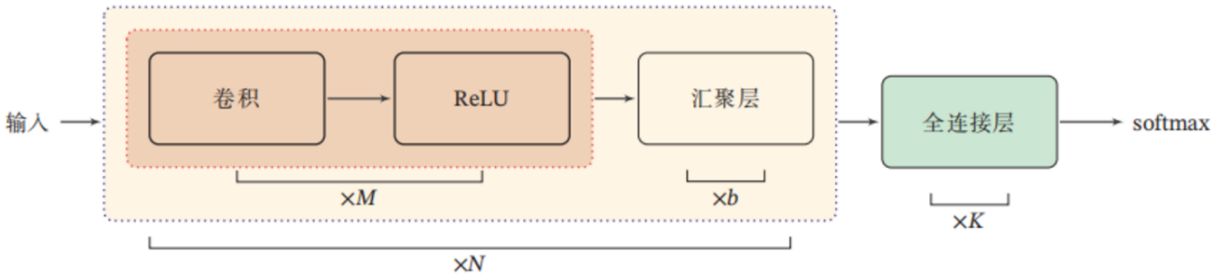
1. 使用mindspore深度学习框架搭建CNN模型，完成地形的多分类任务
2. 完成数据预处理，实现训练集与验证集的划分
3. 使用带有正则化的CNN网络，在测试集上准确率达到97%

二、实验原理

2.1 卷积神经网络

卷积神经网络（Convolutional Neural Network，CNN或ConvNet）一般是由卷积层、汇聚层和全连接层交叉堆叠而成的前馈神经网络．全连接层一般在卷积网络的最顶层． 卷积神经网络有三个结构上的特性：**局部连接**、**权重共享**以及**汇聚**．这些特性使得卷积神经网络具有一定程度上的平移、缩放和旋转不变性．和前馈神经网络相比，卷积神经网络的参数更少．

卷积神经网络主要使用在图像和视频分析的各种任务（比如图像分类、人脸识别、物体识别、图像分割等）上，其准确率一般也远远超出了其他的神经网络模型．本次实验选用卷积神经网络实现地形的分类任务。



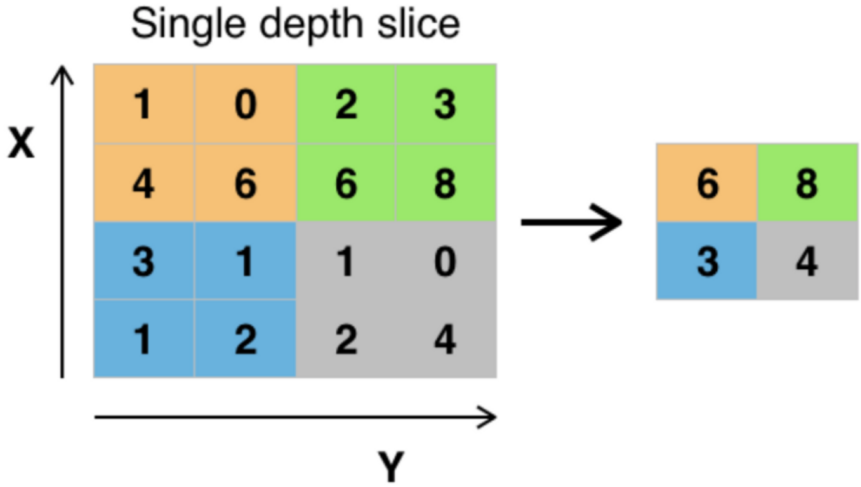
2.2 卷积层

卷积层的作用是提取一个局部区域的特征，不同的卷积核相当于不同的特征提取器．上一节中描述的卷积层的神经元和全连接网络一样都是一维结构．由于卷积网络主要应用在图像处理上，而图像为二维结构，因此为了更充分地利用图像的局部信息，通常将神经元组织为三维结构的神经层，其大小为高度M×宽度N×深度D，由D 个$M \times N$ 大小的特征映射构成.

特征映射（Feature Map）为一幅图像（或其他特征映射）在经过卷积提取到的特征，每个特征映射可以作为一类抽取的图像特征．为了提高卷积网络的表示能力，可以在每一层使用多个不同的特征映射，以更好地表示图像的特征．在输入层，特征映射就是图像本身．如果是灰度图像，就是有一个特征映射，输入层的深度 $D = 1$；如果是彩色图像，分别有 RGB 三个颜色通道的特征映射，输入层的深度$D = 3$．

2.3 池化层

它实际上是一种形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。直觉上，这种机制能够有效地原因在于，在发现一个特征之后，它的精确位置远不及它和其他特征的相对位置的关系重要。池化层会不断地减小数据的空间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了过拟合。通常来说，CNN的卷积层之间都会周期性地插入池化层



池化操作后的结果相比其输入缩小了。池化层的引入是仿照人的视觉系统对视觉输入对象进行降维和抽象。在卷积神经网络过去的工作中，研究者普遍认为池化层有如下三个功效： 1.**特征不变性**：池化操作是模型更加关注是否存在某些特征而不是特征具体的位置。其中不变形性包括，平移不变性、旋转不变性和尺度不变性。平移不变性是指输出结果对输入对小量平移基本保持不变，例如，输入为(1, 5, 3), 最大池化将会取5，如果将输入右移一位得到(0, 1, 5)，输出的结果仍将为5。对伸缩的不变形，如果原先的神经元在最大池化操作后输出5，那么经过伸缩（尺度变换）后，最大池化操作在该神经元上很大概率的输出仍是5. 2.**特征降维（下采样）**：池化相当于在空间范围内做了维度约减，从而使模型可以抽取更加广范围的特征。同时减小了下一层的输入大小，进而减少计算量和参数个数。 3.在一定程度上防止过拟合，更方便优化。 4.实现非线性（类似relu）。 5.扩大感受野。

2.4 Adam优化器

2014年12月，Kingma和Lei Ba两位学者提出了Adam优化器，结合AdaGrad和RMSProp两种优化算法的优点。对梯度的一阶矩估计（First Moment Estimation，即梯度的均值）和二阶矩估计（Second Moment Estimation，即梯度的未中心化的方差）进行综合考虑，计算出更新步长。

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

主要包含以下几个显著的优点：

1. 实现简单，计算高效，对内存需求少
2. 参数的更新不受梯度的伸缩变换影响
3. 超参数具有很好的解释性，且通常无需调整或仅需很少的微调
4. 更新的步长能够被限制在大致的范围内（初始学习率）
5. 能自然地实现步长退火过程（自动调整学习率）
6. 很适合应用于大规模的数据及参数的场景
7. 适用于不稳定目标函数
8. 适用于梯度稀疏或梯度存在很大噪声的问题

综合Adam在很多情况下算作默认工作性能比较优秀的优化器。

三、实验步骤

3.1 环境配置

PyCharm 2020.1.1 x64

MindSpore

Python3.7

3.2 实验环境导入

定义模型封装信息：

```
cfg = edict({
    'data_path': 'data',
    'data_size':4020,
    'image_width': 100,  # 图片宽度
    'image_height': 100,  # 图片高度
    'batch_size': 32,
```

```
    'channel': 3,  # 图片通道数
    'num_class':7,  # 分类类别
    'weight_decay': 0.01,
    'lr':0.0001,  # 学习率
    'dropout_ratio': 0.5,
    'epoch_size': 400,  # 训练次数
    'sigma':0.01,

    'save_checkpoint_steps': 1,  # 多少步保存一次模型
    'keep_checkpoint_max': 1,  # 最多保存多少个模型
    'output_directory': './',  # 保存模型路径
    'output_prefix': "checkpoint_classification"  # 保存模型文件名字
```

3.3 数据集的获取与预处理

hdf5是一种存储相同类型数值的大数组的机制，适用于可被层次性组织且数据集需要被元数据标记的数据模型。本次实验所使用的数据集就是hdf5格式的。由于对此种文件类型的数据集不熟悉，此处选择将hdf5数据文件转换为png格式，便于后续处理。

```
import h5py
import os
from PIL import Image
import numpy as np
hdf5_path = r'./testSet_c7_ver_2.hdf5'  # 读取路径
store_path = r'./data'  # 存储路径
HDF5File = h5py.File(hdf5_path)
if not os.path.exists(store_path):
    os.mkdir(store_path)
images = HDF5File['images']['images'][:]
images = images.reshape(images.shape[0], 224, 224, 3)
labels=HDF5File['labels']['labels'][:]
label_names=['asphalt','grass','gravel','pavement','sand','brick','coated floor']
for i in range(images.shape[0]):
    img=images[i] # 表示第i张图片
    label=labels[i] # 表示第i张图片的标签
    label_name=label_names[label]
    path=store_path+'/'+label_name
    if not os.path.exists(path):
        os.mkdir(path)
    name=os.path.join(path,str(i)+'.png')
    img1=Image.fromarray(np.uint8(img))
    img1.convert('RGB').save(name)
    print('img'+str(i)+'is done!')
```

最终得到png格式的数据集。该数据集总共包括7种地形信息的类型：分别是asphalt（沥青，656张），grass（草，604张），gravel（砾石，513张），pavement（向日葵，629张），sand（沙子，799张），brick（砖，557张），coated floor（涂层地板，619张）。保存在7个文件夹当中，总共4020张。

按照实验要求，我们将数据集按照1：1的比例划分为训练数据集和测试数据集。

```
(de_train,de_test)=de_dataset.split([0.5,0.5])
#设置每个批处理的行数
#drop_remainder确定是否删除最后一个可能不完整的批（default=False）。
#如果为True，并且如果可用于生成最后一个批的batch_size行小于batch_size行，则这些行将被删除，并且不会传播到子节点。
de_train=de_train.batch(cfg.batch_size, drop_remainder=True)
#重复此数据集计数次数。
de_test=de_test.batch(cfg.batch_size, drop_remainder=True)
print('训练数据集数量：',de_train.get_dataset_size()*cfg.batch_size)#get_dataset_size()获取批处理的大小。
print('测试数据集数量：',de_test.get_dataset_size()*cfg.batch_size)

data_next=de_dataset.create_dict_iterator(output_numpy=True).__next__()
print('通道数/图像长/宽：', data_next['image'].shape)
print('一张图像的标签样式：', data_next['label'])  # 一共7类，用0-6的数字表达类别。
```

3.4 构建CNN图像识别模型

```
class Identification_Net(nn.Cell):
    def __init__(self, num_class=7,channel=3,dropout_ratio=0.5,trun_sigma=0.01):  # 一共分7类，图片通道数是3
        super(Identification_Net, self).__init__()
        self.num_class = num_class
        self.channel = channel
        self.dropout_ratio = dropout_ratio
        #设置卷积层
        self.conv1 = nn.Conv2d(self.channel, 32,
                               kernel_size=5, stride=1, padding=0,
                               has_bias=True, pad_mode="same",

weight_init=TruncatedNormal(sigma=trun_sigma),bias_init='zeros')
        #设置ReLU激活函数
        self.relu = nn.ReLU()
        #设置最大池化层
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2,pad_mode="valid")
        self.conv2 = nn.Conv2d(32, 64,
                               kernel_size=5, stride=1, padding=0,
                               has_bias=True, pad_mode="same",

weight_init=TruncatedNormal(sigma=trun_sigma),bias_init='zeros')
        self.conv3 = nn.Conv2d(64, 128,
                               kernel_size=3, stride=1, padding=0,
                               has_bias=True, pad_mode="same",

weight_init=TruncatedNormal(sigma=trun_sigma),bias_init='zeros')
        self.conv4 = nn.Conv2d(128, 128,
                               kernel_size=3, stride=1, padding=0,
                               has_bias=True, pad_mode="same",
                               weight_init=TruncatedNormal(sigma=trun_sigma),
bias_init='zeros')
        self.flatten = nn.Flatten()
```

```
        self.fc1 = nn.Dense(6*6*128, 1024,weight_init=TruncatedNormal(sigma=trun_sigma),bias_init = 0.1)
        self.dropout = nn.Dropout(self.dropout_ratio)
        self.fc2 = nn.Dense(1024, 512, weight_init=TruncatedNormal(sigma=trun_sigma), bias_init=0.1)
        self.fc3 = nn.Dense(512, self.num_class, weight_init=TruncatedNormal(sigma=trun_sigma), bias_init=0.1)
    #构建模型
    def construct(self, x):
        x = self.conv1(x)
        #print(x.shape)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv3(x)
        x = self.max_pool2d(x)
        x = self.conv4(x)
        x = self.max_pool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        #print(x.shape)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        return x
net=Identification_Net(num_class=cfg.num_class, channel=cfg.channel, dropout_ratio=cfg.dropout_ratio)
#计算softmax交叉熵。
net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
#opt
fc_weight_params = list(filter(lambda x: 'fc' in x.name and 'weight' in x.name, net.trainable_params()))
other_params=list(filter(lambda x: 'fc' not in x.name or 'weight' not in x.name, net.trainable_params()))
group_params = [{'params': fc_weight_params, 'weight_decay': cfg.weight_decay},
                {'params': other_params},
                {'order_params': net.trainable_params()}]
#设置Adam优化器
net_opt = nn.Adam(group_params, learning_rate=cfg.lr, weight_decay=0.0)
#net_opt = nn.Adam(params=net.trainable_params(), learning_rate=cfg.lr, weight_decay=0.1)

model = Model(net, loss_fn=net_loss, optimizer=net_opt, metrics={"acc"})
loss_cb = LossMonitor(per_print_times=de_train.get_dataset_size()*10)
config_ck = CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps,
                             keep_checkpoint_max=cfg.keep_checkpoint_max)
ckpoint_cb = ModelCheckpoint(prefix=cfg.output_prefix, directory=cfg.output_directory, config=config_ck)
```

```
print("============== Starting Training ==============")
model.train(cfg.epoch_size, de_train, callbacks=[loss_cb, ckpoint_cb],
dataset_sink_mode=True)
```

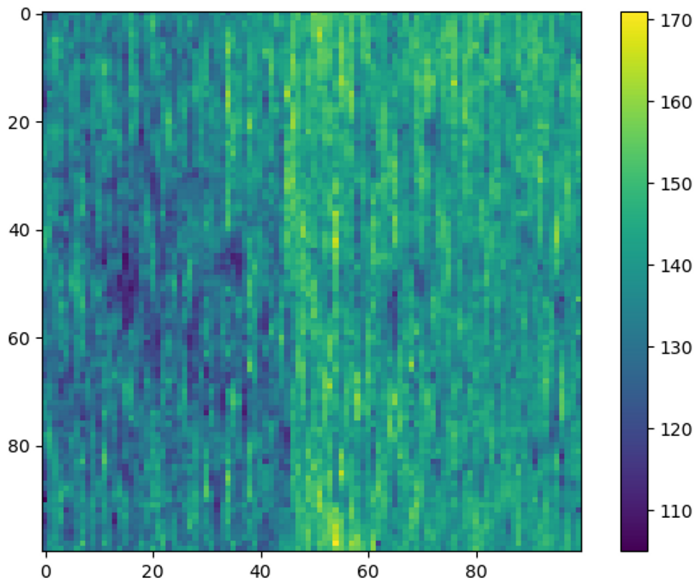
3.5 图像分类模型验证

```
#加载模型
import os
CKPT = os.path.join(cfg.output_directory,cfg.output_prefix+'-
'+str(cfg.epoch_size)+'_'+str(de_train.get_dataset_size())+'.ckpt')
net = Identification_Net(num_class=cfg.num_class, channel=cfg.channel,
dropout_ratio=cfg.dropout_ratio)
load_checkpoint(CKPT, net=net)
model = Model(net)
#验证推理
# 预测
class_names =
{0:'asphalt',1:'grass',2:'gravel',3:'pavement',4:'sand',5:'brick',6:'coated floor'}
test_ = de_test.create_dict_iterator().__next__()
test = Tensor(test_['image'], mindspore.float32)
predictions = model.predict(test)
predictions = predictions.asnumpy()
true_label = test_['label'].asnumpy()
```

四、实验结果

1. 封装信息

```
训练数据集数量:  1984
测试数据集数量:  1984
通道数/图像长/宽:  (3, 100, 100)
一张图像的标签样式:  3
```

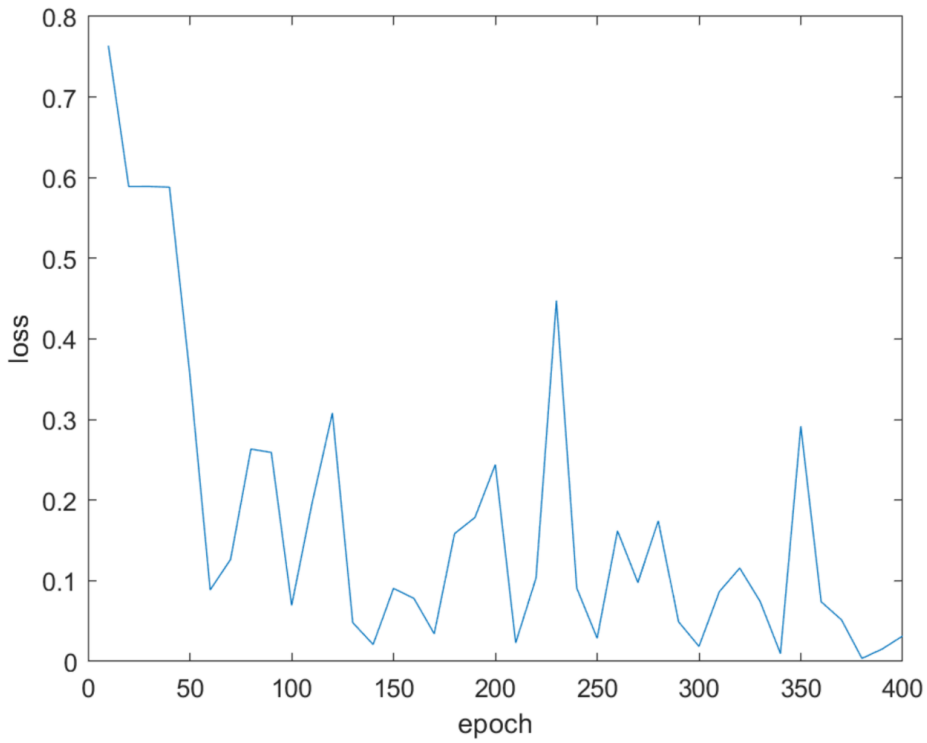


2.训练结果

```
============== Starting Training ==============
epoch: 10 step: 62, loss is 0.7637209296226501
epoch: 20 step: 62, loss is 0.5890305638313293
epoch: 30 step: 62, loss is 0.589104175567627
epoch: 40 step: 62, loss is 0.5882859230041504
epoch: 50 step: 62, loss is 0.3559538722038269
epoch: 60 step: 62, loss is 0.08863288909196854
epoch: 290 step: 62, loss is 0.04904380815241005
epoch: 300 step: 62, loss is 0.0186240803450346
epoch: 310 step: 62, loss is 0.086481973528862
epoch: 320 step: 62, loss is 0.11573179066181183
epoch: 330 step: 62, loss is 0.07449150085449219
epoch: 340 step: 62, loss is 0.009722127579152584
epoch: 350 step: 62, loss is 0.2916003167629242
epoch: 360 step: 62, loss is 0.07396327704191208
epoch: 370 step: 62, loss is 0.05147292837500572
epoch: 380 step: 62, loss is 0.003737802617251873
epoch: 390 step: 62, loss is 0.015404091216623783
epoch: 400 step: 62, loss is 0.03149497136473656
{'acc': 0.9712701612903226}
```

由数据可知，最终模型准确率'acc':为0.9712701612903226

使用matlab绘图，得到loss函数随epoch的变化图象如下：



由图像可以很直观地看出，在60epoch时loss函数已经能下降到0.1左右，但是收敛情况不佳，存在较大振荡。

3.预测效果

```
第0个sample预测结果:  gravel    真实结果:  gravel
第1个sample预测结果:  grass    真实结果:  grass
第2个sample预测结果:  grass    真实结果:  grass
第3个sample预测结果:  asphalt    真实结果:  asphalt
第4个sample预测结果:  gravel    真实结果:  gravel
第5个sample预测结果:  coated floor    真实结果:  coated floor
第6个sample预测结果:  brick    真实结果:  brick
第7个sample预测结果:  brick    真实结果:  brick
第8个sample预测结果:  asphalt    真实结果:  asphalt
第9个sample预测结果:  coated floor    真实结果:  coated floor
```

在训练好的模型上，我们随机抽取十张图片进行类别预测，最终实际结果与预测结果一致。

五、实验总结与分析

1.损失函数的振荡问题

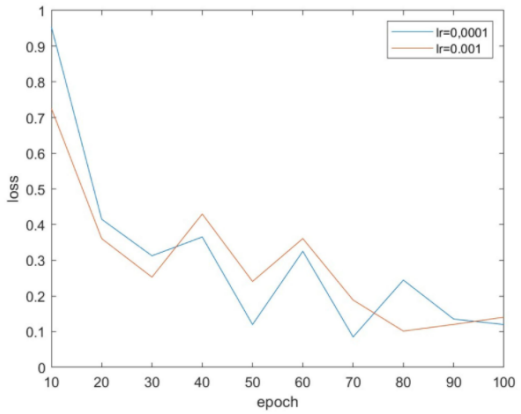
由实验中的损失函数随epoch的变化图像可以明显看出，损失函数在下降时存在较大的振荡。

- **更改学习率**

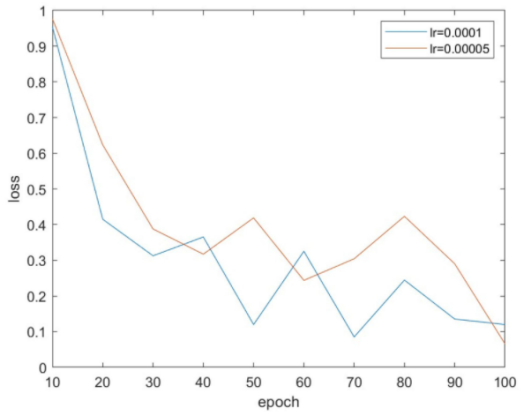
> In machine learning and statistics, the learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.
>
> 在机器学习和统计学中，学习率是优化算法中的调谐参数，该参数可确定确定每次迭代中的步长，使损失函数收敛到最小值

学习率设置**过小**的时候，每步太小，下降速度太慢，可能要花很长的时间才会找到最小值。学习率过大**过大**的时候，每步太大，虽然收敛得速度很快，跨过或忽略了最小值，导致一直来回震荡而无法收敛。

	学习率	**准确率**
原学习率	0.0001	0.9581653225806451
增大学习率	0.001	0.952116935483871
减小学习率	0.00005	0.90625



在学习率调大后，可以明显发现loss的收敛速度变快，振荡情况有所改善



在学习率调小后，loss下降速度较慢，对振荡情况的改进不明显。但模型分类的准确率明显降低。

综上所述，改变学习率在此模型中，对loss的振荡情况优化不明显

- **改动batch_size**

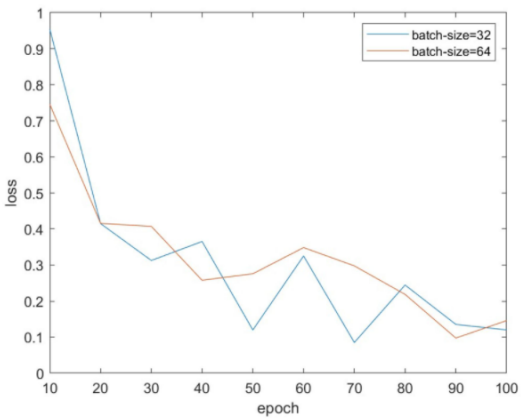
BATCH_SIZE:即一次训练所抓取的数据样本数量.

设置batch_size的优势： **可以减少内存的使用**，因为我们每次部分数据，因此训练时所使用的内存量会比较小。这对于我们的电脑内存不能满足一次性训练所有数据时十分有效。可以理解为训练数据集的分块训练.**提高训练的速度**，因为每次完成训练后我们都会更新我们的权重值使其更趋向于精确值。所以完成训练的速度较快。

	batch size	acc
原本	32	0.958165322
增大	64	0.947580645

	batch size	acc
原本	32	0.958165322
增大	64	0.947580645

增大batch size后，观察右图可以明显看出loss的振荡明显减小。故而增大batch size对loss的振荡有抑制作用



- **通道数改进**

观察数据集可以发现，数据集中的图像多为黑白灰。为了增强特征，可以将通道数改为1。可以使模型更为简单。

2.总结与反思

通过本次实验提高了编程能力，对神经网络图像分类的流程有了进一步的了解。同时积累了数据增强，使用dropout避免过拟合的知识。虽然模型在地形分类的任务中分类准确率达到了97%以上，但是loss函数的振荡问题仍然存在。今后在面对更复杂的问题时可以考虑选择更合适的优化器，并且增加合适的正则化方法。