**2020机器学习**

**印花疵点分类**

**组员：**

须子逸 20210240350

李帅 20210240207

李锦程 20210240213

朱柏霖 20210240022

郭露 20210240321

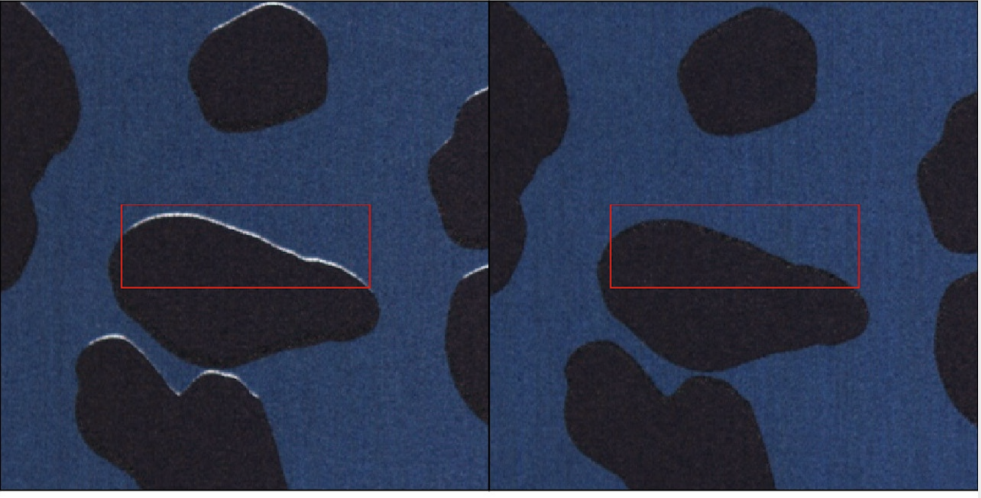
# 1 任务描述

基于机器学习模型对印花疵点数据集中标签为1、2、14的数据以及标签为1、2、4、14、20的数据分别做3分类及5分类任务；

利用ModelArts平台进行模型训练任务。

# 2 数据集介绍

数据集内印花布匹瑕疵被划分为15类，瑕疵以成对图片(正确的图样以及问题图样)给出，并注明了瑕疵的参考位置及类别。下述瑕疵图、模板图及瑕疵参考位置均可作为已知信息用于瑕疵类型的判别。



逃花缺陷（左），正确的印花图样（右）

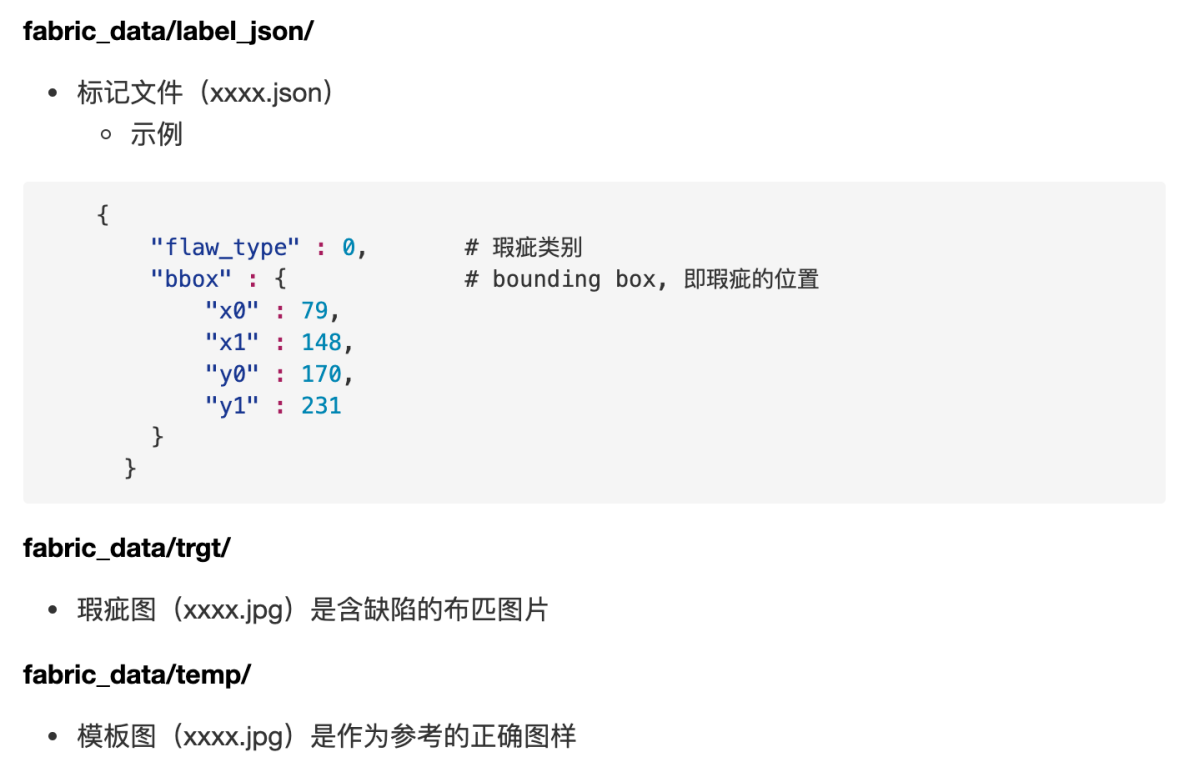


塞网缺陷（左），正确的印花图样（右）

经整理，布匹瑕疵的类别和数量如下表所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **种类** | **1.逃花** | **2.塞网** | **3.破洞** | **4.缝头** | **5.水渍** | **6.脏污** |
| 数量 | 143 | 254 | 10 | 319 | 9 | 47 |
| **种类** | **7.白条** | **8.未对齐** | **9.伪色** | **10.前后色差** | **11.模板取错** | **12.漏浆** |
| 数量 | 8 | 562 | 7 | 25 | 197 | 7 |
| **种类** | **13.脱浆** | **14.色纱（异纤）** | **15.飞絮（线头）** |  |  |  |
| 数量 | 16 | 7 | 23 |  |  |  |

标记文件：示例



说明：

* 每个文件夹下包含各批次的图像，每一批次的印花图案相同，不同批次的印花图案可能相同
* 标记文件、模板图和瑕疵图的文件名称一一对应

# 3 数据预处理

数据预处理的目标是去除数据中的冗余信息，突出数据中的重要信息，即瑕疵。在印花疵点数据集中，已经注明了瑕疵所在的位置，并且可以作为已知信息用于分类任务，因此本文首先对图片进行裁剪，去除瑕疵位置以外的像素点，从而使分类模型专注于瑕疵所在的位置。

由于Bbox的尺寸各不相同，因此在裁剪之后需要对图片进行resize使得数据的尺寸一致，本文分别尝试了缩放和padding两种resize方式，其中缩放可以减少数据为0的像素点，使得瑕疵填满整张图片，但是会因此失去瑕疵区域的尺寸信息，而padding保留了尺寸信息。根据实验结果发现用padding的进行resize后模型的准确率较高，

由于数据集中某些类别的数量过少，因此我们通过旋转、翻转、改变亮度、增加噪点等方式生成新的图片，进行数据增强，扩充了数据集。实验结果表明数据增强可以有效提高准确率。

在印花疵点数据集中，对每个数据提供了瑕疵图和原图一对图片，为了去除背景冗余信息，本文使用了图片差分的方式，通过将瑕疵图和原图进行差分，去除瑕疵图中属于背景部分的像素，只保留瑕疵信息。

最后，本文也尝试了在差分前对瑕疵图和原图进行池化从而提取特征，本文分别对最大池化和平均池化后的数据集进行训练，通过实验得到了效果最好的预处理方式。

## 3.1 裁剪

在裁剪操作中，根据标注中的Bbox信息对图片对进行裁剪，去除瑕疵区域以外的像素，如图3-1所示：

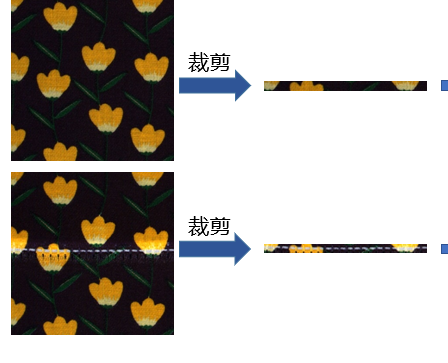


图3-1 裁剪

## 3.2 数据增强

在对图片进行裁剪之后，我们通过旋转随机角度，翻转，改变亮度等操作进行数据扩充。将每一个训练集中类别的图片扩充为600张，从而平衡样本数量。

## 3.3 池化

由于数据集中的图片对没有完全对其，我们尝试了对先对图片进行池化操作，再对池化之后的特征进行差分。

## 3.4 差分

对图片对进行差分后，去除了部分背景信息，只保留存在瑕疵的像素点，实验结果表明差分对分类准确率有明显提升。如图3-2所示：

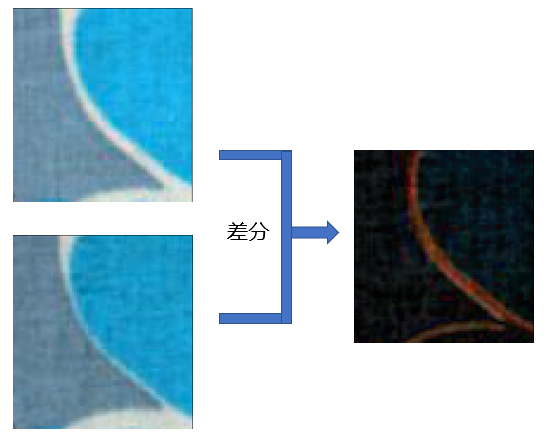


图3-2 差分

## 3.5 resize

将差分后的图片resize成统一大小有两种方法：缩放和padding，缩放通过插值将原图变形成为新的大小，该方法可以使得有效像素填充整个输入向量，但同时会失去图片原来的尺寸信息，如图3-3所示：

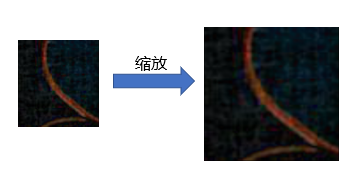


图3-3 缩放

而padding通过在周围填充黑色像素点从而适应形状，该方法保留了图片原来的尺寸信息，并且纹理不会发生变形。如图3-4所示：

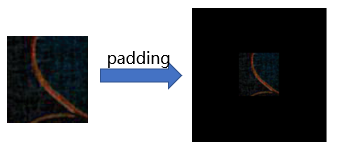


图3-4 padding

# 4 模型介绍及实验

## 4.1 决策树

决策树的建立离不开特征选择，选择一个合适的特征作为判断节点，可以快速的分类，减少决策树的深度。决策树的目标就是把数据集按对应的类标签进行分类。最理想的情况是，通过特征的选择能把不同类别的数据集贴上对应类标签。特征选择的目标使得分类后的数据集比较纯。如何衡量一个数据集纯度，这里就需要引入数据纯度函数。下面将介绍两种表示数据纯度的函数。

## 4.1.1 信息增益

信息熵表示的是不确定度。均匀分布时，不确定度最大，此时熵就最大。当选择某个特征对数据集进行分类时，分类后的数据集信息熵会比分类前的小，其差值表示为信息增益。信息增益可以衡量某个特征对分类结果的影响大小。

假设在样本数据集 D 中，混有 c 种类别的数据。构建决策树时，根据给定的样本数据集选择某个特征值作为树的节点。在数据集中，可以计算出该数据中的信息熵：

图 2. 作用前的信息熵计算公式

其中 D 表示训练数据集，c 表示数据类别数，Pi 表示类别 i 样本数量占所有样本的比例。

对应数据集 D，选择特征 A 作为决策树判断节点时，在特征 A 作用后的信息熵的为 Info(D)，计算如下：

图 3. 作用后的信息熵计算公式

其中 k 表示样本 D 被分为 k 个部分。

信息增益表示数据集 D 在特征 A 的作用后，其信息熵减少的值。公式如下：

图 4. 信息熵差值计算公式

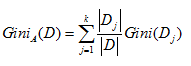
对于决策树节点最合适的特征选择，就是 Gain(A) 值最大的特征。

## 4.1.2 基尼指数

基尼指数是另一种数据的不纯度的度量方法，其公式为：

图 5. 基尼指数计算公式

其中 c 表示数据集中类别的数量，Pi 表示类别 i 样本数量占所有样本的比例。从该公式可以看出，当数据集中数据混合的程度越高，基尼指数也就越高。当数据集 D 只有一种数据类型，那么基尼指数的值为最低 0。如果选取的属性为 A，那么分裂后的数据集 D 的基尼指数的计算公式为：



其中 k 表示样本 D 被分为 k 个部分，数据集 D 分裂成为 k 个 Dj 数据集。

对于特征选取，需要选择最小的分裂后的基尼指数。也可以用基尼指数增益值作为决策树选择特征的依据。公式如下：

图 7. 基尼指数差值计算公式

在决策树选择特征时，应选择基尼指数增益值最大的特征，作为该节点分裂条件。

## 4.1.3 决策树剪枝

在分类模型建立的过程中，很容易出现过拟合的现象。过拟合是指在模型学习训练中，训练样本达到非常高的逼近精度，但对检验样本的逼近误差随着训练次数而呈现出先下降后上升的现象。过拟合时训练误差很小，但是检验误差很大，不利于实际应用。

决策树的过拟合现象可以通过剪枝进行一定的修复。剪枝分为预先剪枝和后剪枝两种。

预先剪枝指在决策树生长过程中，使用一定条件加以限制，使得产生完全拟合的决策树之前就停止生长。预先剪枝的判断方法也有很多，比如信息增益小于一定阀值的时候通过剪枝使决策树停止生长。但如何确定一个合适的阀值也需要一定的依据，阀值太高导致模型拟合不足，阀值太低又导致模型过拟合。

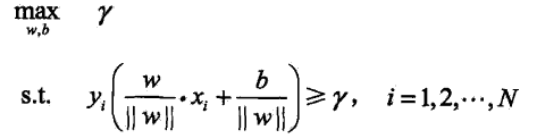
后剪枝是在决策树生长完成之后，按照自底向上的方式修剪决策树。后剪枝有两种方式，一种用新的叶子节点替换子树，该节点的预测类由子树数据集中的多数类决定。另一种用子树中最常使用的分支代替子树。

预先剪枝可能过早的终止决策树的生长，后剪枝一般能够产生更好的效果。但后剪枝在子树被剪掉后，决策树生长的一部分计算就被浪费了。

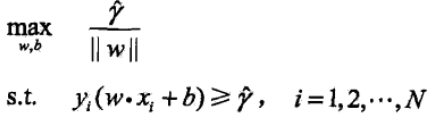
# 4.2 SVM

在感知机中，我们的目标是将训练集分开，只要是能将样本分开的超平面都满足我们的要求，而这样的超平面有很多。支持向量机本质上和感知机类似，然而要求却更加苛刻，我们知道在分类过程中，那些远离超平面的点是安全的，而那些容易被误分类的点是离超平面很近的点，而支持向量机的思想就是要重点关注这些离超平面很近的点，一句话就是在分类正确的同时，让离超平面最近的点到超平面的间隔最大。

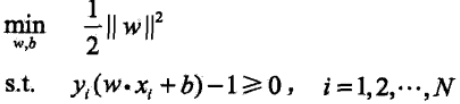
基于上面的感知机我们可以将我们的目标表示为：



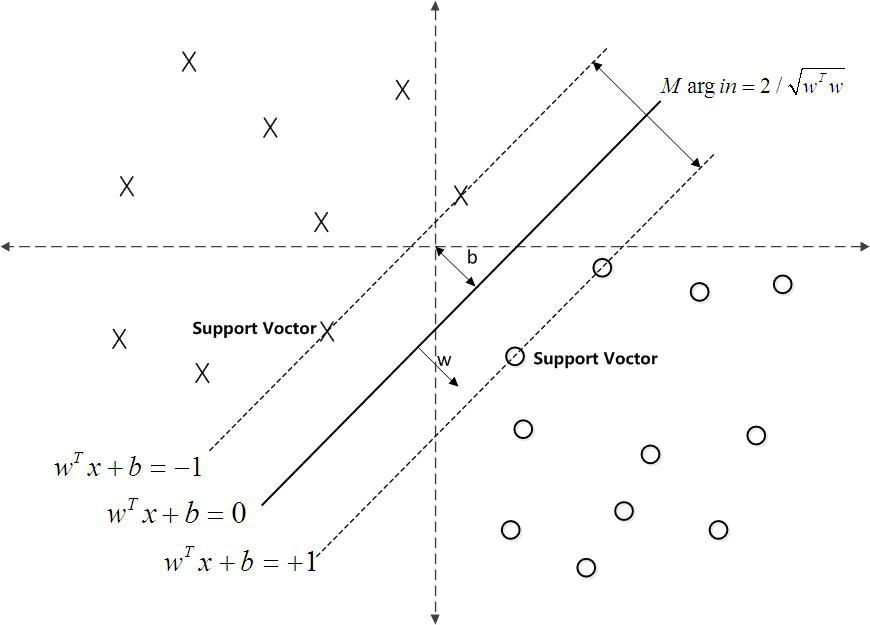
γ是离超平面最近的点的到超平面的几何间隔，将几何间隔用函数间隔替代，可以将式子表示为：



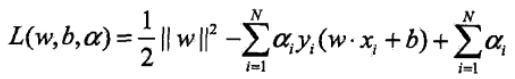
γ表示的是函数间隔，而函数间隔的取值是会随着w，b 成倍数变化而变化的，并不会影响最终的结果，因此令γ(帽子) = 1，则我们最终的问题可以表述为：



　　在这里引出了支持向量机的**第一个亮点：最大化间隔，最大化间隔能使得分类更加精确，且该最大间隔超平面是存在且唯一的。**

****

　　上面的问题中的 1/2||w||2是凸函数，同时约束不等式是仿射函数，因此这是一个凸二次规划问题，根据凸优化理论，我们可以借助拉格朗日函数将我们的约束问题转化为无约束的问题来求解，我们的优化函数可以表达为：

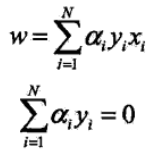


　　αi是拉格朗日乘子，αi ≥ 0 i = 1, 2, 3, ....., n

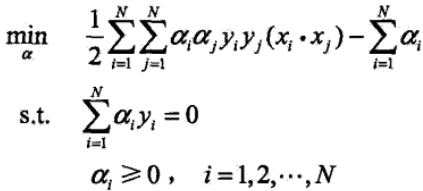
　　根据拉格朗日的对偶性，可以将原始问题转化为对偶问题（只要对偶问题存在，对偶问题的最优化解就是原问题的最优化解，一般对偶问题都比原始问题更容易求解）极大极小问题：



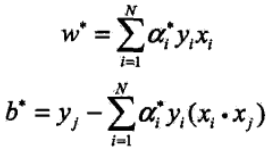
　　先对w，b求导求极小问题， 可以得到w，b的值：



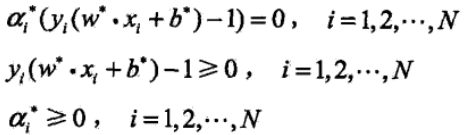
　　将求得的解代入到拉格朗日函数中，可以得到下面的优化函数（将代入后原本的求α 的极大问题转换成了极小问题）：



　　因此只需要求得我们的α 的值就可以求得我们的w，b 的值（求α 的常用算法是SMO算法，假设最终求得的α 的值为α\*，则w，b可以表述成：



　　引入KTT条件（KTT条件是上面拉格朗日函数求最优解的必要条件）：



　　从KTT条件中可以看出，当yi(w\*xi + b\*) - 1 > 0 时，αi\* = 0；当 αi\* > 0 时，yi(w\*xi + b\*) - 1 = 0；

　　结合上面的w，b 表达式可以引出支持向量机的**第二个亮点：w，b 参数只与满足 yi(w\*xi + b\*) - 1 = 0 的样本有关，而这些样本点就是离最大间隔超平面最近的点，我们将这些点称之为支持向量。**因此很多时候支持向量在小样本集分类时也能表现的很好，也正是因为这个原因。

# 4.3 softmax回归

**环境**

Tensorflow1.8、python3.6、ModelArts

**数据集**

64\*64\*3

**算法介绍**

当考虑多分类的问题时，，可以使用​个参数，表示每个类别结果的可能性。显然，。经过一系列推导可以得到下面的softmax函数，，其中，。进一步推导，可得



假设函数的输出为



**算法实现**

softmax\_regression.py

**训练结果**

对不同的数据集，上述代码的参数有所不同，最后的结果如下



# 4.4 简单CNN

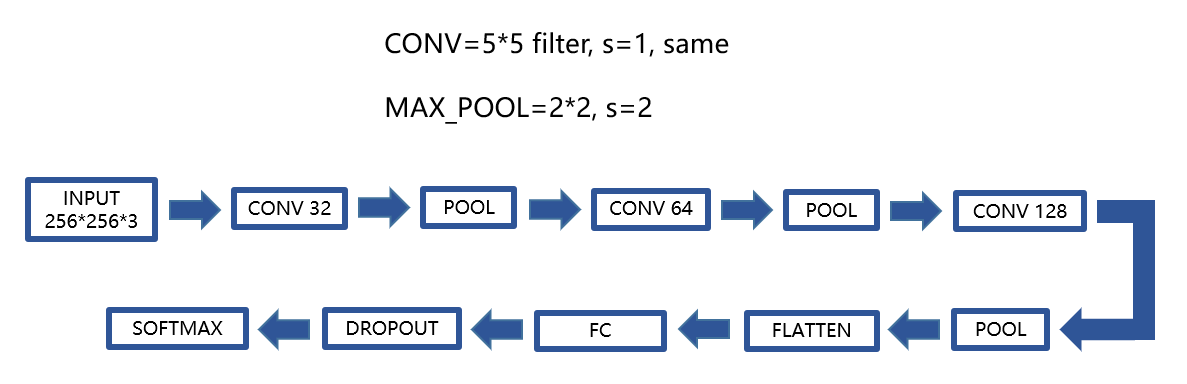
**环境**

Tensorflow1.8、python3.6、ModelArts

**数据集**

256\*256\*3

**算法介绍**



**算法实现**

simple\_cnn.py

**训练结果**

对不同的数据集，上述代码的参数有所不同，最后的结果如下



对task1的4\_diff\_padding\_augment，loss的变化如下



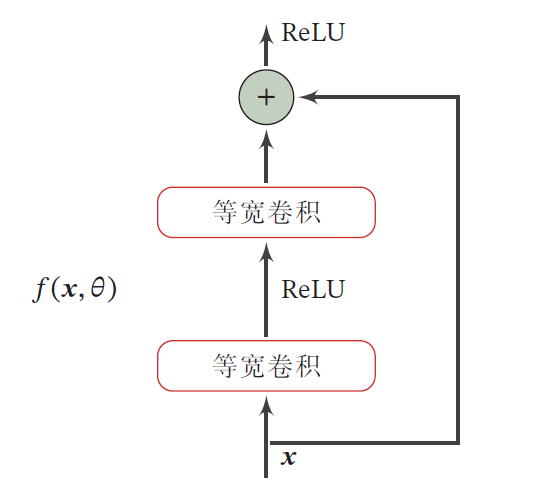
# 4.5 ResNet-18

残差网络(Residual Network,ResNet)通过给非线性的卷积层增加直连边（Shortcut Connection)的方式来提高信息的传播效率。

假设在一个深度网络中，期望一个非线性单元（一层或者多层的卷积层）去逼近一个目标函数。如果将目标函数拆分成两部分：恒等函数（Identity Function)和残差函数（Residue Function)

根据通用近似定理，一个由神经网络构成的非线性单元有足够的能力来近似逼近原始目标函数或残差函数，但实际中后者更容易学习。因此，原来的优化问题可以转换为：让非线性单元去近似残差函数

下图给出了一个典型的残差单元示例，残差单元由多个级联（等宽）卷积层和一个跨层的直连边组成，再经过ReLU激活后得到输出。



残差网络就是将很多个残差单元串联起来构成的一个非常深的网络。

# 本次实验中采用的图片尺寸为64\*64\*3，考虑到图片尺寸较小，实验中采用的残差网络模型为Resnet18模型，其网络结构如下图所示：实验中保持残差块的结构以及大小不变，对残差层之间的卷积层以及池化层结构进行调整，使其更适应数据尺寸。

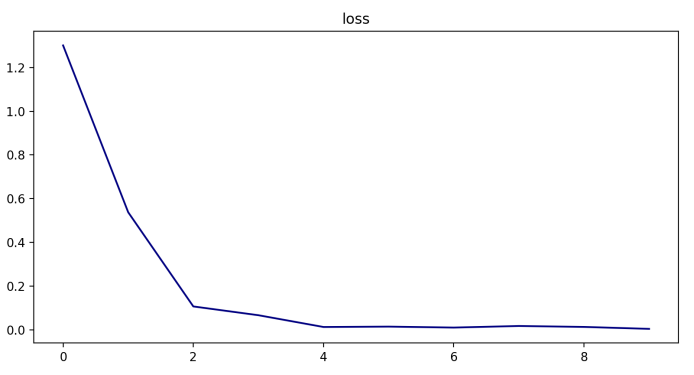
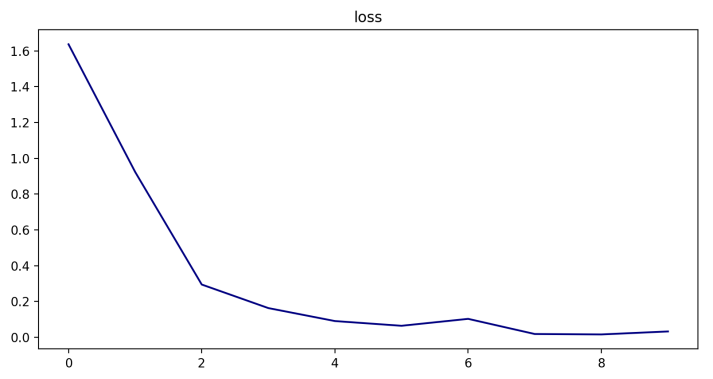
为了减少计算量并保证图像信息的完整性，对第一层卷积层conv1做调整：卷积核大小缩小为3\*3，同时扩大步长为3；同时删去卷积层之后的最大池化层。

# 

**优化器选择：**实验过程中，分别选择了随机梯度下降（SGD）以及Adam优化器对网络参数进行优化，比较实验结果后发现，SGD算法迭代多次后在损失上依然波动，Adam优化器能够使曲线快速收敛，因此实验采用Adam优化器。

**超参数选择：**实验中超参数设置为，epoch\_times=10,lr(优化器学习率)=1e-3。实验以准确率作为评判分类准确率的主要标准，分别比较了6种数据处理方式下的损失函数曲线以及准确率指标。

**损失曲线分析：**实验在task1（3分类）以及task2（5分类）任务下的损失函数曲线如下图所示，可以观察到在3分类（左图），损失曲线在4次epoch时已经收敛；而在5分类（右图），损失函数在7次epoch左右时才完成收敛。



## 4.6 ResNet-50 on ModelArts

使用ModelArts平台提供的ResNet-50模型进行训练。

## 4.6.1 数据上传

ModelArts的数据导入的文件存放方式如图4-1所示：

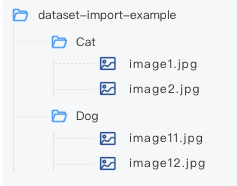


图4-1 ModelArts数据导入文件存放方式示意图

根据指定文件存放方式上船数据，如图4-2所示：



图4-2 上传图片数据到OBS桶

## 4.6.2 创建并发布数据集

在控制台->数据管理->数据集中新建数据集，并从OBS桶导入数据，如图4-3所示：



图4-3 导入数据

最后发布数据集，如图4-4所示：



图4-4 发布数据集

## 4.6.3 创建训练作业

在控制台->训练管理->训练作业中新建训练作业，如图4-5所示：



图4-5 创建训练作业

随后提交作业开始训练。

# 5 实验结果及分析

## 5.x 结果对比

三分类：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 预处理方式 | | | | | | 分类模型 | | | | | |
| 差分 | 最大池化 | 平均池化 | 缩放 | 填充 | 数据增强 | 决策树 | 逻辑回归 | SVM | CNN | ResNet-18 | ResNet-50 |
|  |  |  |  | √ |  |  | 80.1% | 82.2% | 89.6% | 89.0% | 82.1% |
| √ |  |  | √ |  |  |  | 76.4% | 85.8% | 88.5% | 89.5% | 93.2% |
| √ |  |  |  | √ |  |  | 79.1% | 80.1% | 85.3% | 86.4% | 82.1% |
| √ |  |  |  | √ | √ |  | 67.5% | 67.5% | 89.5% | 73.3% | 94.6% |
| √ | √ |  |  | √ | √ |  | 63.4% | 48.1% | 77.5% | 60.7% | 63.5% |
| √ |  | √ |  | √ | √ |  | 63.9% | 50.3% | 72.3% | 62.8% | 59.6% |

五分类：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 预处理方式 | | | | | | 分类模型 | | | | | |
| 差分 | 最大池化 | 平均池化 | 缩放 | 填充 | 数据增强 | 决策树 | 逻辑回归 | SVM | CNN | ResNet-18 | ResNet-50 |
|  |  |  |  | √ |  |  | 68.7% | 70.4% | 78.2% | 81.3% | 72.6% |
| √ |  |  | √ |  |  |  | 65.0% | 74.9% | 77.2% | 80.6% | 78.3% |
| √ |  |  |  | √ |  |  | 67.7% | 70.7% | 77.6% | 79.3% | 79.5% |
| √ |  |  |  | √ | √ |  | 57.8% | 60.5% | 81.6% | 68.4% | 83.5% |
| √ | √ |  |  | √ | √ |  | 54.8% | 50.0% | 62.6% | 57.1% | 66.4% |
| √ |  | √ |  | √ | √ |  | 54.8% | 45.9% | 67.0% | 56.5% | 65.1% |

**Resnet18实验结果分析：**

Resnet-18在task1上的最高分类准确率为89.53%，在task2上的最高分类准确率为81.29%。通过对两表数据观察和比较可以发现，采用填充或者差分+缩放的数据预处理方式，随后使用Resnet18进行分类任务的分类准确率最高，采用差分+缩放以后的准确率次之。而在以上数据预处理方式的基础上加入数据增强步骤，反而会导致准确率的下滑。另一方面，观察损失函数以及测试集上变化趋势以后发现，在5次左右epoch时准确率达到最高值，之后准确率可能下滑，考虑到有可能存在训练集上过拟合问题。

优化方向：使用预训练的Resnet模型进行分类；加入Dropout、Batch Normalization等防止过拟合策略。