# 《人工智能基础》

# 实验报告

项目名称：

项目地址链接：（项目需要设置为公开）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 成员 | 组长 | 组员1 | 组员2 | 组员3 | 组员4 |
| 学号 |  |  |  |  |  |
| 姓名 |  |  |  |  |  |

1. 实验数据集
   1. 数据预处理
2. 说明对数据进行了哪些预处理工作，比如：裁剪、改变通道、改变格式、人脸检测等。说明处理的目的以及处理的结果（没有可以不写）。如果自己加入了新的数据，需要介绍新加入的数据（给出数量、尺寸、特点、如何采集的），以及加入新数据的目的。  
   VGG16网络的输入为224\*224的RGB图像，所以将所有图像格式化为RGB格式，将原图随机裁切为224\*224大小，为了增加训练集样本多样性，训练集中的数据将有50%的概率水平翻转。最后将图像归一化，能够提供更快的收敛和更稳固的训练。
3. 介绍最后所用数据集的规模、训练集中的样本数、测试集中的样本数、输入图像的尺寸。  
   训练集数目: 7096, 验证集数目: 639, 测试集数目: 601，输入图像尺寸224\*224  
   1. 数据加载

描述如何将训练数据和测试数据加载进入PaddlePaddle框架中的。说明加载的具体细节，比如：使用了PaddlePaddle里面哪种数据加载器，参数是如何设置的，这样设置的理由是什么。

继承了paddle.io.Dataset创建自定义数据集，将创建的数据集进行实例化，接着使用了paddle.io.DataLoader将数据加载进paddlepaddle框架中的。训练集、验证集和测试集分别加载对应的数据集实例。

batch\_size开始时设置为128，主要对应着V100的16GB显存，尽量将显存利用率最大化以及最大化矩阵乘法并行化效率，较大的batch\_size收敛更快，需要训练的次数少，准确率上升稳定，但训练后期由于batch\_size较大导致其下降方向已经基本不再变化，跑完一次epoch所需的迭代次数变小，对参数的修正也会显得更加缓慢。

Shuffle设置为True，Shuffle可以防止训练过程中的模型抖动，有利于模型的健壮性，Shuffle可以防止过拟合，并且使得模型学到更加正确的特征。

Dropout设置为True，在向前传播的时候，让某个神经元的激活值以一定概率停止工作，Dropout能够有效缓解模型的过拟合问题，从而使得训练更深更宽的网络成为可能。

1. 网络结构
2. **详细介绍所采用的网络结构，给出网络结构图，说明：**

1）每个卷积层所采用的滤波器个数、卷积尺寸、滤波器深度、激活函数

2）池化层的个数、位置、池化窗口的尺寸及深度

3）全连接层的个数、每个全连接层的滤波器个数、尺寸、深度

4）输出层的神经元个数、激活函数  
**VGG16**

图示

描述已自动生成

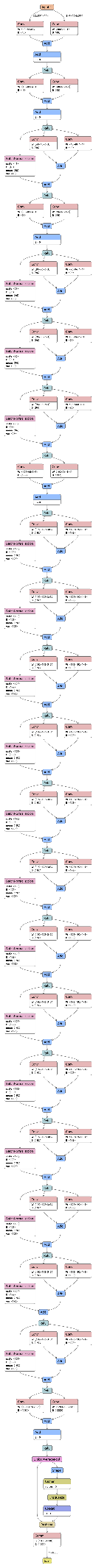
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 卷积层13个 | | | | | | |
| 位置 | 1 | 2 | 4 | 5 | 7 | 8 |
| 输入数据尺寸 | 224\*224\*3 | 224\*224\*64 | 112\*112\*64 | 112\*112\*128 | 56\*56\*128 | 56\*56\*256 |
| 滤波器个数 | 64 | 64 | 128 | 128 | 256 | 256 |
| 卷积尺寸 | 3\*3 | 3\*3 | 3\*3 | 3\*3 | 3\*3 | 3\*3 |
| 滤波器深度 | 3 | 64 | 64 | 128 | 128 | 256 |
| 步长 | 1 | 1 | 1 | 1 | 1 | 1 |
| Padding填充 | same | same | same | same | same | same |
| 激活函数 | Relu | Relu | Relu | Relu | Relu | Relu |
|  |  |  |  |  |  |  |
| 9 | 11 | 12 | 13 | 15 | 16 | 17 |
| 56\*56\*256 | 28\*28\*256 | 28\*28\*512 | 28\*28\*512 | 14\*14\*512 | 14\*14\*512 | 14\*14\*512 |
| 256 | 512 | 512 | 512 | 512 | 512 | 512 |
| 3\*3 | 3\*3 | 3\*3 | 3\*3 | 3\*3 | 3\*3 | 3\*3 |
| 256 | 256 | 512 | 512 | 512 | 512 | 512 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| same | same | same | same | same | same | same |
| Relu | Relu | Relu | Relu | Relu | Relu | Relu |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 池化层5个 | | | | | |  |
| 位置 | 3 | 6 | 10 | 14 | 18 |  |
| 输入数据尺寸 | 224\*224\*64 | 112\*112\*128 | 56\*56\*256 | 28\*28\*512 | 14\*14\*512 |  |
| 滤波器尺寸 | 2\*2 | 2\*2 | 2\*2 | 2\*2 | 2\*2 |  |
| 池化方法 | max pooling | max pooling | max pooling | max pooling | max pooling |  |
|  |  |  |  |  |  |  |
| 全连接层3个 | | | | |  |  |
| 位置 | Flatten | 19 | 20 | 21 |  |  |
| 输入数据尺寸 | 7\*7\*512 | 25088 | 1\*1\*4096 | 1\*1\*4096 |  |  |
| 滤波器尺寸 | 将数据拉平成向量 | 1\*1 | 1\*1 | 1\*1 |  |  |
| 滤波器个数 |  | 4096 | 4096 | 12 |  |  |
| 滤波器深度 |  | 1 | 4096 | 4096 |  |  |
|  |  |  |  |  |  |  |
| 输出层 | |  |  |  |  |  |
| 神经元个数 | 12 |  |  |  |  |  |
| 激活函数 | softmax |  |  |  |  |  |

**RepVGG**

有点难去详细介绍，在此统计一下网络的必要信息：

有44个卷积层，卷积尺寸3\*3，滤波器个数各有48、96、192个。激活层22个Relu，1个自适应池化层，22个RepVGG Block，最后有1个全连接层。

下图为详细模型网络结构图



1. **简述选择该网络结构的理由。**VGG16具有巨大的参数数目，具有很高的拟合能力。VGG网络结构简洁，都是由小卷积核、小池化核、ReLU组合而成RepVGG将训练推理网络结构进行独立设计，在训练时使用高精度的多分支网络学习权值，在推理时使用低延迟的单分支网络，然后通过结构重参数化将多分支网络的权值转移到单分支网络。RepVGG性能达到了SOTA，思路简单新颖。

**（3）说明尝试了几种网络模型，分别是什么模型，优缺点是什么。**VGG优点：小卷积核，小池化核，层数更深，卷积核堆叠的感受野，全连接转卷积；缺点：训练时间长，存储容量大。RepVGG优点：简单即快速、内存使用经济、灵活，训练时的多分支结构；缺点：模型较大，结构没有VGG简单。

1. 损失函数及学习率

（1）给出网络所使用的损失函数公式，说明选择该损失函数的理由，说明反向传播是如何作用在权重上的（可以简写或不写）以及其他相关的内容。  
**交叉熵损失**：

图形用户界面, 文本, 应用程序

描述已自动生成

交叉熵能够衡量同一个随机变量中的两个不同概率分布的差异程度，交叉熵作为损失函数在进行梯度下降计算的时候可以避免。出现梯度弥散，导致学习速率下降。

**负对数似然损失**：

图形用户界面, 文本, 应用程序

描述已自动生成

Nll是交叉熵的底层实现，softamax+log+nllLoss=crossEntropyLoss，探究其与交叉熵损失的差距。

在反向传播过程中，更新每个权值，将其输出差值与输入激活相乘，以便得到该权值梯度。从权值中减去梯度的比值（百分比）。该比值称为学习率。

（2）说明尝试了几种损失函数，分别是什么函数。  
两种，交叉熵损失函数与负对数似然损失函数

（3）学习率是多少？简述确定该学习率的过程。  
学习率为0.0001，有方向性地尝试了六组值，最终确定为0.0001

1. 优化器

（1）介绍所采用的优化器的原理、特点，及选择它的原因。  
**Adam优化器**

原理：

文本

描述已自动生成

特点：

Adma吸收了Adagrad（自适应学习率的梯度下降算法）和动量梯度下降算法的优点，既能适应稀疏梯度（即自然语言和计算机视觉问题），又能缓解梯度震荡的问题

原因：

最常用的Adam优化器，有着收敛速度快、调参容易等优点

**Momentum优化器**

原理：

文本

描述已自动生成

特点：

Momentum梯度下降算法在与原有梯度下降算法的基础上，引入了动量的概念，使网络参数更新时的方向会受到前一次梯度方向的影响，换句话说，每次梯度更新都会带有前几次梯度方向的惯性，使梯度的变化更加平滑；Momentum梯度下降算法能够在一定程度上减小权重优化过程中的震荡问题。

原因：

增加了稳定性：不只是依赖于当前时刻的梯度，还考虑了以往时刻的梯度。在梯度方向有所改变的维度上更新速度变慢，可加快收敛并减小震荡；收敛速度更快：惯性加持，在梯度方向不变的维度上速度更快；还有一定摆脱局部最优的能力。

（2）说明尝试了几种优化器，分别是什么优化器。  
两种，Adam优化器与Momentum优化器

1. 实验结果
2. 用表格或图的形式，对比各种尝试（使用不同的网络模型、损失函数、优化器）的实验结果（recall，precision，accuracy，error-rate）；  
   **1.不同学习率的比较**  
   VGG16网络，Adam优化器，交叉熵损失函数，batch\_size=128，epochs=20

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学习率 | 0.000005 | 0.00001 | 0.00005 | 0.0001 | 0.0005 | 0.001 |
| accuracy | 0.35625 | 0.390625 | 0.489063 | 0.553125 | 0.4125 | 0.323437 |
| error rate | 0.64375 | 0.609375 | 0.510938 | 0.446875 | 0.5875 | 0.676562 |
| precision | 0.36034357 | 0.395815 | 0.510386 | 0.580247 | 0.41017 | 0.341005 |
| recall | 0.35612422 | 0.389922 | 0.490332 | 0.552678 | 0.412028 | 0.32535 |

**2.不同网络的比较**Epochs=20,学习率=0.0001

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 优化器 | | Adam | | Momentum | |
| 损失函数 | | 交叉熵 | NLL | 交叉熵 | NLL |
| **VGG16** | accuracy | 0.50625 | 0.49375 | 0.253125 | 0.25625 |
| error rate | 0.49375 | 0.50625 | 0.746875 | 0.74375 |
| precision | 0.532701 | 0.525507 | 0.242026 | 0.271008 |
| recall | 0.505401 | 0.492008 | 0.252789 | 0.256476 |
| **RepVGG** | accuracy | 0.410937 | 0.45625 | 0.157812 | 0.148437 |
| error rate | 0.589062 | 0.54375 | 0.842187 | 0.851562 |
| precision | 0.414161 | 0.465634 | 0.159138 | 0.155777 |
| recall | 0.411218 | 0.458458 | 0.157989 | 0.149538 |

**3.不同损失函数的比较**

Epochs=20,学习率=0.0001

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 优化器 | | Adam | | Momentum | |
| 网络 | | VGG | RepVGG | VGG | RepVGG |
| **交叉熵** | accuracy | 0.50625 | 0.410937 | 0.253125 | 0.157812 |
| error rate | 0.49375 | 0.589062 | 0.746875 | 0.842187 |
| precision | 0.532701 | 0.414161 | 0.242026 | 0.159138 |
| recall | 0.505401 | 0.411218 | 0.252789 | 0.157989 |
| **NLL** | accuracy | 0.49375 | 0.45625 | 0.25625 | 0.148437 |
| error rate | 0.50625 | 0.54375 | 0.74375 | 0.851562 |
| precision | 0.525507 | 0.465634 | 0.271008 | 0.155777 |
| recall | 0.492008 | 0.458458 | 0.256476 | 0.149538 |

**4.不同优化器的比较**

Epochs=20,学习率=0.0001

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 网络 | | VGG16 | | RepVGG | |
| 损失函数 | | 交叉熵 | NLL | 交叉熵 | NLL |
| **Adam** | accuracy | 0.50625 | 0.49375 | 0.410937 | 0.45625 |
| error rate | 0.49375 | 0.50625 | 0.589062 | 0.54375 |
| precision | 0.532701 | 0.525507 | 0.414161 | 0.465634 |
| recall | 0.505401 | 0.492008 | 0.411218 | 0.458458 |
| **Momentum** | accuracy | 0.253125 | 0.25625 | 0.157812 | 0.148437 |
| error rate | 0.746875 | 0.74375 | 0.842187 | 0.851562 |
| precision | 0.242026 | 0.271008 | 0.159138 | 0.155777 |
| recall | 0.252789 | 0.256476 | 0.157989 | 0.149538 |

**5.最终测试准确率**

accuracy: 0.833333

error rate: 0.166666

precision: 0.836218

recall: 0.833892

文本

中度可信度描述已自动生成

1. 分析、解释实验过程中出现的一些现象。

1.对于不同学习率的比较分析：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学习率 | 0.000005 | 0.00001 | 0.00005 | 0.0001 | 0.0005 | 0.001 |
| accuracy | 0.35625 | 0.390625 | 0.489063 | 0.553125 | 0.4125 | 0.323437 |
| error rate | 0.64375 | 0.609375 | 0.510938 | 0.446875 | 0.5875 | 0.676562 |
| precision | 0.36034357 | 0.395815 | 0.510386 | 0.580247 | 0.41017 | 0.341005 |
| recall | 0.35612422 | 0.389922 | 0.490332 | 0.552678 | 0.412028 | 0.32535 |

2.对于不同网络的比较分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 优化器 | | Adam | | Momentum | |
| 损失函数 | | 交叉熵 | NLL | 交叉熵 | NLL |
| **VGG16** | accuracy | 0.50625 | 0.49375 | 0.253125 | 0.25625 |
| error rate | 0.49375 | 0.50625 | 0.746875 | 0.74375 |
| precision | 0.532701 | 0.525507 | 0.242026 | 0.271008 |
| recall | 0.505401 | 0.492008 | 0.252789 | 0.256476 |
| **RepVGG** | accuracy | 0.410937 | 0.45625 | 0.157812 | 0.148437 |
| error rate | 0.589062 | 0.54375 | 0.842187 | 0.851562 |
| precision | 0.414161 | 0.465634 | 0.159138 | 0.155777 |
| recall | 0.411218 | 0.458458 | 0.157989 | 0.149538 |

RepVGG相对VGG16来说在除在Adam优化器与NLL损失函数的情况下差距较小，其他情况下准确率都比VGG16小了10%，可能是VGG16在20个epoch内的提升更加明显，以及RepVGG在相同的batch\_size=128下无法发挥全部的实力，显存空余较多，对参数的修正更为缓慢。  
3.对于不同的损失函数的比较分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 优化器 | | Adam | | Momentum | |
| 网络 | | VGG | RepVGG | VGG | RepVGG |
| **交叉熵** | accuracy | 0.50625 | 0.410937 | 0.253125 | 0.157812 |
| error rate | 0.49375 | 0.589062 | 0.746875 | 0.842187 |
| precision | 0.532701 | 0.414161 | 0.242026 | 0.159138 |
| recall | 0.505401 | 0.411218 | 0.252789 | 0.157989 |
| **NLL** | accuracy | 0.49375 | 0.45625 | 0.25625 | 0.148437 |
| error rate | 0.50625 | 0.54375 | 0.74375 | 0.851562 |
| precision | 0.525507 | 0.465634 | 0.271008 | 0.155777 |
| recall | 0.492008 | 0.458458 | 0.256476 | 0.149538 |

交叉熵和NLL均为多分类问题常用的损失函数，经实验可以发现，在相同网络以及优化器下，交叉熵和NLL的效果没有太大差异，交叉熵总体上的效果优于NLL，说明softmax函数对提高预测正确率存在一定的作用但并不明显。  
4.对于不同优化器的比较分析

5.最终结果的分析：

accuracy: 0.833333

error rate: 0.166666

precision: 0.836218

recall: 0.833892

图片包含 文本

描述已自动生成

最终结果准确率为83.33%，并没有特别

1. 总结与收获（必写）

谈谈学习这门课的心得体会。每个小组成员分别写自己的心得（每人的心得不得少于100字），汇总于此。

学生1姓名：心得。。。。

学生2姓名：心得。。。。

学生3姓名：心得。。。。

学生4姓名：心得。。。。

学生5姓名：心得。。。。