# 《人工智能基础》

# 实验报告

项目名称：_____

项目地址链接：https://aistudio.baidu.com/aistudio/projectdetail/4214486?contributionType=1（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

## 1.1. 数据预处理

（1） 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。

（2） 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

## 1.2. 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。

# 2. 网络结构

**（1） 详细介绍所采用的网络结构，给出网络结构图，说明：**

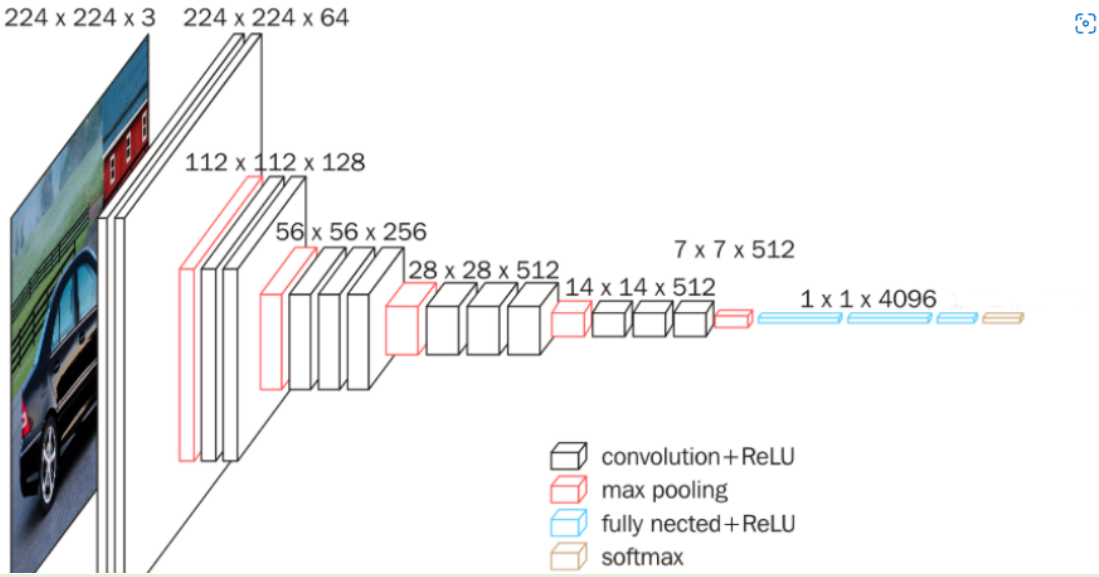
1）每个卷积层所采用的滤波器个数、卷积尺寸、滤波器深度、激活函数

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**




| 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

（2） **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。

# 3. 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：**

如果提供 *weight* 参数的话，它是一个 *1-D* 的tensor，每个值对应每个类别的权重。 该损失函数的数学计算公式如下：

$$loss_j = -\text{input}[\text{class}] + \log\left(\sum_{i=0}^{K}\exp(\text{input}_i)\right), j = 1, \dots, K$$

当 *weight* 不为 *none* 时，损失函数的数学计算公式为：

$$loss_j = \text{weight}[\text{class}](-\text{input}[\text{class}] + \log\left(\sum_{i=0}^{K}\exp(\text{input}_i)\right)), j = 1, \dots, K$$

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：**

当 *reduction* 设置为 *none* 时，损失函数的数学计算公式为：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} x_{n,y_n}, \quad w_c = \text{weight}[c] \cdot 1\{c \neq \text{ignore\_index}\},$$

其中 *N* 表示 *batch_size* 。如果 *reduction* 的值不是 *none* (默认为 *mean*)，那么此时损失函数 的数学计算公式为：

$$\ell(x, y) = \begin{cases} \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} l_n, & \text{if reduction= 'mean';} \\ \sum_{n=1}^{N} l_n, & \text{if reduction= 'sum'.} \end{cases}$$

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

# 4. 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：

**算法 8.7** Adam 算法

**Require:** 步长 $\epsilon$（建议默认为：0.001）

**Require:** 矩估计的指数衰减速率，$\rho_1$ 和 $\rho_2$ 在区间 $[0,1)$ 内。（建议默认为：分别为 0.9 和 0.999）

**Require:** 用于数值稳定的小常数 $\delta$（建议默认为：$10^{-8}$）

**Require:** 初始参数 $\boldsymbol{\theta}$

初始化一阶和二阶矩变量 $\boldsymbol{s}=0, \boldsymbol{r}=0$

初始化时间步 $t=0$

**while** 没有达到停止准则 **do**

从训练集中采包含 $m$ 个样本 $\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)})$

$t \leftarrow t+1$

更新有偏一阶矩估计：$\boldsymbol{s} \leftarrow \rho_1\boldsymbol{s}+(1-\rho_1)\boldsymbol{g}$ Momentum项

更新有偏二阶矩估计：$\boldsymbol{r} \leftarrow \rho_2\boldsymbol{r}+(1-\rho_2)\boldsymbol{g}\odot\boldsymbol{g}$ RMSProp项

修正一阶矩的偏差：$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$

修正二阶矩的偏差：$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$

计算更新：$\Delta\boldsymbol{\theta} = -\epsilon\frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$（逐元素应用操作）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}+\Delta\boldsymbol{\theta}$

**end while**



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：

**算法 8.2** 使用动量的随机梯度下降（SGD）

**Require:** 学习率 $\epsilon$，动量参数 $\alpha$

**Require:** 初始参数 $\theta$，初始速度 $v$

**while** 没有达到停止准则 **do**

从训练集中采包含 $m$ 个样本 $\{x^{(1)}, \ldots, x^{(m)}\}$ 的小批量，对应目标为 $y^{(i)}$。

计算梯度估计：$g \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(x^{(i)}; \theta), y^{(i)})$

计算速度更新：$v \leftarrow \alpha v - \epsilon g$ 指数衰减平均，以alpha为衰减力度，alpha越大，之前梯度对现在方向的影响也越大

应用更新：$\theta \leftarrow \theta + v$

**end while**

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

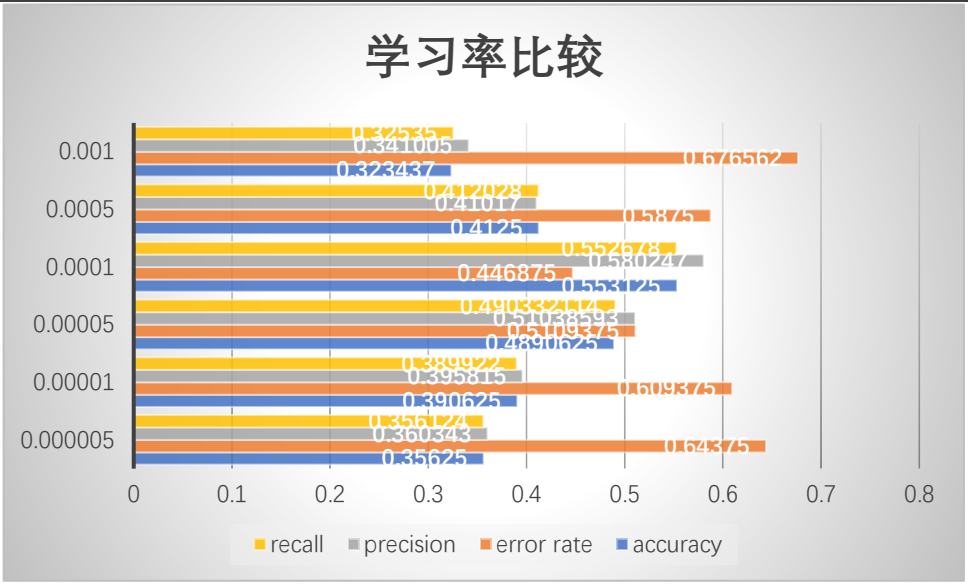
# 5. 实验结果

（1） 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 |

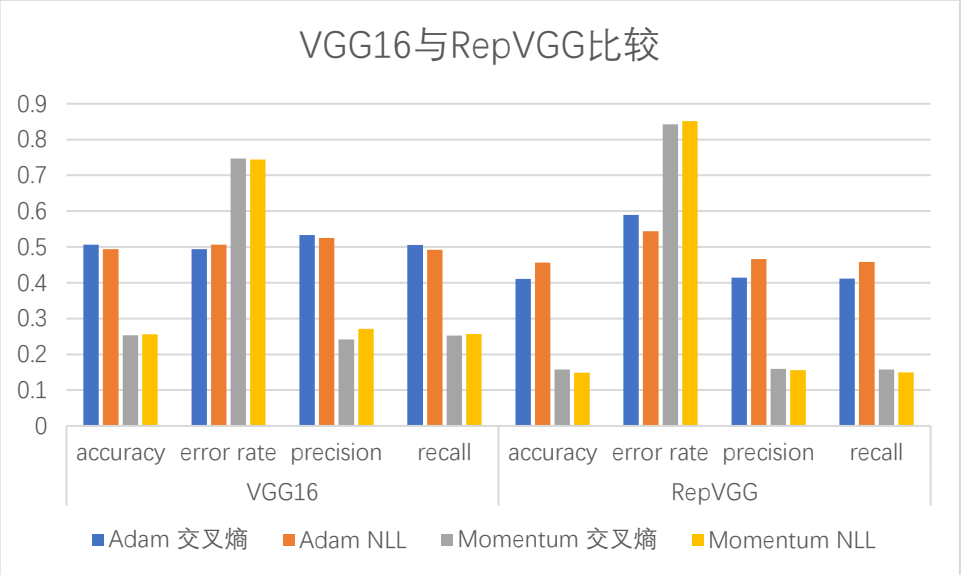



**2.不同网络的比较**

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 |
| | accuracy | 0.410937 | 0.45625 | 0.157812 | 0.148437 |

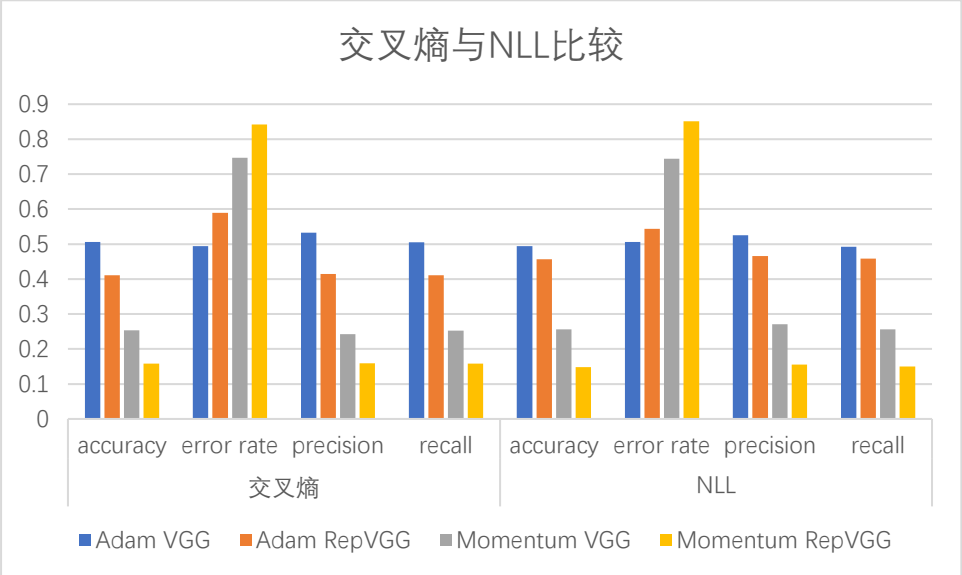
| | | | | | |
|---|---|---|---|---|---|
| RepVGG | error rate | 0.589062 | 0.54375 | 0.842187 | 0.851562 |
| | precision | 0.414161 | 0.465634 | 0.159138 | 0.155777 |
| | recall | 0.411218 | 0.458458 | 0.157989 | 0.149538 |




**3.不同损失函数的比较**

Epochs=20,学习率=0.0001

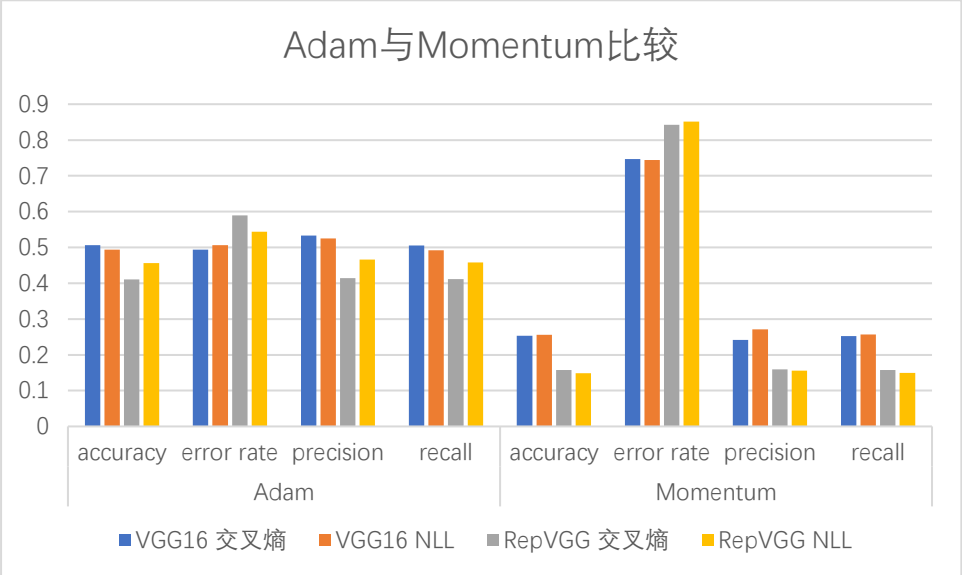
| 优化器 | | Adam | | Momentum | |
|---|---|---|---|---|---|
| 网络 | | VGG | RepVGG | VGG | RepVGG |
| 交叉熵 | accuracy | 0.50625 | 0.410937 | 0.253125 | 0.157812 |
| | error rate | 0.49375 | 0.589062 | 0.746875 | 0.842187 |
| | precision | 0.532701 | 0.414161 | 0.242026 | 0.159138 |
| | recall | 0.505401 | 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 |



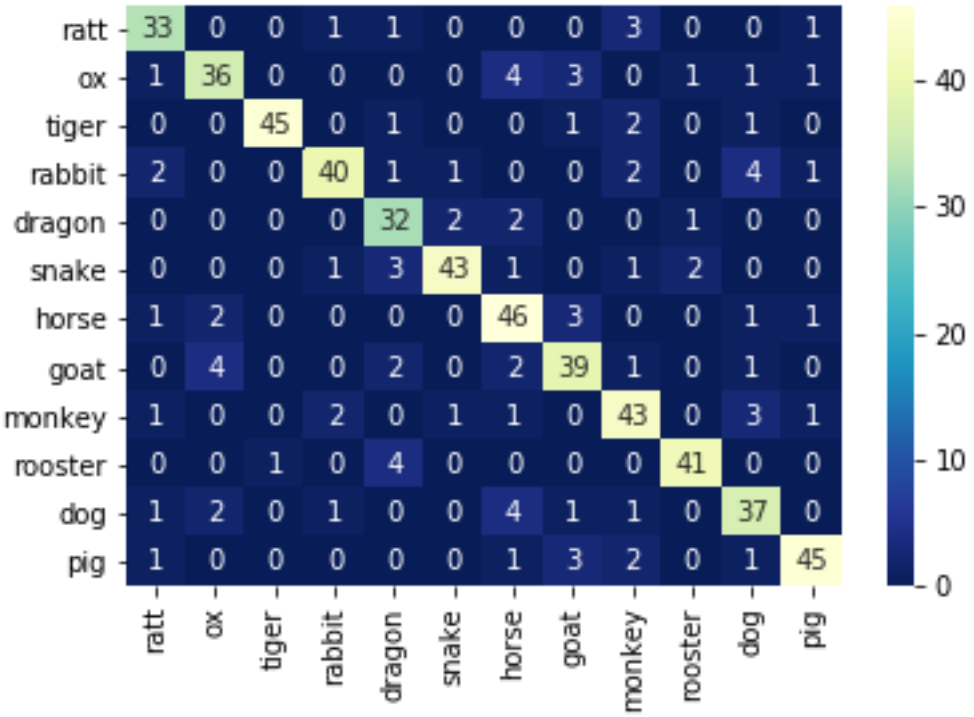

**5.最终测试准确率**

accuracy: 0.833333

error rate: 0.166666

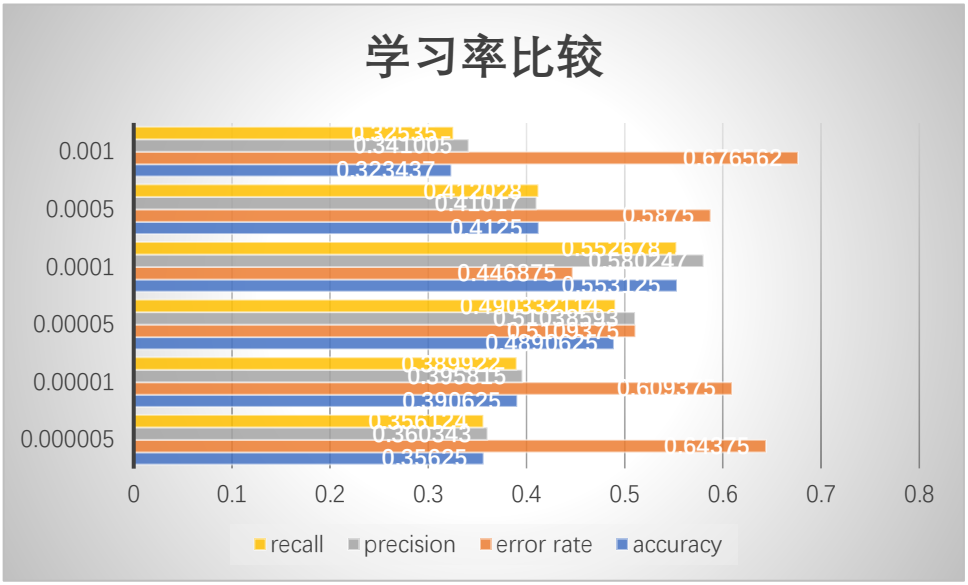
precision: 0.836218

recall: 0.833892




（2） 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：




学习率为 0.0001 时，accuracy、precision、recall 显著高于其他五组数据，表明此时整体的预测准确程度较高、对正样本结果中的预测准确程度较高、实际为正的样本中被预测为正样本的概率较高； error rate 显著低于其他五组数据，表明此时整体的预测错误率较低。总体看来，学习率为 0.0001 时模型的预测能力最强。

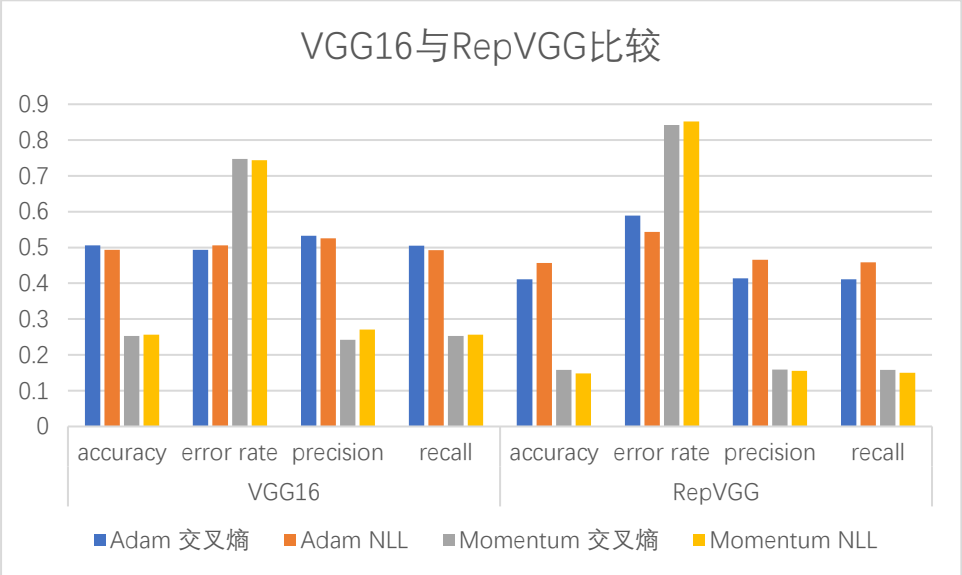
学习率设置过小，收敛速度会非常慢，会增大找到最优值的时间，而且有可能进入局部极值点就收敛，找不到真正的最优解。所以随学习率的增大，预测能力会逐渐增强。

如果学习率太高，网络在训练过程中可能会跳过最小值点。更糟糕的是，高学习率可能会导致 loss 不断变大，这样就脱离了模型的学习目标。所以学习率超过 0.0001 时，错误率反而增大，预测准确率降低。

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 |

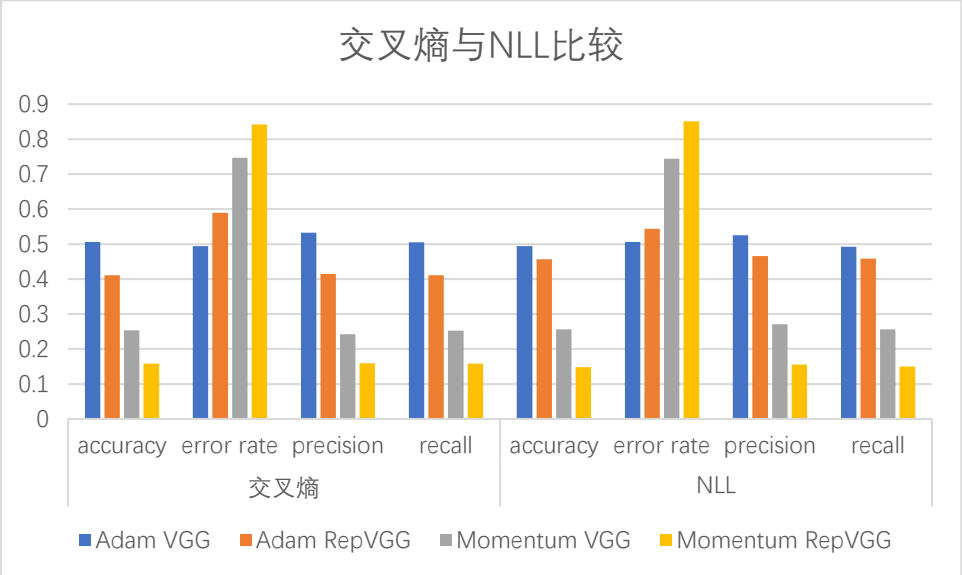
| | | | | | |
|---|---|---|---|---|---|
| | precision | 0.414161 | 0.465634 | 0.159138 | 0.155777 |
| | recall | 0.411218 | 0.458458 | 0.157989 | 0.149538 |




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：

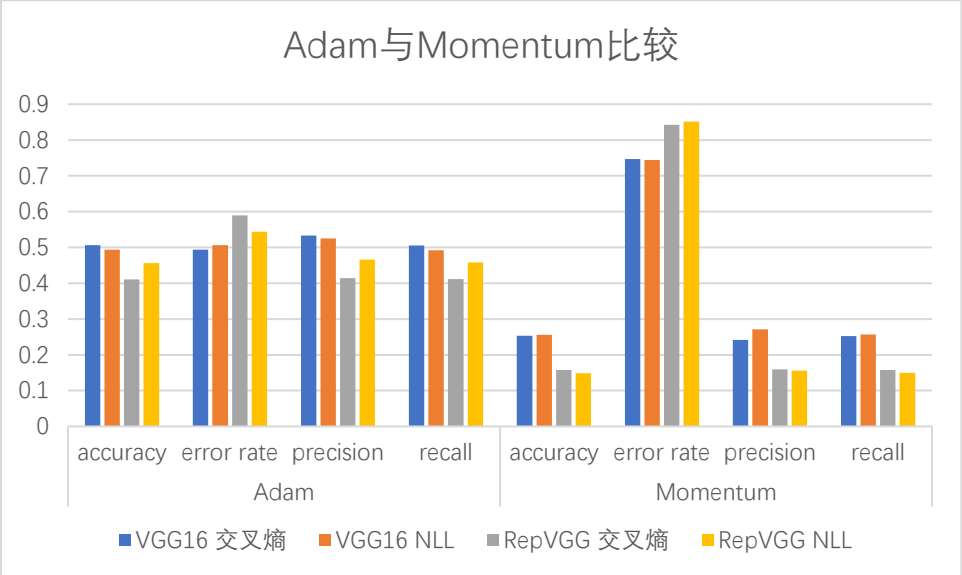
| 优化器 | | Adam | | Momentum | |
|---|---|---|---|---|---|
| 网络 | | VGG | RepVGG | VGG | RepVGG |
| 交叉熵 | accuracy | 0.50625 | 0.410937 | 0.253125 | 0.157812 |
| | error rate | 0.49375 | 0.589062 | 0.746875 | 0.842187 |
| | precision | 0.532701 | 0.414161 | 0.242026 | 0.159138 |
| | recall | 0.505401 | 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

| 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 |




在其他可控因素不变的情况下，

相比于 Momentum，Adam 在 accuracy，precision，recall 三个方面的数值明显更高

即使用 Adam 优化器的情况下，对于各方面的预测准确度和预测正样本概论较高；

同时 error rate 相比于使用 Momentum 的情况，出现了大幅度降低。

从理论上分析，Adam 优化器不但吸收了 Momentum 的动量梯度下降的优点，能够缓解梯度震荡的问题
同时也能通过自适学习率应梯度下降算法，解决稀疏数据的问题。因此从理论上看，Adam 优化器显然好于

Momentum 优化器，而实际实验数据也印证了这一点。因此采用 Adam 优化器

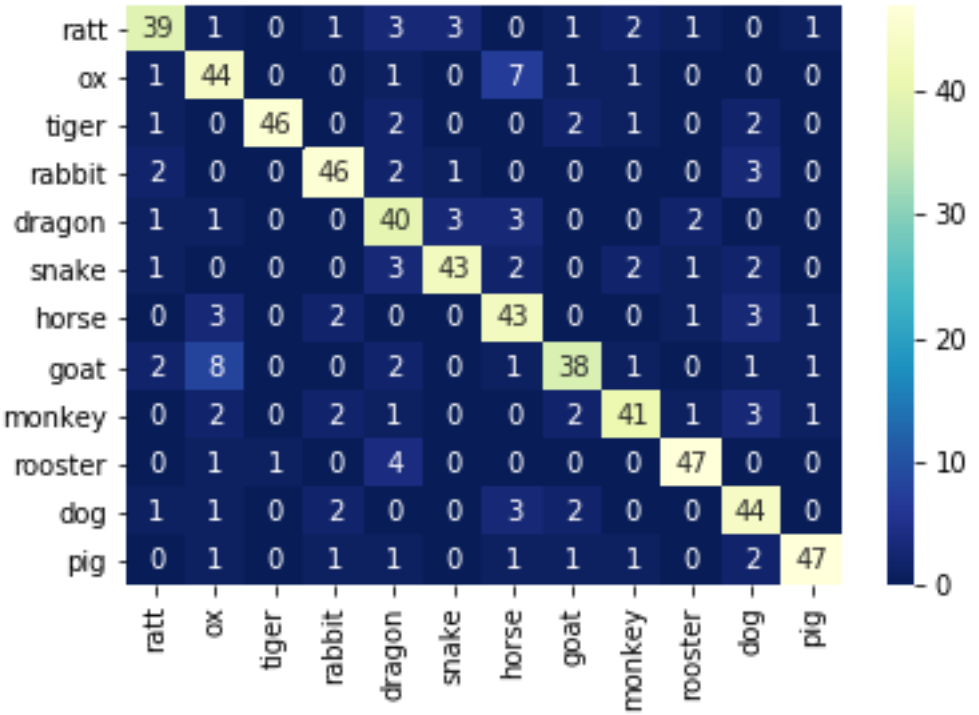
5.最终结果的分析：

accuracy: 0.833333

error rate: 0.166666

precision: 0.836218

recall: 0.833892




最终结果准确率为 83.33%，由于使用 VGG16 网络和训练集较大、输出结果较多导致训练较慢，若需要进一步提升还需要大量时间来进行提升。从混淆矩阵中可以看出一些有意思的东西，老鼠和龙的识别成功率较低，可能训练集中的图像风格不一，同一类别中都有较大的差别。牛也有可能识别为羊，马有可能识别成牛，因为这三类动物相似度较高。识别成功率较高的是猪和鸡，可能因为形态和颜色与其他种类的动物具有较大的差别

# 6. 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。。。。

最后将项目做出来确实没有那么容易，尽管最后的准确率只达到了 83%，但也是尽了最大的努力。这次项目总体思路不难想，而是对所有的细节都要有深层的了解，在课上只是简单地了解了每个超参的含义，但真正实际操作起来需要慢慢去琢磨与分析。像 batch_size 和 learning rate 这种都要根据具体的网络具体的优化器去选择合适的数值更是有所挑战，本来想着尝试 RepVGG 网络去达到一个更好的效果，结果对这个网络还不够熟悉所以 batch_size 和 learning rate 并没有很好地去考量，而是简单地去与 VGG16 网络设置的一致，导致最终的效果并不是尽如人意。人工智能这个学科的知识面很广，而且每时每刻都有可能会有更新的知识加入进来，要想跟上人工智能学习的步伐，必须牢牢地打好基础并一直持续不断地给自己灌输新的知识。

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。。。。