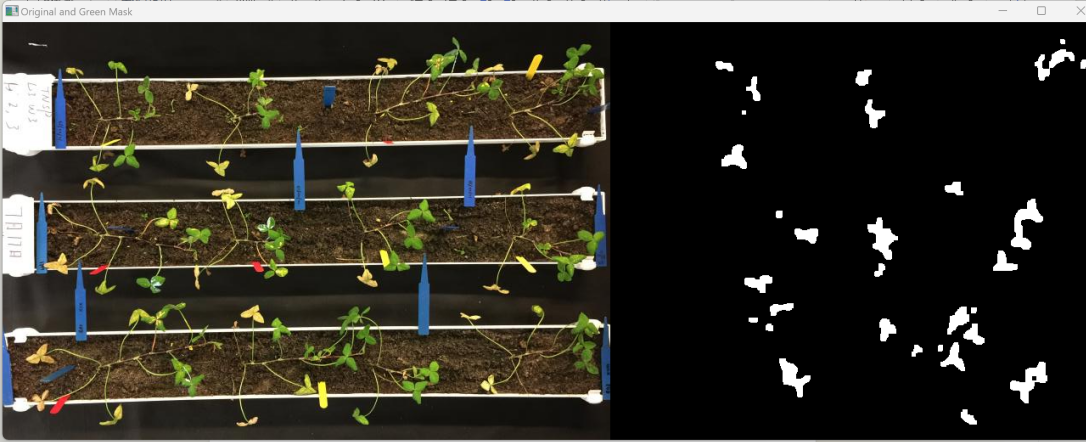
这里用cpp来完成作业1-6，这里用的是Clion作为ide，编译环境为MinGW，用CMake来作为编译脚本，Opencv的版本为4.10.0，cpp版本为C++11

一、提取图中的植物部分，并估算植物的绿色部分面积，已知植物生长的槽的宽是20 cm。



读取图像，转换图像至 HSV 色彩空间，便于分离绿色区域，定义绿色的 HSV 范围，通过 cv::inRange 生成绿色掩码,使用形态学操作（开闭运算）去除小噪声和孔洞，得到绿色区域掩码，通过 cv::countNonZero 统计绿色像素数，依据槽宽 20 cm 和图像宽度计算每像素对应的实际面积，估算绿色部分总面积



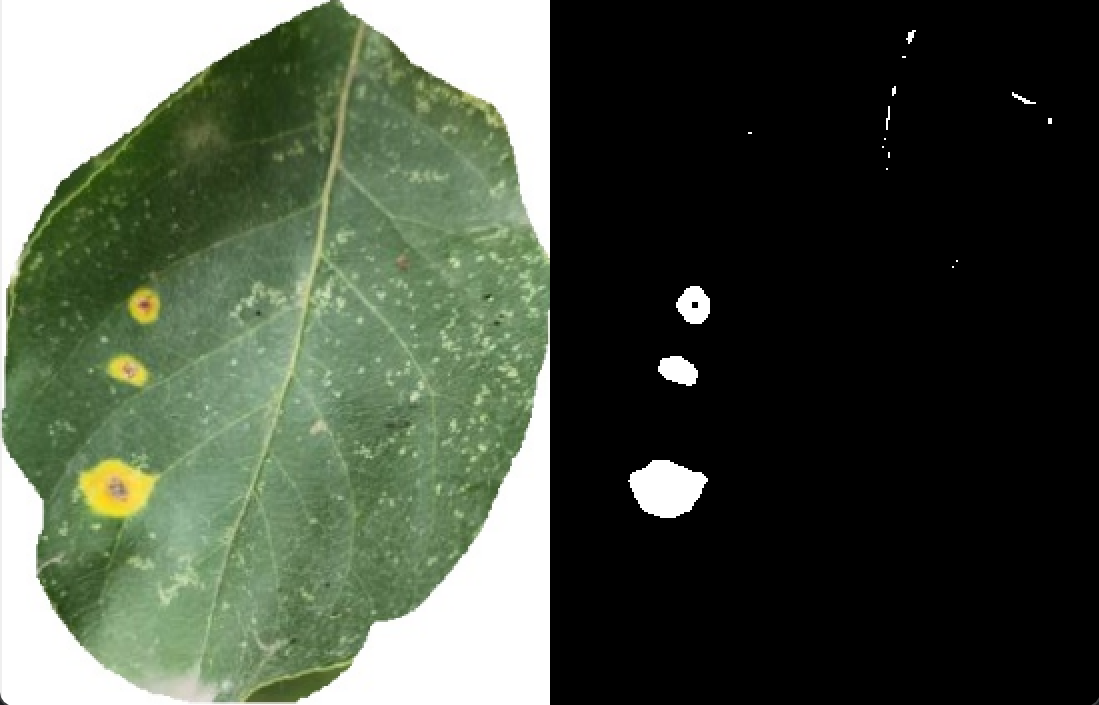
绿色面积为10.6847平方厘米

具体代码详见work1.cpp

二、提取图像中的叶片病害的图斑数目、估算病害图斑占叶片的总面比例。



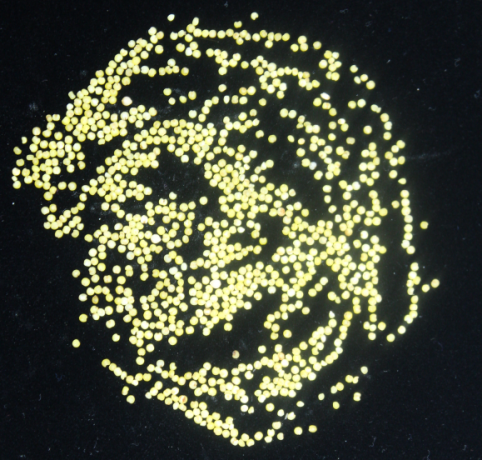
读取叶片图像转换为 HSV 颜色空间，定义黄色区域的 HSV 范围，用 cv::inRange 生成病害掩码图，使用形态学操作（闭运算）去除噪声和孔洞，清理掩码区域，转换原图为灰度图，统计非零像素数，作为叶片总面积，统计掩码中非零像素数，作为病害区域面积计算病害面积占比：(病害面积 / 叶片总面积) × 100%



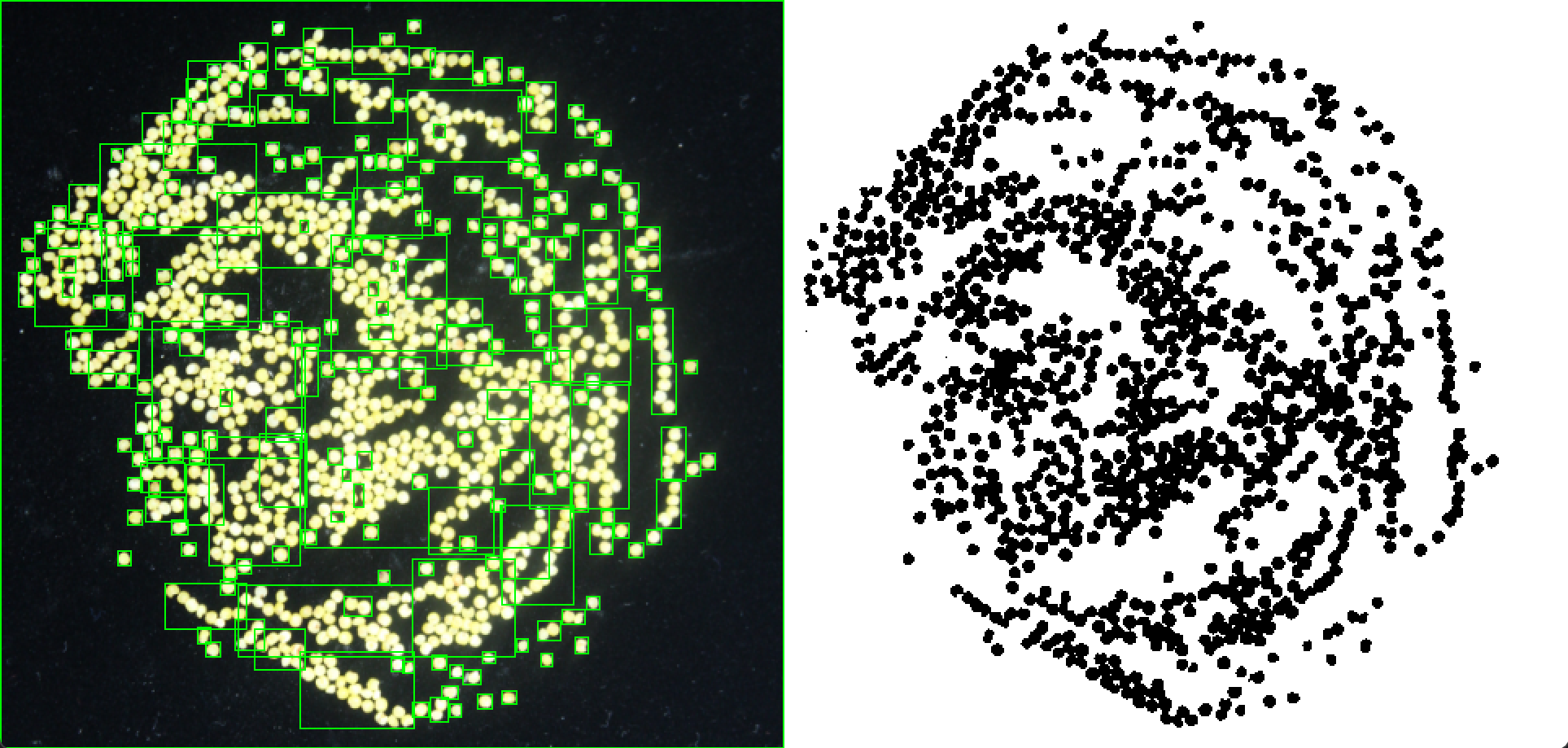
比例为1.36698%

具体代码详见work2.cpp

三、采用图像形态学获取图像中小米的粒数、每粒小米的最大投影面积或者外接最大正方形的长与宽。



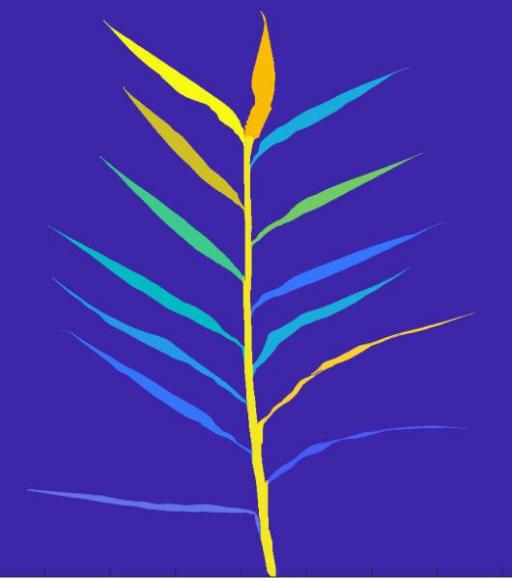
读取图像并转换为灰度图，通过固定阈值或自适应阈值，将图像转化为二值图，突出小米颗粒并抑制背景，使用 cv::findContours 检测小米颗粒的轮廓，遍历轮廓，计算每颗颗粒的面积，过滤小面积噪声，在原图上绘制外接框，以便直观验证结果。



数量为309

具体代码详见work3.cpp

四、分割获取谷子的各叶片，即把每个叶片独立分割，并用不同颜色表示。



读取输入图像并转化为灰度图像，通过高斯模糊平滑图像，减少噪声对边缘检测的干扰，使用 Canny 算法检测图像中的边缘，使用 cv::findContours 获取每片叶片的轮廓。直接展示结果。



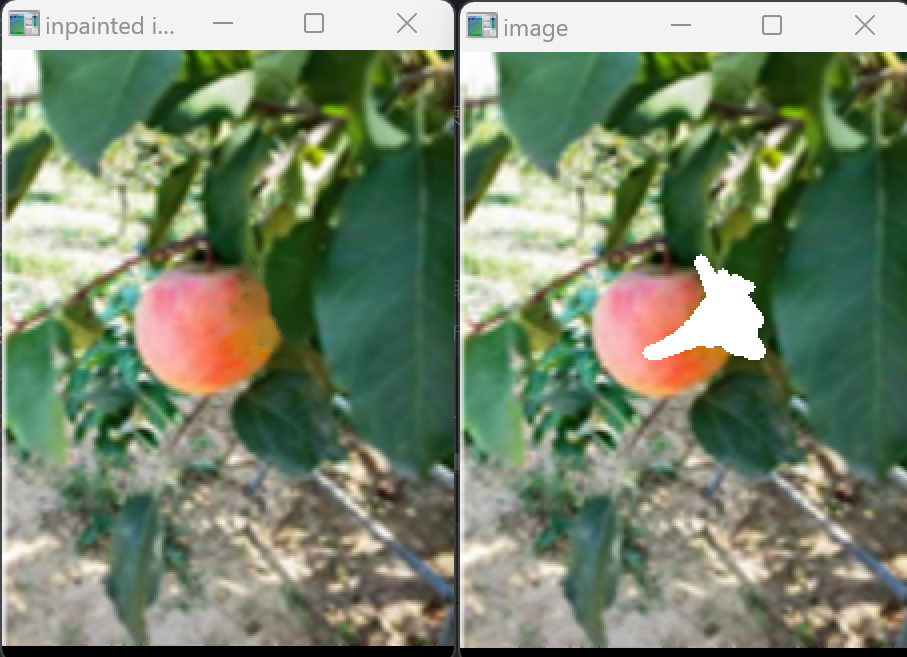
具体代码详见work4.cpp

五、还原图中被部分遮挡的苹果



遮挡苹果 还原苹果

读取目标图像并显示，创建与图像同尺寸的空白掩码，用于标记需要修复的区域，鼠标拖动绘制修复区域，实时更新原图和掩码，使用修复算法 INPAINT\_TELEA来进行平滑处理



具体代码详见work5.cpp

六、采用直方图均衡方法处理图像A，结果类似图B。

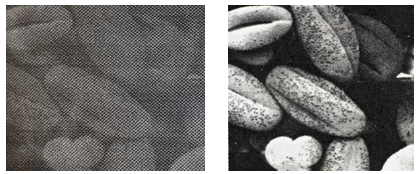
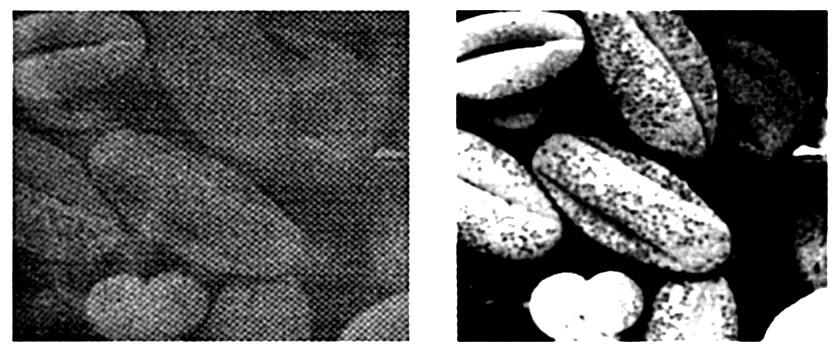


图 原始图（左图）和直方图均衡方法处理后图像（右图）

读取图像，转换为灰度图，转化为CV\_8UC1，进行高斯模糊，直方图均衡化，锐化图像

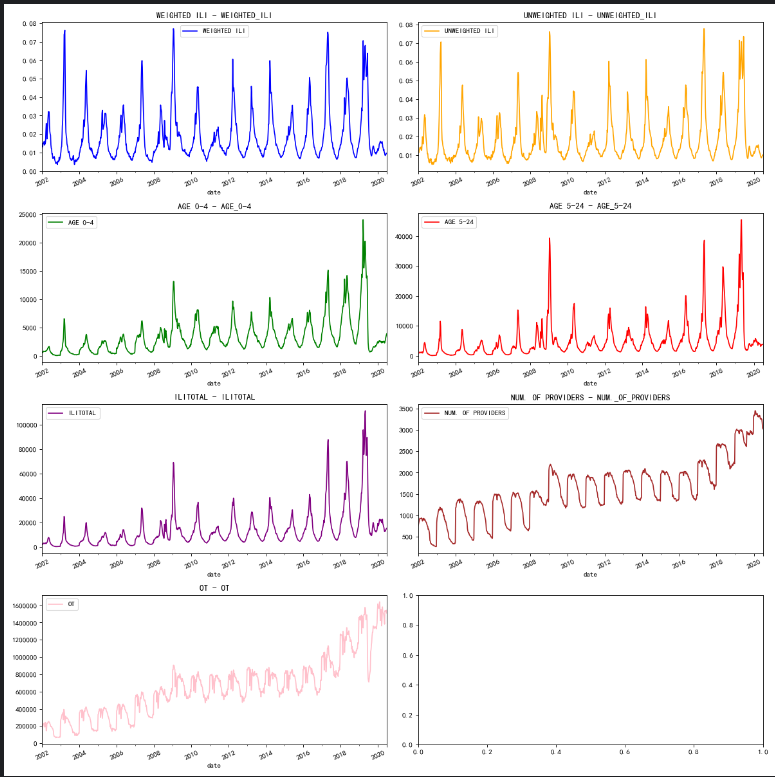


具体代码详见work6.cpp

作业7、8用cpp写过于复杂，libtorch太难用了，这里使用keras2的库，用tensorflow来进行训练，keras为2.10.0，tensorflow为2.10.0，均使用cudnn进行加速，其余的库版本，详见requirements.txt

七、利用LSTM模型预测一个时间序列数据的未来趋势。

这里我选择了美国疾病控制和预防中心每周流感统计数据，时间段为2002/01/01 00:00—2020/06/30 00:00，共966条数据，包括2002年至 2021年美国疾病控制和预防中心每周数据，数值描述了患有流感疾病的患者与患者数量的比率。数据集使用.csv格式保存，包含322个数据字段，具体如下：date：1周颗粒度的时间戳、WEIGHTED ILI：加权比率、UNWEIGHTED ILI：非加权比率、AGE 0-4：0-4岁患者数量、AGE 5-24：5-24岁患者数量、ILITOTAL：患有流感疾病的患者总数、NUM. OF PROVIDERS：提供人数、OT：患者数量



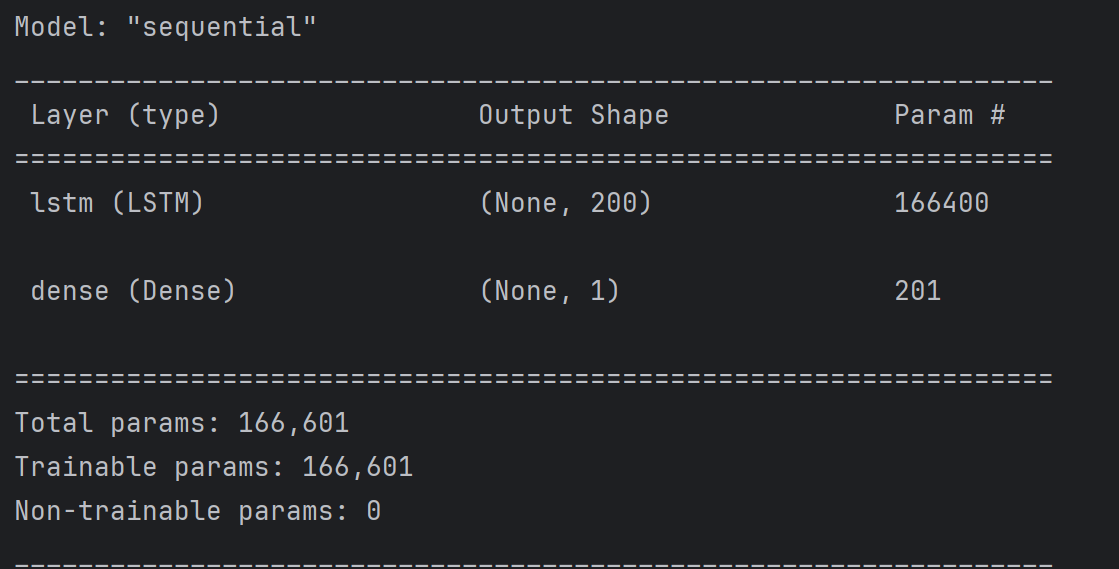
优势：标准 RNN 在反向传播长序列时，梯度会随着时间步指数衰减或爆炸，导致模型难以学习长期依赖。LSTM 的细胞状态通过直接传递和门控机制，允许梯度直接流过多个时间步，大大缓解了梯度消失问题。能够同时捕获长短期依赖，特别适合处理时间序列数据中的动态模式和复杂关系。

实际代码：

具体看代码work7.py

数据预处理的功能由preprocess\_data实现

LSTM模型构建由build\_lstm\_model、build\_advanced\_lstm\_model、build\_conv1d\_lstm\_model、conv\_lstm\_ConvLSTM2D\_model、select\_model来实现 这里实现了多种LSTM模型可以用select\_model进行选择训练。这里用第一个模型为例



数据集创建用create\_dataset

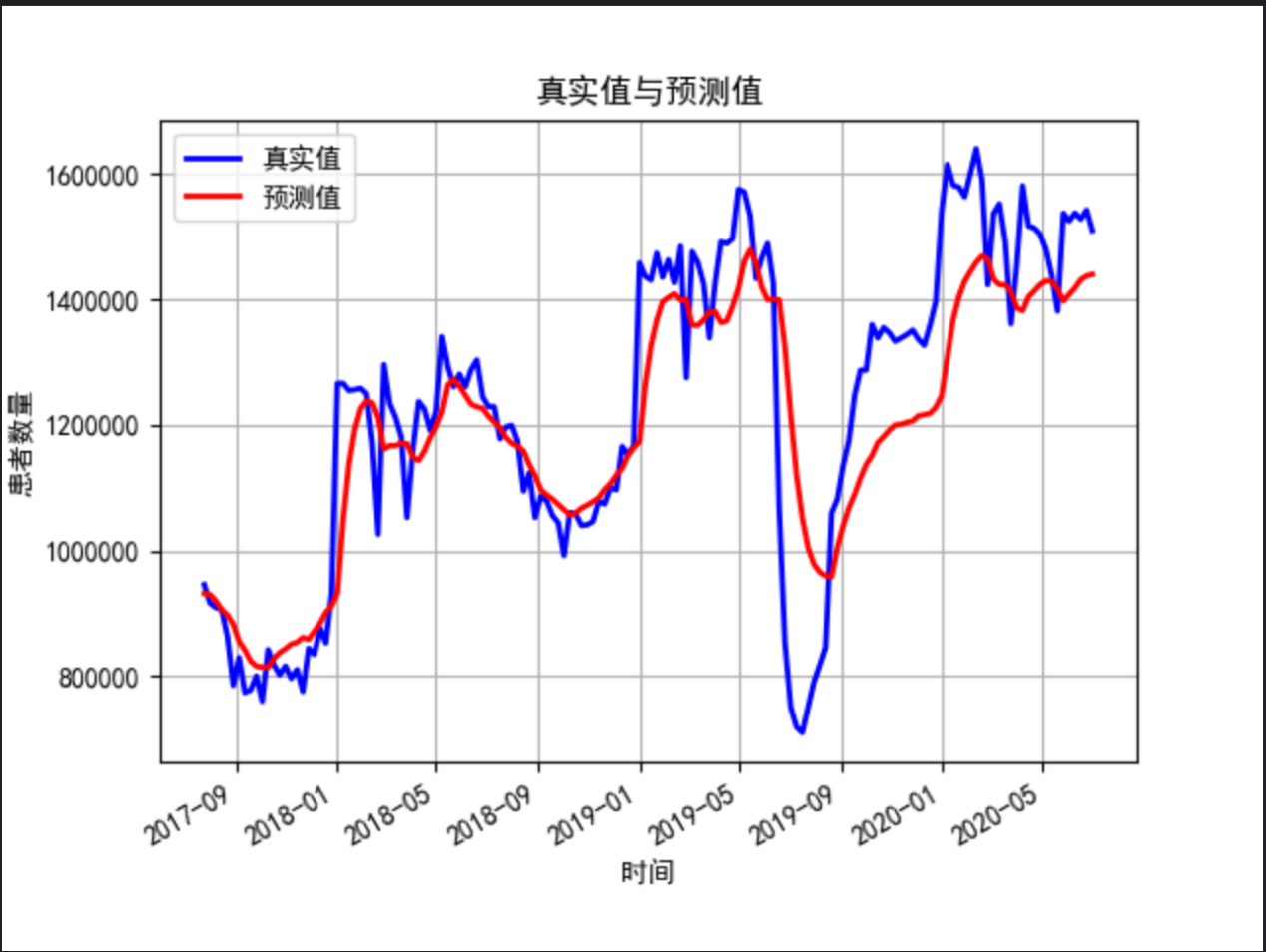
数据集划分和标准化使用的是keras自带的train\_test\_split和sklearn的MinMaxScaler

模型训练：这里使用keras自带的函数，定义了学习率衰减和早停机制。用的是model.fit

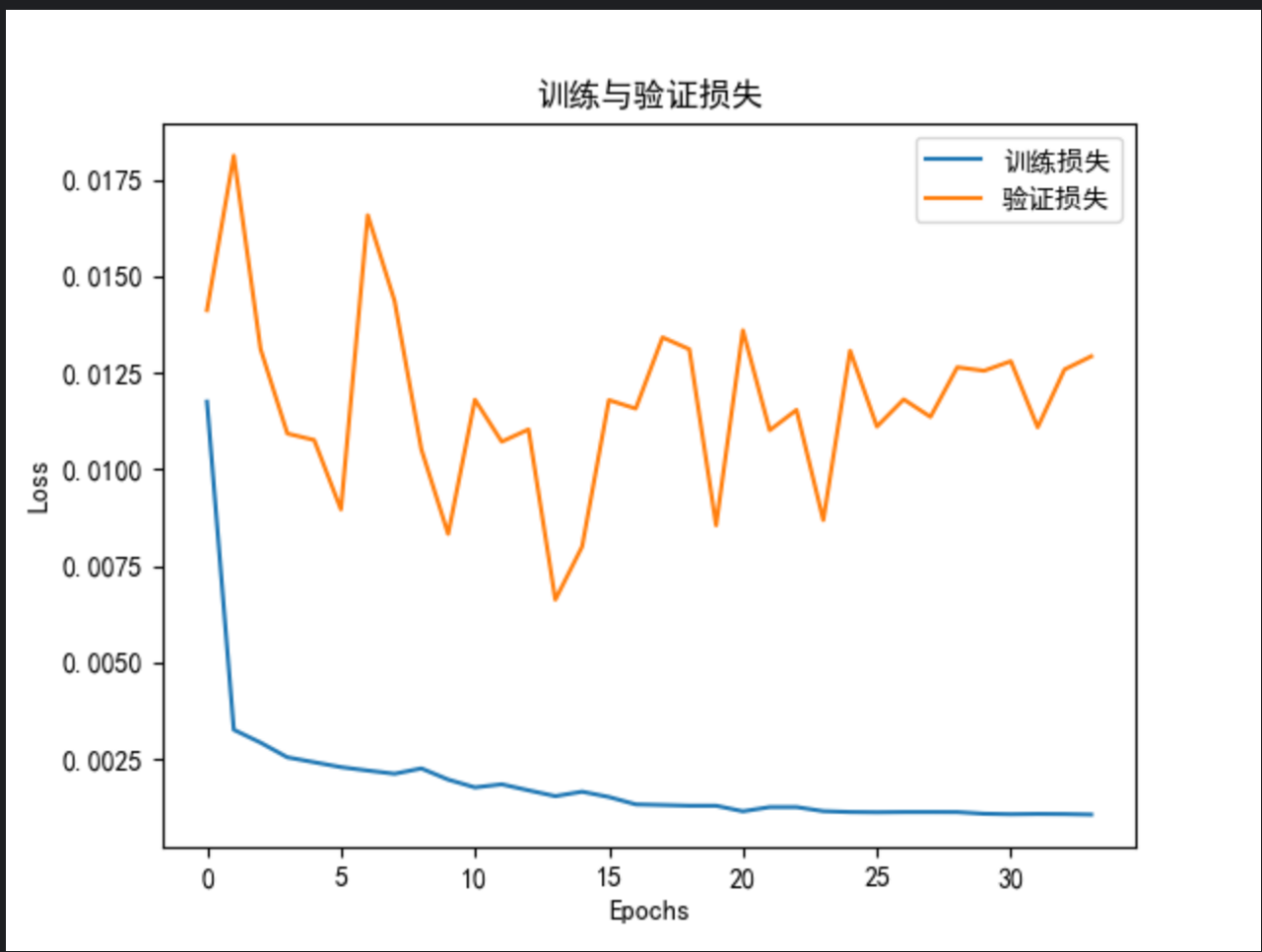
训练过程记录功能是由keras的model.fit函数自带的



预测结果

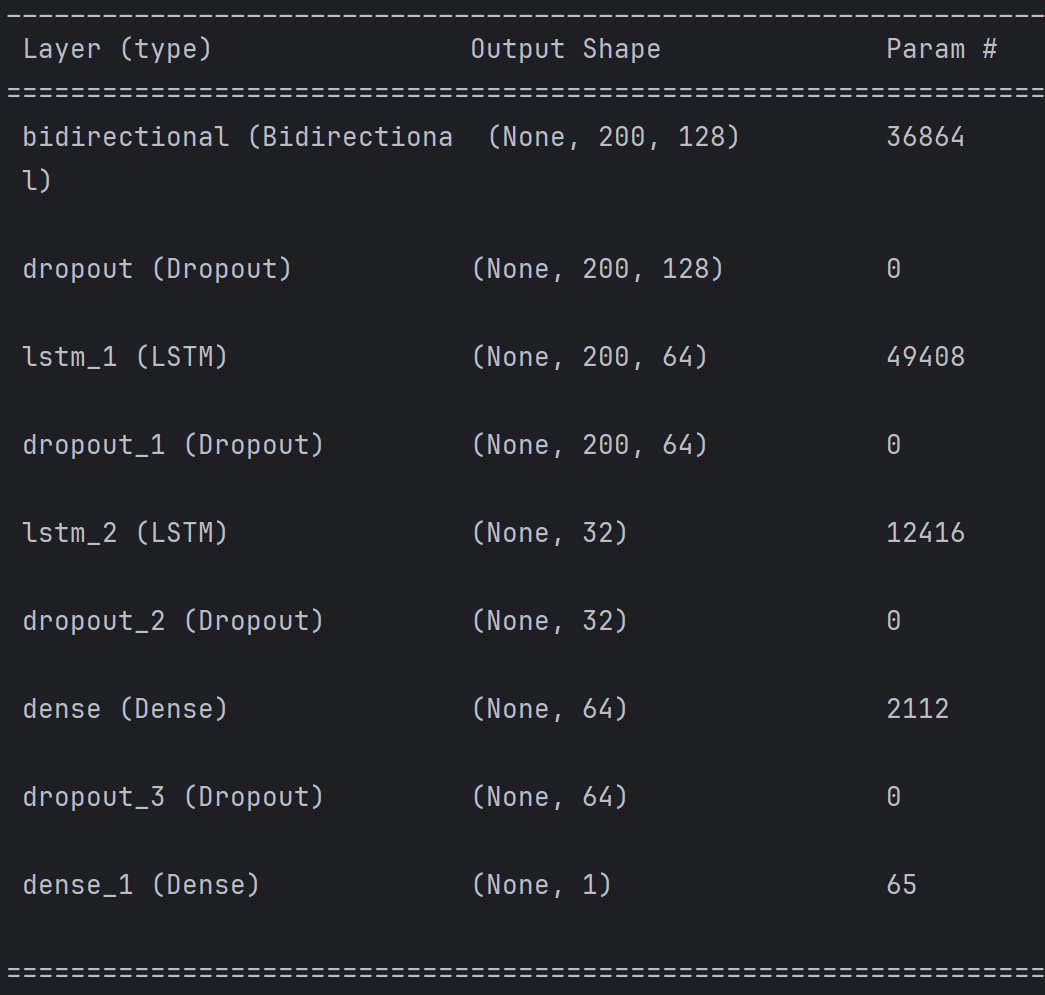


训练图表

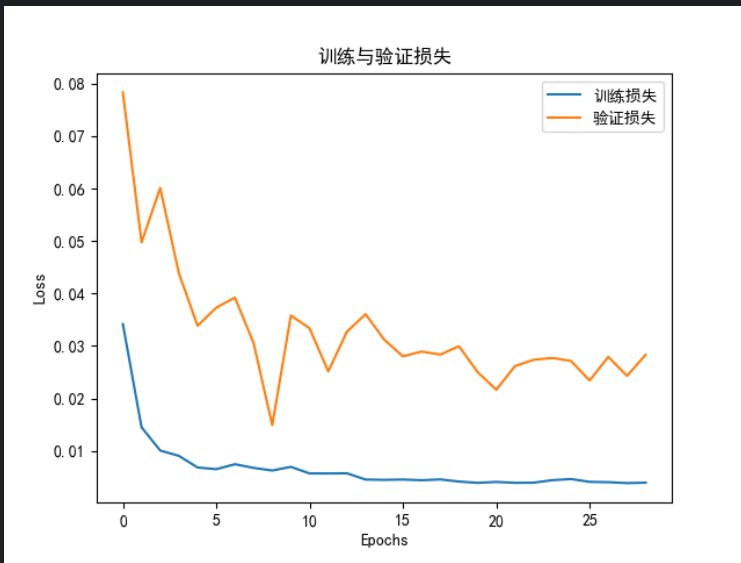


模型分析：目测有拟合，模型过于简单，要增加模型的复杂程度。

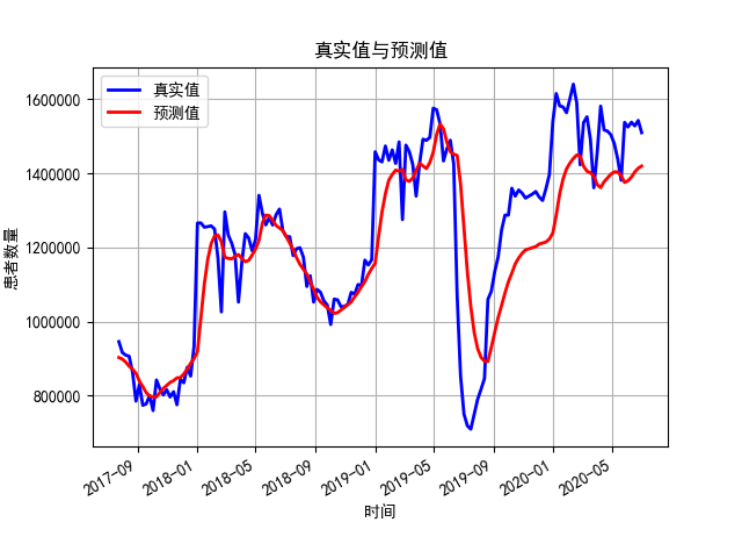
改进模型：



训练图表：



预测结果



结果分析：模型训练过程趋于稳定 验证集的损失在降低，拟合程度在降低，因为模型的复杂程度提高，训练过程逐渐趋于稳定状态。

总体分析：数据的总体趋势是对的，但是很多点的仍然不准确。仍需对模型继续改进。

八、基于迁移学习实现一个图像分类任务，例如：使用预训练模型（如ResNet、VGG）对新的小型图像数据集（如猫狗分类）进行分类。

这里我选择用resnet32来训练，因为resnet的的残差结构通过跳跃连接直接传递信息，具有较少的参数和较低的计算需求，非常适合中小型数据集和有限的计算资源。

迁移学习是一种机器学习方法，它将一个任务上训练得到的知识应用到另一个相关任务上。通常情况下，迁移学习通过使用在大型数据集（如ImageNet）上预训练的模型，并将其应用于新的、数据较少的任务上，显著提高新任务的学习效果。

迁移学习非常适合数据量较小的场景，因为预训练模型已经在大量数据上学习了有效的特征，可以有效地帮助新任务，即使新任务的数据较少。

当目标任务的数据样本较少时，直接训练一个深度学习模型往往会导致过拟合。而迁移学习能够有效避免这一问题，因为预训练模型已经在大规模数据上学到了一些通用的特征。

这里选择的数据集是CIFAR-10，keras2目前可以直接加载CIFAR-10的数据，但是每次要进行在线下载，所以我直接本地读取了数据。  
 这里还是用keras2来进行编写代码。

详细代码见work8.py

数据加载和预处理代码在load\_and\_preprocess\_cifar10函数里。

这里的模型构建使用的是resnet-32的模型。

Keras没有预训练好的resnet-32的模型，所以要先写残差模块的函数residual\_block。

之后再用residual\_network函数构建resnet网络，模型的微调过程，这里对优化器进行微调，来观察模型的表现。

模型结构为：

第一层（输入卷积层）：1 层。

每个阶段：每个残差块由 2 层卷积构成，因此一个残差块有 2 层卷积，多个残差块的总层数为 2 \* 残差块数。

在 ResNet-32 中，第一阶段有 5 个残差块，因此有 5 \* 2 = 10 层卷积。

第二阶段有 5 个残差块，因此也是 5 \* 2 = 10 层卷积。

第三阶段有 5 个残差块，也是 5 \* 2 = 10 层卷积。

全连接层：1 层。

计算总层数

输入卷积层：1 层

每个阶段的残差块：每个残差块有 2 层卷积，堆叠 5 个残差块时，共有 5 \* 2 = 10 层卷积

最终的全连接层：1 层

因此，ResNet-32 的总层数为：

1（输入卷积层）

10（第一阶段的 5 个残差块，每个残差块有 2 层卷积）

10（第二阶段的 5 个残差块，每个残差块有 2 层卷积）

10（第三阶段的 5 个残差块，每个残差块有 2 层卷积）

1（全连接层）

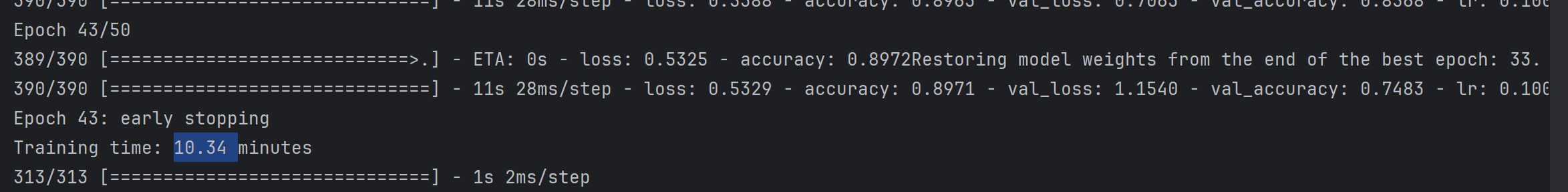
总层数 = 1 + 10 + 10 + 10 + 1 = 32 层

ResNet-32 的层数为 32 层

这里使用的优化器有：sgd、adam和adamw。

第一次训练使用sgd的优化器进行。

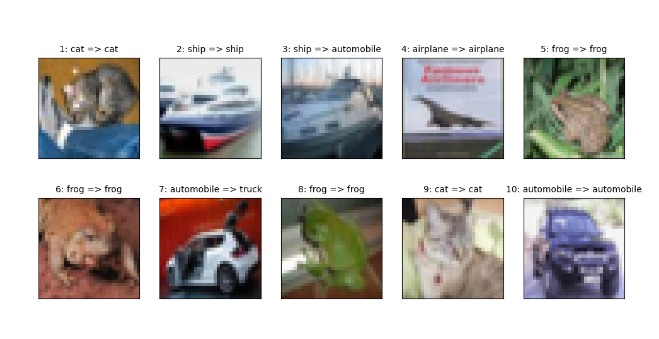
训练时间是：15.94 分钟



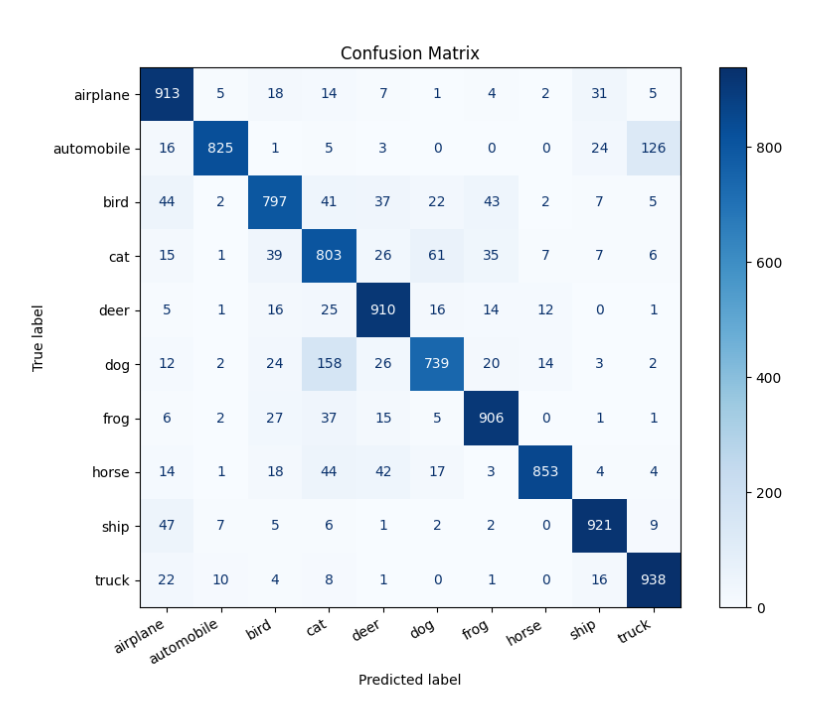
由于写了早停机制，在训练模型最优的情况下停止训练。



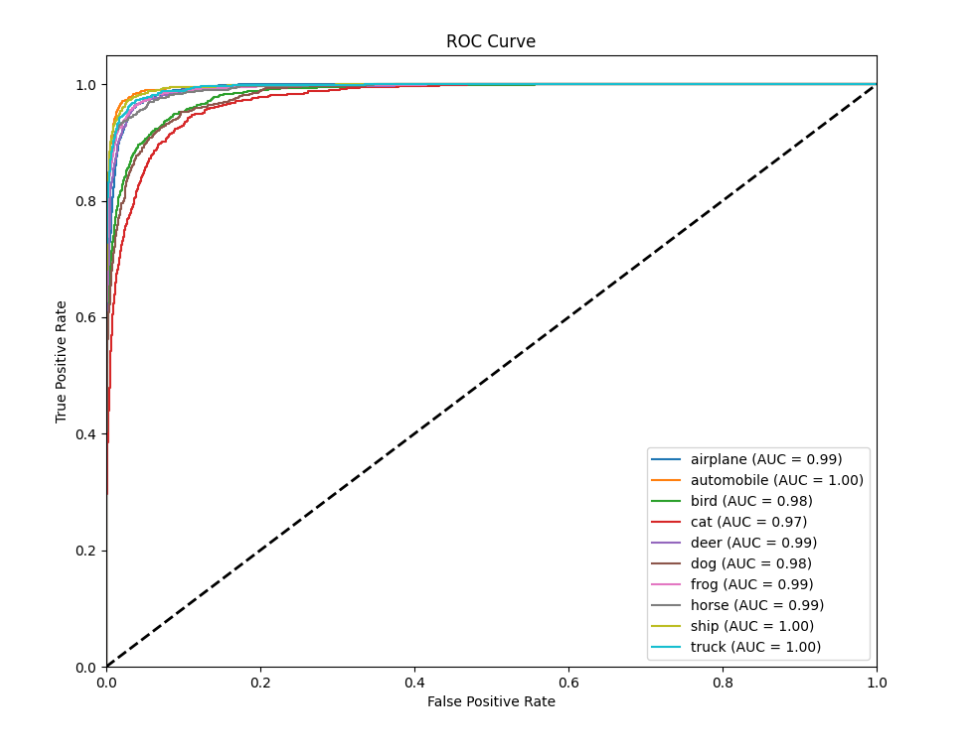
这是训练过程中的训练ac和loss以及测试的ac和loss，发现测试的ac和loss是不稳定的下降，并且在训练快结束时，发现loss有回升的情况，ac有下降的情况，并且写的scheduler的函数并没有用，是因为epoch数不到，以后要增加epoch的数量，以及调低lr的数值，进行微调，使训练过程更加稳定。



随机抽取10个结果进行预测，发现在船和汽车的预测会出现问题，应该提升模型分类程度，或者继续增加残差块，比如用resnet50.



这是混淆矩阵图，其中可以发现分类出现有问题的数量大多数都是猫狗和汽车和卡车之间的分类，这是接下来要关注的优化问题。



这是ROC曲线的结果，从图中可以看到，automobile、ship 和 truck 的 AUC 值为 1.00，这意味着这些类别的分类模型在所有阈值下都能区分正例和负例。

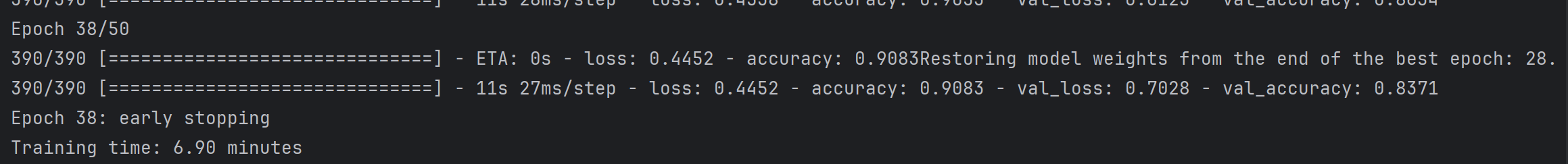
而 cat 类别的 AUC 值最低，为 0.97，虽然仍然是一个很高的值，但相对其他类别来说，其分类性能略差。

因为并没有选择更为复杂的模型，所以最终的结果，并不是太完美，但是更复杂的模型需要更好的算力进行计算，所以优化方向注重在优化器上是目前比较好的选择。

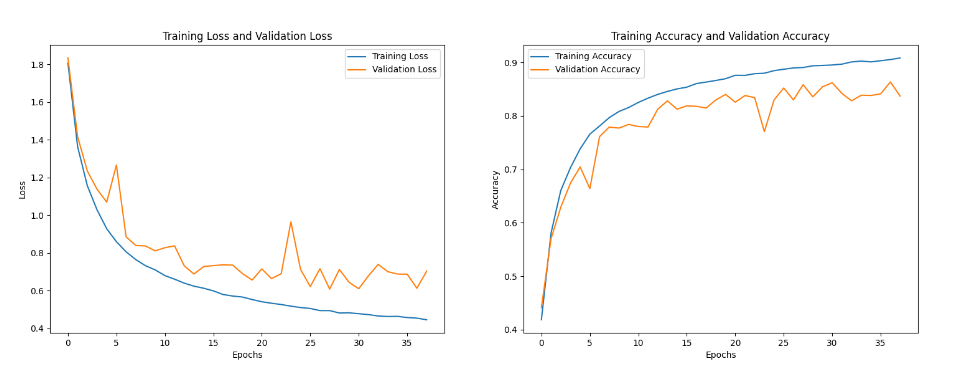
数据增强已经写了，所以就不作为优化方向的选择了。

接下来对优化器进行改变，这里举例使用adam，并且因为adam的自适应能力，callback里就不会添加scheduler来修改lr的值。

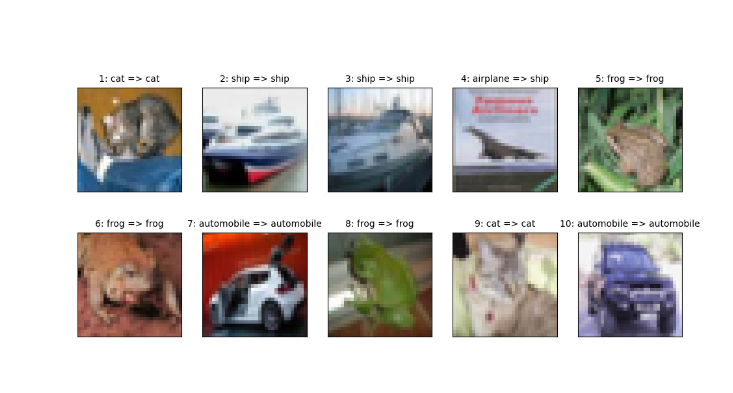
训练时间为：6.9分钟



由于写了早停机制，在训练模型最优的情况下停止训练。



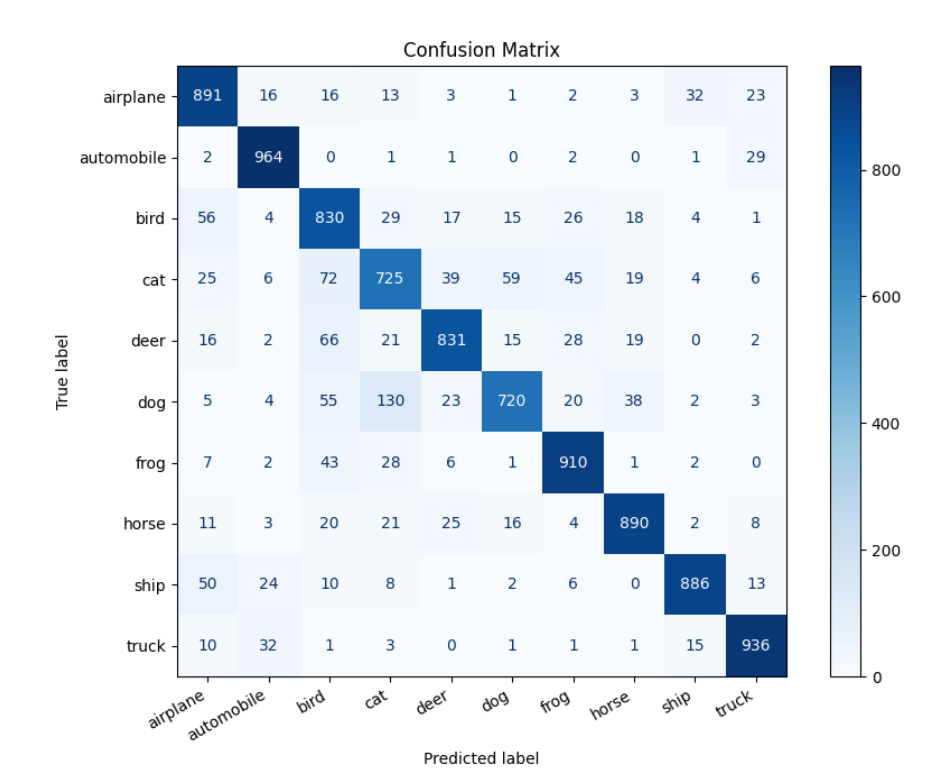
训练和测试的loss和ac曲线，相对于sgd的来说，更加稳定，并且最高精度要比sgd高，训练时间更短。



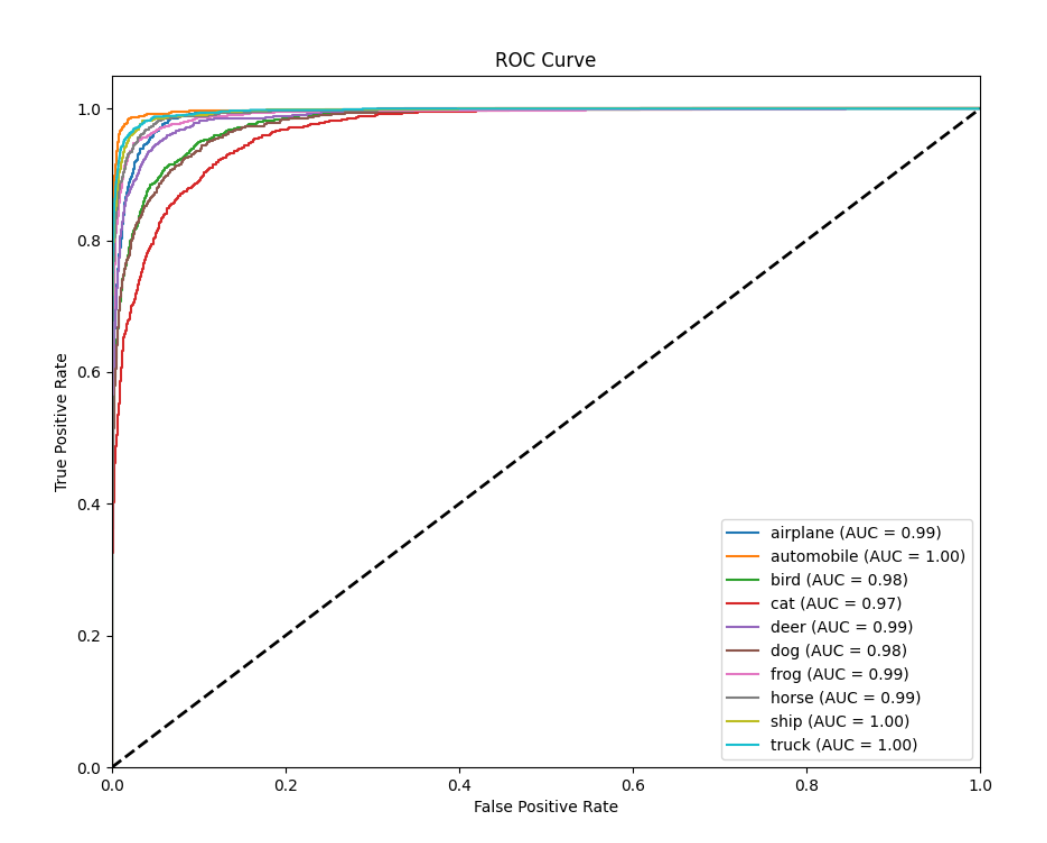
此时再进行10次的预测，并且有重复与上次sgd的预测，这次分类成功，证明优化的方向是可行的。



左图是sgd优化器的预测结果 右图是adam优化器的预测结果



此次的预测结果混淆矩阵，相比上次的混淆矩阵，算是解决了汽车和卡车的分类问题，猫狗的分类是由改善，但是还是有问题，认为需要进一步的提升精度，因为猫狗的形态上相似，所以需要在更高的精度上进行识别。



这是ROC分辨率图，观察发现sgd和adam的图差不多，并没有很明显的差距，但是adam的时间更少，因为其自适应学习率的特性而表现出更快的收敛速度和更好的稳定性。

后面的优化方向，我认为是应该在模型的结构上进行，例如，将resnet32改成resnet50，增加残差结构，增加深度，改进卷积层，或者引入注意力机制，比如SE模块等， 使用超参数，利用自动化超参数优化算法（如贝叶斯优化、Hyperband、SMBO等）