人工智能课程总结报告

黎春颖 2016202136

2018-12-30

第三组

这一个学期以来，无论是从知识储备还是学习方法上，人工智能课都给我带来了很大改变。老师教授了机器学习和神经网络的知识，深入浅出的讲解让我克服了对这些东西的畏惧，也让我们掌握了工业界需要的技能。对我来说是与以往的计算机课程都不同的体验，更注重实践难度也更大。除此之外我们还从智能小车的比赛中也感受到学以致用的乐趣，在完善小车的过程中不仅有编写代码的挑战，还有对创新思维的挑战。

这个学期里学到的东西非常多也非常实用，趁着写课程总结报告的机会，正好按照老师上课的顺序总结一下本学期的所有知识，以备日后所用。

SCIKIT-LEARN：机器学习基础

1. 分类

我们使用了 MNIST 数据集进行学习，首先实现了二分类器，之后再实现多类分类器。将 MNIST 数据集前 60000 张图片作为训练集，后 10000 张图片作为测试集，并且分别将他们的顺序打乱，随机排列，避免样例的顺序影响算法表现。

1) 二分类器

为了简化问题，先使用二分类器尝试识别一个数字。标记训练集中所有标签为 5 的样例为 TRUE，其他的为 FALSE。使用 Scikit-Learn 的 SGDClassifier 类作为分类器，将做好标记的训练集输入进行训练即可。然后使用交叉验证、混淆矩阵、准确率/召回率曲线、ROC 曲线对二分类器的效果进行测试，二分类器的效果并不十分显著。

2) 多类分类器

二分类器只能区分两个类，而多类分类器（也称为多项式分类器）可以区分多个类。有两种策略可以实现多类分类器：OvA 与 OvO。前者也称为“一对其他”，具体而言是训练 10 个二分类器，每一个对应一个数字。然后当对某张图片进行分类的时候，让每一个分类器对这个图片进行分类，选出决策分数最高的那个分类器。后者也称为“一对一”，具体而言是对每一对数字都训练一个二分类器，一个分类器用来处理数 0 和数字 1，一个用来处理数字 0 和数字 2，一个用来处理数字 1 和 2，以此类推。

SGDClassifier 分类器默认使用 OvA 策略，我们也可以使用 OneVsOneClassifier 类或 OneVsRestClassifier 类强制 SKLearn 是使用 OvO 策略或 OvA 策略。训练 RandomForestClassifier 分类器能够直接计算一个样例分到多个类别的概率，调用 predict_proba()可以看到样例对应的类别的概率值的列表。分类完成之后需要像之前一样使用交叉验证检测多类分类器的精度。

3) 多标签分类器

使用多目标数组训练 KNeighborsClassifier 分类器可以得到。

2. 训练模型

我们通常把机器学习模型和训练算法当作黑箱来使用，但是并不知道内部的运行的原则，这一节课学习机器学习模型构建和神经网络的训练。

1) 线性回归

我们可以使用 np.linalg.inv()计算正态方程的 θ hat，以求得使 MSE 损失函数最小的回归曲线。但是这个方法的计算复杂度很高，因此对于特征数量多的样本需要使用梯度下降方法。梯度下降的整体思路是通过的迭代来逐渐调整参数使得损失函数达到最小值。具体来说是在开始时选定一个随机的 θ，然后逐渐去改进它，每一次变化一小步，每一步都尝试降低损失函数，直到算法将 MSE 收敛到一个最小值。

为了让正态方程表现更好，我们还可以使用批量梯度下降，通过设置不同的学习率和不同的迭代次数，求得最合适的参数。批量梯度下降的主要问题是计算每一步的梯度时都需要使用整个训练集，这导致在规模较大的数据集上，算法变得非常的慢。因此需要使用 learning schedule()函数求得的随机学习率，实现随机梯度下降。这样就可以避免算法太慢的问题。批量梯度下降在迭代的每一步都使用整个训练集，而在小批量梯度下降中，它每次仅使用一个随机的小型训练集。

2) 多项式回归

多项式回归的方法是使用 Scikit-Learning 的 PolynomialFeatures 类进行训练数据集的转换，让训练集中每个特征的 n 次方作为新特征，然后将新特征扩展训练集使用线性回归模型 LinearRegression 进行拟合。

3) 逻辑回归

我们使用鸢尾花数据集来分析 Logistic 回归。加载数据之后，使用 LogisticRegression()函数训练回归模型。Logistic 回归模型还可以不必组合和训练多个二分类器，直接推广到支持多类别分类，称为 Softmax 回归。当给定一个实例时，Softmax 回归模型首先计算 k 类的分数 s_k，然后将分数应用在 Softmax 函数上，估计出每类的概率，然后将估计概率最高的那类作为预测结果。

3. 决策树

决策树也是随机森林的基本组成部分，而随机森林是当今最强大的机器学习算法之一。

1) 决策树的训练和可视化

我们使用鸢尾花数据集进行学习。使用 DecisionTreeClassifier()函数进行决策树分类器的训练，然后通过使用 export_graphviz()方法，通过生成图形定义文件将一个训练好的决策树模型可视化。

SKLearn 用 CART 算法训练决策树，算法思想主要是这样：首先使寻找一对特征和阈值，使得训练集能够产生最纯粹的子集，接着算法尝试最小化的损失函数，然后使用

最小化损失函数的特征和阈值将训练集分成两个子集。当它成功的将训练集分成两部分之后，它将会继续使用相同的递归式逻辑继续的分割子集，然后是子集的子集，直到达到预定的最大深度之后停止，或者是直到找不到可以继续降低纯度的分裂为止。

2) 回归

我们使用 Scikit-Learn 的 DecisionTreeRegressor 类构建一个回归树，通过 max_depth 设置回归最大深度，以最小化 MSE 的方式分割训练集。

4. 集成学习和随机森林

集成是指为了得到一个比单一分类器更好的预测结果，而合并了一组分类器的预测。这种技术就叫做集成学习，一个集成学习算法就叫做集成方法。在这节课中学习了一些著名的集成方法，包括 bagging, boosting, stacking 以及随机森林等。

1) 投票分类

硬分类器是指整合多种分类器的预测然后经过投票预测分类。只要数量足够，即使每一个分类器都是一个弱学习器，集成后会成为一个强学习器，我们可以用 VotingClassifier()方法来集成多个分类器。与之对应的还有软分类器，软分类器会将概率更高的投票更大的权重，进而提高正确率。我们可以通过设置 voting="hard"或者 voting="soft"来选择软分类器或者硬分类器。

2) Bagging 和 Pasting

投票分类是指使用不同的训练算法去得到一些不同的分类器，还有另一种方法就是对每一个分类器都使用相同的训练算法，但是在不同的训练集上去训练。有放回采样被称为 Bagging，无放回采样称为 Pasting，也就是说，Bagging 和 Pasting 都允许在多个分类器上对训练集进行多次采样，但只有 Bagging 允许对同一种分类器上对训练集进行多次采样。当所有的分类器被训练后，集成可以通过对所有分类器结果的简单聚合来对新的实例进行预测。SKlearn 可以用 BaggingClassifier 类进行分类器集成，bootstrap=False 设置 Pasting 采样。

由于使用 Bagging 会导致训练集中一些样例可能被一些分类器重复采样，有可能不会被采样，因此在训练后需要设置 oob_score=True 来评估集成效果，然后调用 bag_clf.oob_score_查看这个数值。

3) 随机森林

随机森林是决策树的一种集成，通常是通过 bagging 方法进行训练，用 max_samples 设置训练集的大小。接着建立一个 BaggingClassifier，把 DecisionTreeClassifier 放入其中。另一种更加简便的方法是直接使用 RandomForestClassifier（对于回归是 RandomForestRegressor），n_estimators 限制数数目，max_leaf_node 限制叶子节点数，n_jobs 设置使用 CPU 核数。

还有一种方法是对特征使用随机阈值，这种极端随机的树被简称为极端随机树，这种方法使得 Extra-Tree 比规则的随机森林训练得更快。我们可以使用 SKLearn 的

ExtraTreesClassifier 类来创建一个 Extra-Tree 分类器，它的 API 跟 RandomForestClassifier 相似的， ExtraTreesRegressor 跟 RandomForestRegressor 也是相同的 API。这两者的优劣需要在使用时候根据交叉验证的结果判别。

5. 降维

降维会导致模型丢失一些信息，所以降维可以加快模型训练的速度，同时也会让模型表现的稍微差一点。

1) 主成分分析（PCA）

首先找到接近数据集分布的超平面，也就是选择一个使得将原始数据集投影到该轴上的均方距离最小的轴，然后将所有的数据都投影到这个超平面上。我们可以使用 Numpy 提供的 svd()函数获得训练集的所有主成分，然后使用 X_centered.dot()函数通过将数据集投影到由前 d 个主成分构成的超平面上，从而将数据集的维数降至 d 维。

使用 SKLearn 的方法就简单很多，调用 PCA 库，并使用 PCA(n_components=n)函数即可，其中 n_components 用于设置目标数据集维度，或者也可以设置为 0.0 到 1.0 之间的浮点数，表明希望保留的方差比率。我们还可以通过 explained_variance_ratio_变量获得每个主成分的方差解释率，它表示位于每个主成分轴上的数据集方差的比例，一般来说越高越好。在我们对数据集进行降维操作之后，输出结果时需要使用 inverse_transform()函数将其解压。

2) 局部线性嵌入（LLE）

LLE 首先测量每个训练实例与其最近邻之间的线性关系，然后寻找能最好地保留这些局部关系的训练集的低维表示。尤其是在没有太多噪音的情况下，它特别擅长展开扭曲的流形。我们可以直接使用 Scikit-Learn 的 LocallyLinearEmbedding 类来使用 LLE 方法。

3) 其他降维方法

多维缩放（MDS）在尝试保持实例之间距离的同时降低了维度。Isomap 通过将每个实例连接到最近的邻居来创建图形，然后在尝试保持实例之间的测地距离时降低维度。t-分布随机邻域嵌入（t-SNE）可以用于降低维度，同时试图保持相似的实例临近并将不相似的实例分开。它主要用于可视化，尤其是用于可视化高维空间中的样本。线性判别分析（LDA）实际上是一种分类算法，但在训练过程中，它会学习类之间最有区别的轴，然后使用这些轴来定义用于投影数据的超平面。LDA 的好处是投影会尽可能地保持各个类之间距离。

TENSORFLOW：神经网络和深度学习

1. 人工神经网络

这一节主要学习了常见的神经网络模型构造过程，过程如下：

首先使用 neuron_layer()函数，传入给定的神经元数量，激活函数和图层的名称，一次创建一个图层。它把所有输入都连接到图层中的所有神经元，创建一个完全连接的层，我们可以设置 activation 参数选择激活函数。然后我们将为模型建立多层神经网络构造神经网络模型。

有了神经网络模型之后，我们需要定义我们用来训练的损失函数。TensorFlow 提供了几种计算交叉熵的功能。 例如 sparse_softmax_cross_entropy_with_logits()函数可以根据对数计算交叉熵，并且期望整数形式作为标签，返回一个包含每个实例的交叉熵的 1D 张量。然后，可以使用 TensorFlow 的 reduce_mean()函数来计算所有实例的平均交叉熵。

创建神经网络并定义好损失函数之后，需要定义一个 GradientDescentOptimizer 来调整模型参数以最小化损失函数。使用 tf.train.GradientDescentOptimizer()函数并设定合适的学习率，使得损失最小化对模型进行训练。

建模阶段的最后一个重要步骤是确定如何评估模型。对于每个实例，通过检查最高对数是否对应于目标类别来确定神经网络的预测是否正确。我们可以使用 in_top_k()函数获取一个布尔值的 1D 张量，将这些布尔值转换为浮点数，然后计算平均值便可以得到神经网络的整体准确性。

执行的时候使用 input_data.read_data_sets()函数每次随机加载一个小批量数据，对这批数据进行操作，重复这个操作直到将训练集所有数据都迭代完成。

训练完成神经网络之后，我们可以使用它进行预测。首先使用 saver.restore()方法从磁盘中加载模型，然后使用 logits.eval()评估节点属于的类概率，最后使用 argmax()函数选择有最高对数值的类作为标签即可。

2. 训练深度神经网络

由于梯度爆炸问题的存在，深度神经网络的训练会变得比较困难，并且当训练集增大之后，训练将非常缓慢，具有严重的过拟合训练集的风险。科学家们发现部分爆炸的梯度问题是由于激活函数的选择不好造成的，因此我们可以选择 He、 ELU 或者 ReLU 作为激活函数来减少训练开始阶段的梯度爆炸问题。

还有一种称为“批量标准化”的技术可以解决这个问题。具体而言是在每层的激活函数之前在模型中添加操作，对输入进行 zero-centering 和规范化操作，然后每层使用两个新参数对结果进行尺度变换和偏移。换句话说，这个操作可以让模型学习到每层输入值的最佳尺度和均值。TensorFlow 的具体实现方法如下：首先使用 tf.layers.dense () 方法构建全连接层，然后使用 tf.layers.batch_normalization()对每个隐藏层运行批量标准化，再手动使用 tf.nn.elu()方法激活隐藏层，重复上述操作即可。为了避免一遍又一遍重复相同的参数，我们也可以使用 Python 的 partial()函数。

减少梯度爆炸问题的另一种常用技术是在反向传播过程中简单地剪切梯度，使它们不超过某个阈值。使用 tf.train.GradientDescentOptimizer()函数设置梯度学习的学习率，然

后优化器的 minimize()函数计算梯度，最后创建一个操作来使用优化器的 apply_gradients()方法应用裁剪梯度。

我们还可以复用以前训练好的模型。创建占位符和全连接层，使用 get_collection()函数加载以前的模型即可。如果模型是使用其他框架进行训练的，则需要使用 get_variable()函数手动加载权重，然后使用 tf.assign()函数将它们分配给相应的变量。

3. 卷积神经网络

卷积神经网络也被称为 CNN，多用于图像识别任务，如图像搜索服务，自动驾驶汽车，视频自动分类系统等。但是 CNN 也并不局限于视觉感知，它在其他任务中也很成功，如语音识别或自然语言处理（NLP）等。

1) 卷积层

第一卷积层中的神经元不是连接到输入图像中的每一个像素，而是仅仅连接到它们的固定大小的局部感受野内的像素。进而，第二卷积层中的每个神经元只与位于第一层中的小矩形内的神经元连接，每层都重复这样的操作。在训练过程中，CNN 为其任务找到最有用的卷积和，学习并训练将它们组合成更复杂的模式

输入图像一般也由多个子图层组成：每个颜色通道一个。通常有三种：红色，绿色和蓝色（RGB）；灰度图像通常只有一个通道。在一个特征映射中，所有神经元共享相同的参数，但是不同的特征映射可能具有不同的参数。神经网络同时对其输入应用多个卷积核，使其能够检测输入中的任何位置的多个特征。

具体实现方法如下：首先使用 load_sample_image()函数加载图片，创建 filter；然后使用 tf.nn.conv2d()方法获取卷积层；最后使用 imshow()方法将图片打印即可。

2) 池化层

池化层是对输入图像进行二次抽样以减少计算负担，内存使用量和参数数量。 减少输入图像的大小也使得神经网络必须容忍一些图像的改变。池化层中的每个神经元都连接到前一层中一个矩形感受野内神经元的输出，所以需要定义感受野的大小，跨度和填充类型。但是，池化层中的神经元没有权重，它所做的只是使用聚合函数来聚合输入。只有每个核中的最大输入值才会进入下一层。 其他输入被丢弃。池化层通常独立于每个输入通道工作，因此输出深度与输入深度相同，因此池化层结果导致图像的高度和宽度保持不变，但是通道数目减少。

具体实现方法如下：首先使用 load_sample_image()函数加载图片，创建 filter；然后创建占位符，使用 tf.nn.max_pool()方法创建一个最大池化层；最后使用 imshow()方法将图片打印即可。

3) CNN 架构

典型的 CNN 体系结构有一些卷积层，接着一个池化层，然后是另外几个卷积层，接着是另一个池化层。 随着网络的进展，图像变得越来越小，但是由于卷积层的缘故，图像通常也会越来越深。 在堆栈的顶部，添加由几个全连接层组成的神经网络，最终

层输出预测。一般来说两个 3×3 卷积核堆叠在一起来的效果与 9×9 神经核相同，但是计算量更少。

4. 循环神经网络

循环神经网络是一类预测未来的网络。它们可在任意长度的序列上工作，而不是只能在固定长度的输入上工作。RNN 可以同时进行一系列输入并产生一系列输出，输入可以是图像，输出可以是该图像的标题。

使用 TensorFlow 实现时间序列静态展开的具体过程如下：首先，使用 tf.placeholder()创建两个占位符，tf.contrib.rnn.BasicRNNCell()类构建 RNN 模型；然后，使用 tf.contrib.rnn.static_rnn()函数通过链接单元来创建一个展开的 RNN 网络；接下来，使用 transpose()函数交换前两个维度，以便时间步骤变为第一维度，unstack()函数沿第一维提取张量的列表；最后，使用 stack()函数将所有输出张量合并成一个张量输出即可。

但是由于反向传播期间占用内存很大，甚至可能会发生内存不足的情况，所以需要考虑时间序列的动态展开。具体实现时将.static_rnn()函数换为 dynamic_rnn()函数，使用 while_loop()操作，在单元上运行适当的次数。更为方便的是，它还可以在每个时间步接受所有输入的单个张量，并且在每个时间步上输出所有输出的单个张量，因此不需要堆叠，拆散或转置操作。

如果需要构建一个神经元数量和深度都更多的 RNN 网络，就需要创建深度 RNN。先使用 tf.contrib.rnn.BasicRNNCell()类创建一些神经元，然后使用 MultiRNNCell()方法将这些神经元堆叠起来，最后用 dynamic_rnn()方法动态展开即可。

5. 自编码器

自编码器是能够在无监督的情况下学习输入数据的编码的人工神经网络。这些编码通常的维度通常比输入数据的维度低得多，更重要的是，自编码器可以作为强大的特征检测器，用于无监督的深度神经网络预训练。除此之外，自编码器还能够随机生成与训练数据非常相似的新数据。

一个自编码器会查看输入信息，将它们转换为高效的内部表示形式，然后吐出一些接近输入的东西。因此自编码器由两部分组成：将输入转换为内部表示的编码器（识别网络），然后是将内部表示转换为输出的解码器（生成网络）。

自编码器有多个隐藏层，被称为栈式自编码器（SAE）。SAE 的架构通常是关于中央隐藏层（编码层）对称的。TensorFlow 实现方式与之前的神经网络构建代码相同，只是输入的样本无需输入标签。

对于比较深的自编码器来说，一般不是一次完成整个栈式自编码器的训练，而是一次训练一个浅自编码器，然后将所有这些自编码器堆叠到一个栈式自编码器中。为了实现这种多阶段训练算法，最简单的方法是对每个阶段使用不同的 TensorFlow 图。训练完一个自编码器后，运行训练集并捕获隐藏层的输出，然后把这个输出作为下一个

自编码器的训练集。最后需复制每个自编码器的权重和偏置，然后使用它们来构建堆叠的自编码器。

6. 强化学习

强化学习是指智能体在环境中观察并且做出决策，随后它会得到相应的奖励，如果结果满足我们的期望，那么奖励为正，否则为负。它的目标是去学习如何行动能最大化期望奖励。

1) OpenAI 介绍

OpenAI gym 是一个提供各种各样的模拟环境的工具包，可以在上面训练或开发新的 RL 算法。使用 make()函数创建新环境，reset()初始化，render()方法可视化环境设置，action_space 查看可能的决策空间，step()函数执行给定的动作并返回相应的值。

除了进行通常的监督学习之外，在强化学习中，智能体获得指导的唯一途径是通过奖励，但是奖励通常是稀疏的和延迟的。为了解决这个问题，我们需要设置衰减率使得基于这个动作后的多个动作得分的总和来评估这个个动作。

2) 策略梯度

增强算法是一种流行的 PG 算法，算法思路如下：首先，让神经网络策略玩几次游戏，并在每一步计算梯度，这使智能体不应用这些梯度选择行为。运行几次后，计算每个动作的得分。如果一个动作的分数是正的，则可应用较早计算的梯度，以便将来有更大的的概率选择这个动作。但是如果分数是负的，就要应用负梯度来使得这个动作在将来采取的可能性更低。我们的方法就是简单地将每个梯度向量乘以相应的动作得分。最后，计算所有得到的梯度向量的平均值，并使用它来执行梯度下降步骤。

大作业工作总结

1. 第一阶段

这一阶段中，我们小组完成了小车的拼装、基本功能和物体跟随功能。其中基本功能包括前进、刹车、后退、左右拐弯、直角拐弯、红外循迹、红外避障、蓝牙控制等，跟随功能是基于 OpenCV 根据颜色判别物体并实现追踪，本阶段我主要参与追踪功能的实现。

我们使用一款基于 OpenCV 的图像处理 App，在手机上处理摄像头实时捕获的图像，获取追踪物体在显示屏内的 X、Y 坐标，然后通过蓝牙将相应的数据传输给小车。具体操作是定义一个用于接收和传输数据的类，接着根据实验测量出需要给小车发送直走、后退、左拐、右拐信号时的物体的 X、Y 坐标范围，例如当 X>130、 Y<156 时，意味着物体出现在手机屏幕的左侧，此时将会给小车左轮发送低频信号、右轮发送高频信号，实现小车左拐追踪物体。同时还需要调用限制函数对发送给左右轮的 PWM 信号进行适当的调整，使得小车追踪的速度更为平稳。根据以上代码和优化，我们最终实现了小车直线跟随、曲线跟随和随着物体的靠近小车后退的功能。

本阶段我遇到的困难主要是在拼装小车和追踪功能中信号发送上。由于缺乏相关的经验，拼装也成了一大阻碍，我们先后购买了两款小车才最终完成了搭建工作。在实现小车跟随功能时，也出现了一些问题。虽然算法思路很简单，但是在最终发送信号的时候出现诸如小车拐弯过急跟丢物体（处理信号的速度滞后于小车拐弯的速度）、小车无法调用已有函数等问题，也使得我们纠结了许久。最终展示时，我们小组的总体完成度还是比较高的，在后续丰富小车功能的时候也体会到了提前完成基本功能的好处。

2. 第二阶段

这一阶段中，我们小组给小车增加了视觉和听觉两方面的功能，前者包括了人脸检测、人脸识别，后者实现了语音控制。我主要负责了视觉方面功能，实现了小车检测到人脸则前行以及追踪识别到的特定人脸的功能。

人脸检测功能是通过使用 OpenCV 自带的 Haar 人脸特征分离器器。将手机与电脑连在同一个局域网内，电脑端接受手机摄像头实时采集的图像，对每一帧构建分离器，然后使用分离器的内置函数检测人脸并显示在电脑屏幕上。当检测到人脸时，使用蓝牙给小车发送前进信号；反之，则控制小车停止。主要原理是使用特定大小的窗口扫描图片，检测每个窗口内是否存在人脸并标记；然后对图片按照特定比例缩小，再次重复扫描操作，这样可以检测到像素更大的人脸并标记；循环这个操作直至图片与窗口大小相同为止。这个功能依赖于 Haar 特征分离器的选择，虽然算法比较麻烦，但是由于小车蓝牙借口无法过于频繁的接收数据，因此算法速度满足实验要求。

人脸识别功能主要调用了 face_recognition 库。首先使用爬虫爬取了 120 张高清的白敬亭单人照和魏大勋单人照作为训练集，标签分别为两人的名字，使用 HOG 算法提取照片内人脸的特征，并使用这些特征对已经训练好的神经网络进行调整，以便识别人脸。接下来使用与人脸检测功能相同的方法，获取手机摄像头实时捕获的图像，使用 HOG 算法识别图像内人脸，并返回人脸特征向量。然后将这个特征向量与训练集内提取的特征向量进行比较，选取匹配率最高的名字作为该人脸的标签。在获取到人脸标签和人脸坐标后，便可以通过蓝牙对小车发送相应信号，控制小车运动。例如，如果希望小车向检测到白敬亭脸的方向追踪，则当小车检测到标签为“白敬亭”的人脸，并且人脸左上角横坐标大于 480，则发送右转的信号；如果小于 230，则发送左转的信号；反之则前行。如果没有检测到标签为“白敬亭”的人脸，发送停止信号。

由于我们在上一阶段已经实现了小车的基本功能，因此在本阶段中将 AI 算法与小车功能结合时很容易实现。我在本阶段遇到的主要问题在于人脸识别算法过慢，难以达到实时识别的效果。并且使用了另外爬取的 120 张风景照与路人照片作为测试集后发现，算法的召回率很高但精确度和准确度低，在后续实验中会继续优化这个问题。总体而言，我们小组本阶段的小车功能完成度比较高，但仍有很大提升空间。

3. 第三阶段

本阶段还没有展示，仍然在准备过程中，所以这部分阐述我们的计划。我们计划实现小车在模拟现实情况下的自动驾驶，涉及的内容包括道路识别、路标识别等，是一个

综合了我们前两阶段功能并优化完善的项目。我在这阶段中负责路标识别部分，在写这篇日志的时候已经完成了模型的训练，接下来要将模型装入手机并在小车上测试。

综上，本学期中人工智能课上学习到的东西很丰富，学习过程也十分有趣。希望日后能在实践中对深度学习算法多加学习和重复使用，提高熟练度和并且掌握更好的优化算法。