华中农业大学

《计算机视觉》

课程报告

学　　院：_____信息学院_____

班　　级：_____计科 1803_____

姓　　名：_____邢广杰_____

学　　号：___2017308210404____

指导教师：_____翟瑞芳_____

日　　期：__2021 年 6 月 1 日___

目录

第一章 项目分析与设计

1.1 项目背景

自动驾驶汽车（Autonomous vehicles；Self-piloting automobile）又称无人驾驶汽车、电脑驾驶汽车、或轮式移动机器人，是一种通过电脑系统实现无人驾驶的智能汽车。在 20 世纪也已经有数十年的历史，于 21 世纪初呈现出接近实用化的趋势，比如，谷歌自动驾驶汽车于 2012 年 5 月获得了美国首个自动驾驶车辆许可证，预计于 2015 年至 2017 年进入市场销售。

自动驾驶汽车依靠人工智能、视觉计算、雷达、监控装置和全球定位系统协同合作，让电脑可以在没有任何人类主动的操作下，自动安全地操作机动车辆。 2014 年 12 月中下旬，谷歌首次展示自动驾驶原型车成品，该车可全功能运行。 2015 年 5 月，谷歌宣布将于 2015 年夏天在加利福尼亚州山景城的公路上测试其自动驾驶汽车。

近两年，自动驾驶发展的尤为迅速。有数据表明，2017 年中国汽车产量高达 2970 万台，汽车的保有量也首次突破 2 亿，其中，拥有自动驾驶功能的汽车超过 700 万台。随着“互联网+”与移动终端的完美结合，车联网在汽车制造业上得到了大规模的广泛应用，不仅将智能交通成为现实，也将自动驾驶变成了可能。同时，自动驾驶也是当前全球汽车与交通出行领域智能化和网联化发展的主要方向，他的独特优势将在未来的交通中具有重要的价值。

今年 4 月，发布在 bilibili 弹幕视频网站上的投稿《全网首试，公开道路体验极狐 × 华为自动驾驶！ 》向大众展示了中国华为公司研发的自动驾驶技术，目前累计 200 万播放量，视频中在复杂道路上完全自动行驶的汽车让人叹为观止，落地产品的体验让人们感受到了未来近在眼前。视频中尤为瞩目的是位于驾驶位中央的智能显示屏，汽车的行驶过程中将道路的实时状况通过显示屏展现，智能汽车可以实时的检测到道路上的汽车、行人以及各种目标并且以三维的形式输出，这项技术在自动驾驶中尤为关键。它是如何做到如此迅速、准确的识别出道路目标并将其显示的呢，这是一个非常有挑战性的课题。

在本学期开展的**《计算机视觉》**课程中，我们学习了计算机是如何理解、表示三维空间中的信息的，学习了计算机视觉中强大的算法和技术。因此，我们已经具备了理解和实践**三维目标检测**所需要的知识和能力。通过一系列的研究和探索，可以开发出具有一定识别能力的三维目标检测系统，这将帮助我们理解计算机视觉的相关理论，帮助我们将理论结合到实践中。

经过长时间的调研，阅读了若干篇发表在计算机视觉顶级会议上的文献后，最终确定了以发表在 2017 年的 CVPR 上的论文**《3D Bounding Box Estimation Using Deep Learning and Geometry》[Arsalan Mousavian e.t.]**的研究成果作为实现项目系统的理论基础。

本项目的设计与实现依据上述背景展开。

1.2 项目要求及目的

本项目作为《计算机视觉》的课程实践项目，根据上述的项目实施背景，需要按照以下要求展开，以达成项目目的：

1. 项目需要一定的理论支撑，通过学习三维目标检测领域相关的研究成果，完成对三维目标检测问题定义的理解，明确解决该问题的重点难点。
2. 尽量理解用于该项目相关技术的原理，由于三维目标检测任务的困难性，需要利用目前掌握的有限知识对实施于项目中的相关技术进行解释。
3. 与《计算机视觉》理论课程紧密结合，由于该项目的性质，需要尽可能多的利用理论课程中学习到的相关知识，利用学习到的知识解决问题，加深对理论学习的理解，发挥理论课程的作用。
4. 项目需要具备一定的实际意义，针对该项目，最终得到的系统需要具备一定的识别能力，以及识别结果需要具备一定的准确性，有助于发现问题、优化问题，达到项目的研究目的。
5. 本课程项目使用的技术方法应尽量简单易于理解，开展的过程应尽量简明扼要，有助于对知识的掌握和理解。
6. 对于本项目的结果分析需要尽量详细，有一定的依据，应把结果分析作为重点，以达成学习的主要目的。

1.3 开发环境和工具

本机环境

OS: macOS Catalina 10.15.7 Arch: x86_64

torch: 1.8.1

torchvision: 0.9.1

opencv-python: 4.5.2.52

Colaboratory

由于 GPU 资源稀缺和项目成本有限，本项目中大部分代码的运行和模型训练均在 Google 公司的 *Colaboratory* 平台上展开

Colaboratory 是 Google 的一个研究项目，旨在提供开发者一个云端训练神经网络的工具。它是 Jupyter 一个笔记本环境，不用做任何配置，完全运行在云端。Colaboratory 存储在 Google Drive 中，可以进行共享。Colaboratory 向开发者提供了免费的 Tesla K80 GPU 使用。

PyTorch

PyTorch 是一个针对深度学习，并且使用 GPU 和 CPU 来优化的 tensor library (张量库)

2017 年 1 月，由 Facebook 人工智能研究院（FAIR）基于 Torch 推出了 PyTorch。它是一个基于 Python 的可续计算包，提供两个高级功能：

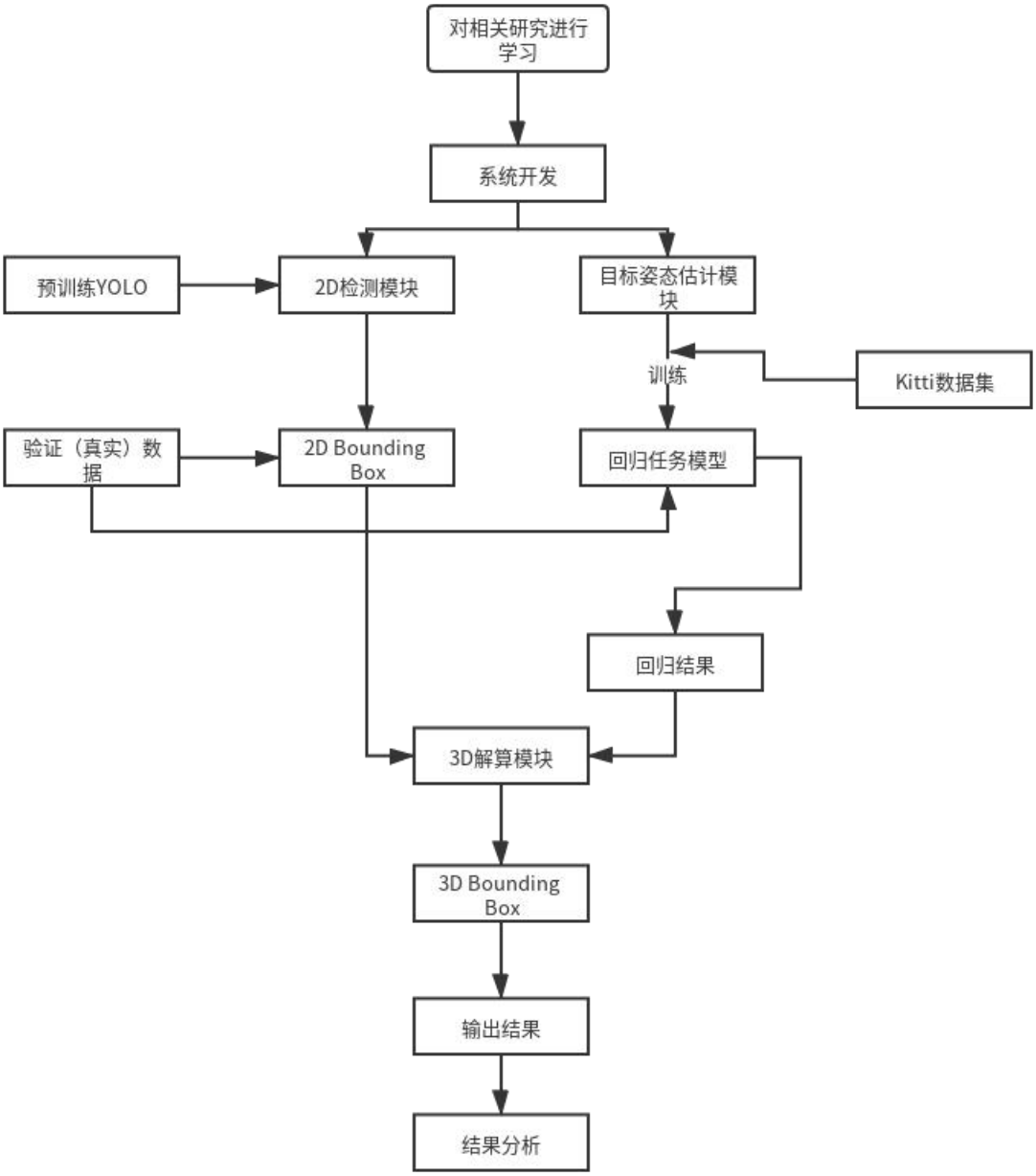
1. 具有强大的 GPU 加速的张量计算（如 NumPy）。
2. 包含自动求导系统的深度神经网络。

其特点如下：

- PyTorch 是相当简洁且高效快速的框架
- 设计追求最少的封装
- 设计符合人类思维，它让用户尽可能地专注于实现自己的想法
- 与 google 的 Tensorflow 类似，FAIR 的支持足以确保 PyTorch 获得持续的开发更新
- PyTorch 作者亲自维护的论坛 供用户交流和求教问题
- 入门简单

1.4 系统设计

根据上述对于本项目系统的分析，结合本组成员的研究环境条件，按照参考文献的指导，本项目将按照以下流程进行开发与测试：


对相关研究进行学习
系统开发
预训练YOLO
2D检测模块
目标姿态估计模块
训练
Kitti数据集
验证（真实）数据
2D Bounding Box
回归任务模型
回归结果
3D解算模块
3D Bounding Box
输出结果
结果分析

第二章 相关算法和技术理论

2.1 常用坐标系

世界坐标系(world coordinate system)：用户定义的三维世界的坐标系，为了描述目标物在真实世界里的位置而被引入。单位为 m。

相机坐标系(camera coordinate system)：在相机上建立的坐标系，为了从相机的角度描述物体位置而定义，作为沟通世界坐标系和图像/像素坐标系的中间一环。单位为 m。

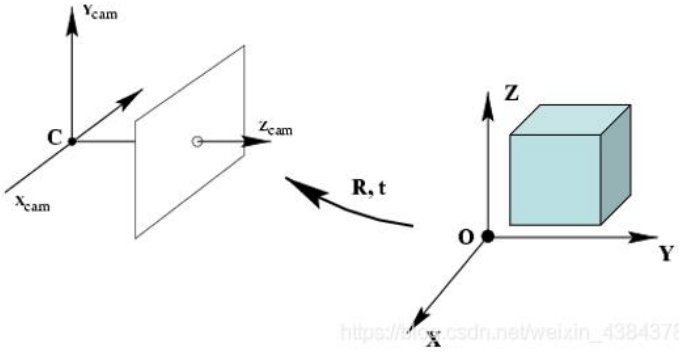
图像坐标系(image coordinate system)：为了描述成像过程中物体从相机坐标系到图像坐标系的投影透射关系而引入，方便进一步得到像素坐标系下的坐标。 单位为 m。

2.2 相机标定

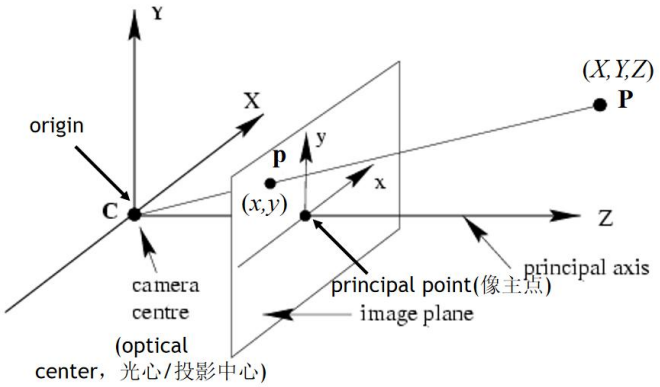
摄像机标定(Camera calibration)简单来说是从世界坐标系转换为相机坐标系，再由相机坐标系转换为图像坐标系的过程，也就是求最终的投影矩阵 P 的过程。

从世界坐标系到相机坐标系：

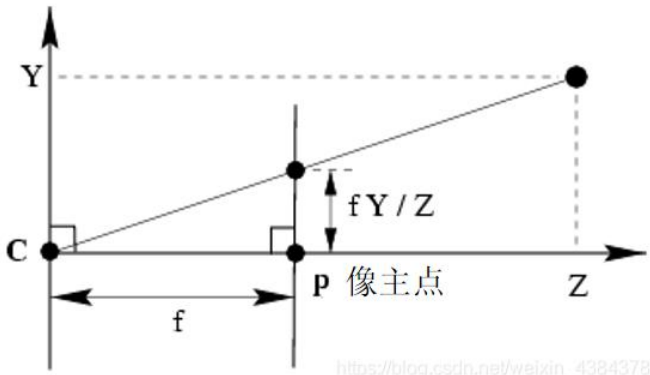
这一步是三维点到三维点的转换，包括 R,t（相机外参）等参数；



相机坐标系转换为图像坐标系：



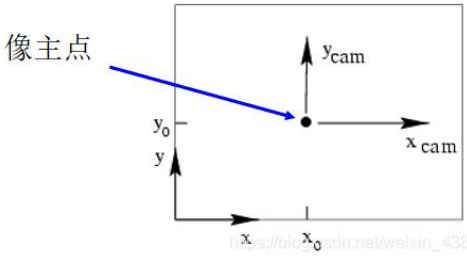
这一步是三维点到二维点的转换，包括 K（相机内参）等参数；



根据上述的关系图可以推导出下面的变换公式：

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

像主点的偏移：



可以推出：

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

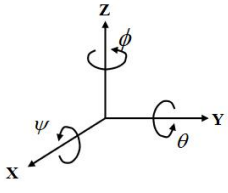
内参矩阵 K：

$$\begin{bmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

外参矩阵[R | t]：

表示三个方向的偏转：

$$\mathbf{x} \sim \mathbf{K}\underbrace{[\mathbf{R}|\mathbf{t}]}_{\text{外参矩阵}}\mathbf{X}$$



$$R = Rot(z,\varphi)Rot(y,\theta)Rot(x,\psi)$$

$$= \begin{bmatrix} C\varphi & -S\varphi & 0 \\ S\varphi & C\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\theta & 0 & S\theta \\ 0 & 1 & 0 \\ -S\theta & 0 & C\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\psi & -S\psi \\ 0 & S\psi & C\psi \end{bmatrix}$$

$$= \begin{bmatrix} C\varphi C\theta & -S\varphi S\psi + C\varphi S\theta S\psi & C\varphi S\theta C\psi + S\varphi S\psi \\ S\varphi C\theta & C\varphi S\theta S\psi + C\varphi S\psi & -C\varphi S\psi + S\varphi S\theta C\psi \\ -S\theta & C\theta S\psi & C\theta C\psi \end{bmatrix}$$

投影矩阵 P

（在这里可以认为旋转矩阵 R 为单位矩阵 I，平移矩阵 t 都为 0）：

$$P = K[I|0]$$

根据上述变换最终可以得到一个投影矩阵 P 的公式为：

$$P = K[R|t]$$

2.3 透射投影（perspective projection）

针孔成像模型又称为线性计算机模型。空间任何一点 P 在图像中的成像位置可以用针孔成像模型近似表示，即任何点 P 在图像中的投影位置 p，为光心 O 与 P 点的连线 OP 与图像平面的交点。这种关系也被称为中心射影（也就是我们平时说的透射投影 perspective projection）。参考上图模型，比例关系可以确定如下：

$$\begin{cases} X = \frac{fx}{z} \\ Y = \frac{fy}{z} \end{cases}$$

（X,Y）为 p 点的图像坐标；（x,y,z）为空间点 P 在摄像机坐标系下的坐标，f 为 xy 平面与图像平面的距离（大多数的时候我们把 f 称为摄像机的焦距），上面的关系我们仍能够用一个矩阵表示：

$$s\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

其中，s 是一个比例因子，P 就是我们最为关心的透视投影矩阵。

那么，通过上面的总结，我们就能够很轻易地得到世界坐标系表示下的 P 点与其在图像坐标系下的 p 点的坐标变换关系。具体如下所示：

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{dX} & 0 & u_0 \\ 0 & \frac{1}{dY} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \alpha_x & 0 & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \boldsymbol{M}_1 \boldsymbol{M}_2 \boldsymbol{X}_w = \boldsymbol{M}\boldsymbol{X}_w$$

ax=f/dX 为 u 轴上的尺度因子（平时我们叫 u 轴上归一化焦距）；ay=f/dY 为 v 轴上的尺度因子（平时我们称 v 轴上归一化焦距）。M 是投影矩阵；M1 由 ax,ay,u0,v0 四个参数决定（这些参数可是仅仅跟摄像机内部参数有关，所以我们叫摄像机内部参数）。M2 由摄像机相对于世界坐标系的方位决定，称为摄像机外部参数。确定某一摄像机的内外参数，称为摄像机的标定。

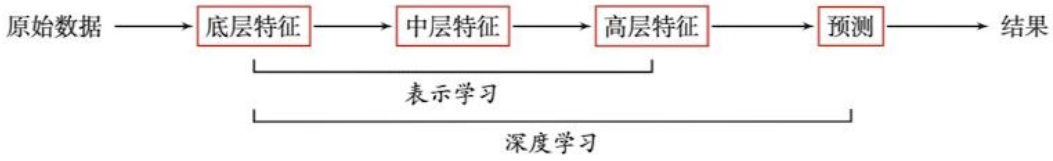
通过上式，如果我们得到了摄像机的内外参数，就拿到了投影矩阵 M，这是对于任何空间点 P，如果我们知道他在世界坐标系下的坐标 Cw=(Xw,Yw,Zw),就可以准确定位他在图像中的投影位置。但是，反过来这是不成立的，这就是摄像机成像损失了成像深度的理论来源

2.4 深度学习

为了学习一种好的表示, 需要构建具有一定“深度”的模型, 并通过学习算法来让模型自动学习出好的特征表示(从底层特征, 到中层特征, 再到高层特征),从而最终提升预测模型的准确率. 所谓“深度”是指原始数据进行非线性特征转换的次数. 如果把一个表示学习系统看作一个有向图结构, 深度也可以看作从输入节点到输出节点所经过的最长路径的长度

这样我们就需要一种学习方法可以从数据中学习一个“深度模型”,这就是深度学习(Deep Learning , DL). 深度学习是机器学习的一个子问题, 其主要目的是从数据中自动学习到有效的特征表示

下图给出了深度学习的数据处理流程. 通过多层的特征转换, 把原始数据变成更高层次、 更抽象的表示. 这些学习到的表示可以替代人工设计的特征, 从而避免“特征工程”



深度学习是将原始的数据特征通过多步的特征转换得到一种特征表示, 并进一步输入到预测函数得到最终结果. 和“浅层学习”不同, 深度学习需要解决的关键问题是贡献度分配问题(Credit Assignment Problem , CAP) [Minsky,1961] , 即一个系统中不同的组件(component 或其参数对最终系统输出结果的贡献或影响. 以下围棋为例, 每当下完一盘棋, 最后的结果要么赢要么输. 我们会思考哪几步棋导致了最后的胜利, 或者又是哪几步棋导致了最后的败局. 如何判断每一步棋的贡献就是贡献度分配问题, 这是一个非常困难的问题. 从某种意义上讲, 深度学习可以看作一种强化学习(Reinforcement Learning , RL), 每个内部组件并不能直接得到监督信息, 需要通过整个模型的最终监督信息(奖励)得到,并且有一定的延时性.

目前, 深度学习采用的模型主要是神经网络模型, 其主要原因是神经网络模型可以使用误差反向传播算法, 从而可以比较好地解决贡献度分配问题. 只要是超过一层的神经网络都会存在贡献度分配问题, 因此可以将超过一层的神经网络都看作深度学习模型. 随着深度学习的快速发展, 模型深度也从早期的 5 ~ 10 层增加到目前的数百层. 随着模型深度的不断增加, 其特征表示的能力也越来越强,从而使后续的预测更加容易.

2.5 人工神经网络

人工神经网络是为模拟人脑神经网络而设计的一种计算模型, 它从结构、 实现机理和功能上模拟人脑神经网络. 人工神经网络与生物神经元类似, 由多个节点(人工神经元)互相连接而成, 可以用来对数据之间的复杂关系进行建模. 不同节点之间的连

接被赋予了不同的权重, 每个权重代表了一个节点对另一个节点的影响大小. 每个节点代表一种特定函数, 来自其他节点的信息经过其相应的权重综合计算, 输入到一个激活函数中并得到一个新的活性值(兴奋或抑制).

从系统观点看, 人工神经元网络是由大量神经元通过极其丰富和完善的连接而构成的自适应非线性动态系统

虽然我们可以比较容易地构造一个人工神经网络, 但是如何让人工神经网络具有学习能力并不是一件容易的事情. 早期的神经网络模型并不具备学习能力. 首个可学习的人工神经网络是赫布网络, 采用一种基于赫布规则的无监督学习方法. 感知器是最早的具有机器学习思想的神经网络, 但其学习方法无法扩展到多层的神经网络上. 直到 1980 年左右, 反向传播算法才有效地解决了多层神经网络的学习问题,并成为最为流行的神经网络学习算法.

人工神经网络诞生之初并不是用来解决机器学习问题. 由于人工神经网络可以用作一个通用的函数逼近器(一个两层的神经网络可以逼近任意的函数),因此我们可以将人工神经网络看作一个可学习的函数, 并将其应用到机器学习中.理论上,只要有足够的训练数据和神经元数量,人工神经网络就可以学到很多复杂的函数. 我们可以把一个人工神经网络塑造复杂函数的能力称为网络容量(Network Capacity), 这与可以被储存在网络中的信息的复杂度以及数量相关

人工神经元(Artificial Neuron), 简称神经元(Neuron), 是构成神经网络的基本单元, 其主要是模拟生物神经元的结构和特性, 接收一组输入信号并产生输出.

假设一个神经元接收 D 个输入 x 1 , x 2 , ⋯ , x D ,令向量 x = [x 1 ; x 2 ; ⋯ ; x D] 来表示这组输入, 并用净输入(Net Input) z ∈ R 表示一个神经元所获得的输入信号 x 的加权和

$$
\begin{aligned}
z &= \sum_{d=1}^{D} w_d x_d + b \\
&= \boldsymbol{w}^{\top} \boldsymbol{x} + b,
\end{aligned}
$$

净输入 z 在经过一个非线性函数 f(·) 后, 得到神经元的活性值(Activation)a

$$a = f(z)$$

该非线性函数称为激活函数

2.6 ReLU 函数

ReLU(Rectified Linear Unit，修正线性单元)[Nair et al., 2010]，也叫 Rectifier 函数 [Glorot et al., 2011]，是目前深度神经网络中经常使用的激活函数. ReLU 实际上是一个斜坡(ramp)函数，定义为

$$\begin{aligned}\mathrm{ReLU}(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \\ &= \max(0, x).\end{aligned}$$

2.7 反向传播和梯度下降

为了充分利用凸优化中一些高效、成熟的优化方法，比如共轭梯度、拟牛顿法等，很多机器学习方法都倾向于选择合适的模型和损失函数，以构造一个凸函 数作为优化目标.但也有很多模型(比如神经网络)的优化目标是非凸的，只能退而求其次找到局部最优解.

在机器学习中，最简单、常用的优化算法就是梯度下降法，即首先初始化参 数 ℒ0，然后按下面的迭代公式来计算训练集 𝒟 上风险函数的最小值:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha \frac{\partial \mathcal{R}_{\mathcal{D}}(\theta)}{\partial \theta} \\ &= \theta_t - \alpha \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}\left(y^{(n)}, f(\boldsymbol{x}^{(n)}; \theta)\right)}{\partial \theta},\end{aligned}$$

其中 θ_t 为第 t 次迭代时的参数值，α 为搜索步长.在机器学习中，α 一般称为学习率(Learning Rate).

假设采用随机梯度下降进行神经网络参数学习，给定一个样本 $(\boldsymbol{x}, \boldsymbol{y})$，将其输入到神经网络模型中，得到网络输出为 $\hat{\boldsymbol{y}}$.假设损失函数为 $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$，要进行参数学习就需要计算损失函数关于每个参数的导数.

第 l 层的误差项可以通过第 $l+1$ 层的误差项计算得到，这就是误差的反向传播(BackPropagation，BP).反向传播算法的含义是: 第 l 层的一个神经元的误差项(或敏感性)是所有与该神经元相连的第 $l+1$ 层 的神经元的误差项的权重和.然后，再乘上该神经元激活函数的梯度.

在计算出每一层的误差项之后，我们就可以得到每一层参数的梯度.因此， 使用误差反向传播算法的前馈神经网络训练过程可以分为以下三步:

1. 前馈计算每一层的净输入 $\boldsymbol{z}$

2. 反向传播计算每一层的误差项 δ (l)

3. 计算每一层参数的偏导数，并更新参数.和激活值 $\boldsymbol{a}$ ，直到最后一层

输入: 训练集 $\mathcal{D}=\{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$, 验证集 $\mathcal{V}$, 学习率 α, 正则化系数 λ, 网络层数 L, 神经元数量 $M_l, 1 \le l \le L$.

随机初始化 $\boldsymbol{W}, \boldsymbol{b}$;
repeat
  对训练集 $\mathcal{D}$ 中的样本随机重排序;
  for $n = 1 \cdots N$ **do**
    从训练集 $\mathcal{D}$ 中选取样本 $(\boldsymbol{x}^{(n)}, y^{(n)})$;
    前馈计算每一层的净输入 $\boldsymbol{z}^{(l)}$ 和激活值 $\boldsymbol{a}^{(l)}$, 直到最后一层;
    反向传播计算每一层的误差 $\delta^{(l)}$; // 公式(4.63)
    // 计算每一层参数的导数
    $\forall l,\ \frac{\partial \mathcal{L}(y^{(n)}, \hat{y}^{(n)})}{\partial \boldsymbol{W}^{(l)}} = \delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\top}$; // 公式(4.68)
    $\forall l,\ \frac{\partial \mathcal{L}(y^{(n)}, \hat{y}^{(n)})}{\partial \boldsymbol{b}^{(l)}} = \delta^{(l)}$; // 公式(4.69)
    // 更新参数
    $\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \alpha(\delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\top} + \lambda \boldsymbol{W}^{(l)})$;
    $\boldsymbol{b}^{(l)} \leftarrow \boldsymbol{b}^{(l)} - \alpha \delta^{(l)}$;
  end
until 神经网络模型在验证集 $\mathcal{V}$ 上的错误率不再下降;
输出: $\boldsymbol{W}, \boldsymbol{b}$

随机梯度下降（SGD）算法

随机梯度下降（SGD）也称为增量梯度下降，是一种迭代方法，用于优化可微分目标函数。该方法通过在小批量数据上计算损失函数的梯度而迭代地更新权重与偏置项。SGD 在高度非凸的损失表面上远远超越了朴素梯度下降法，这种简单的爬山法技术已经主导了现代的非凸优化。

SGD 算法是从样本中随机抽出一组，训练后按梯度更新一次，然后再抽取一组，再更新一次，在样本量及其大的情况下，可能不用训练完所有的样本就可以获得一个损失值在可接受范围之内的模型了。

2.8 小批量梯度下降

在训练深度神经网络时，训练数据的规模通常都比较大.如果在梯度下降 时，每次迭代都要计算整个训练数据上的梯度，这就需要比较多的计算资源.另 外大规模训练集中的数据通常会非常冗余，也没有必要在整个训练集上计算梯 度.因此，在训练深度神经网络时，经常使用小批量梯度下降法(Mini-Batch Gradient Descent).

影响小批量梯度下降法的主要因素有:1)批量大 小K 、2)学习率α 、3)梯度估计.为了更有效地训练深度神经网络，在标准的小批 量梯度下降法的基础上，也经常使用一些改进方法以加快优化速度，比如如何选 择批量大小、如何调整学习率以及如何修正梯度估计.我们分别从这三个方面来 介绍在神经网络优化中常用的算法.这些改进的优化算法也同样可以应用在批量或随机梯度下降法上.

2.9 卷积神经网络

在全连接前馈神经网络中，如果第 l 层有 Ml 个神经元，第 l − 1 层有 Ml −1 个 神经元，连接边有 Ml × Ml −1 个，也就是权重矩阵有 Ml × Ml −1 个参数.当 Ml 和 Ml −1 都很大时，权重矩阵的参数非常多，训练的效率会非常低.如果采用卷积来代替全连接，第 l 层的净输入 $\boldsymbol{z}$ (l) 为第 l − 1 层活性值 $\boldsymbol{a}$ (l −1) 和卷积核 $\boldsymbol{w}$ (l) ∈ RK 的卷积，即

$$\boldsymbol{z}\ (l\ ) = \boldsymbol{w}\ (l\ ) \otimes \boldsymbol{a}\ (l\ -1) + b\ (l\ ),$$

其中卷积核 $\boldsymbol{w}$ （l ）∈ RK 为可学习的权重向量，b （l ）∈ R 为可学习的偏置.

- 局部连接

  在卷积层(假设是第 l 层)中的每一个神经元都只和下一层(第 l − 1 层)中某个局部窗口内的神经元相连，构成一个局部连接网络.如图 5.5b 所示，卷 积层和下一层之间的连接数大大减少，由原来的 M l × Ml −1 个连接变为 Ml × K 个连接，K 为卷积核大小.

- 权重共享

  作为参数的卷积核 $\boldsymbol{w}$ （l ）对于第 l 层的所有的 神经元都是相同的.如图 5.5b 中，所有的同颜色连接上的权重是相同的.权重共享可以理解为一个卷积核只捕捉输入数据中的一种特定的局部特征.因此，如果 要提取多种特征就需要使用多个不同的卷积核.

2.10 Softmax 回归

Softmax 回归(Softmax Regression)，也称为多项(Multinomial)或多类 (Multi−Class)的 Logistic 回归，是 Logistic 回归在多分类问题上的推广.

对于多类问题，类别标签 y ∈ {1, 2, ⋯ , C } 可以有 C 个取值.给定一个样本 $\boldsymbol{x}$ ， Softmax 回归预测的属于类别 c 的条件概率为

$$p(y=c|\boldsymbol{x}) = \text{softmax}(\boldsymbol{w}_c^{\top}\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_c^{\top}\boldsymbol{x})}{\sum_{c'=1}^{C}\exp(\boldsymbol{w}_{c'}^{\top}\boldsymbol{x})},$$

其中 $\boldsymbol{w}\,c$ 是第 c 类的权重向量.

Softmax 回归的决策函数可以表示为

$$\hat{y} = \arg\max_{c=1}^{C} p(y = c|\boldsymbol{x})$$
$$= \arg\max_{c=1}^{C} \boldsymbol{w}_c^{\mathrm{T}} \boldsymbol{x}.$$

2.11 目标检测及 YOLO 模型

2D 目标检测

目标检测（Object Detection）的任务是找出图像中所有感兴趣的目标（物体），确定它们的类别和位置，是计算机视觉领域的核心问题之一。由于各类物体有不同的外观、形状和姿态，加上成像时光照、遮挡等因素的干扰，目标检测一直是计算机视觉领域最具有挑战性的问题。

计算机视觉中关于图像识别有四大类任务：

1. 分类-Classification：解决“是什么？”的问题，即给定一张图片或一段视频判断里面包含什么类别的目标。
2. 定位-Location：解决“在哪里？”的问题，即定位出这个目标的的位置。
3. 检测-Detection：解决“在哪里？是什么？”的问题，即定位出这个目标的位置并且知道目标物是什么。
4. 分割-Segmentation：分为实例的分割（Instance-level）和场景分割（Scene-level），解决“每一个像素属于哪个目标物或场景”的问题。

2D Object Detection 的研究已经非常成熟了，代表作品有 RPN 系列的 FasterRCNN 和 MaskRCNN，One Shot 系列的 YOLOv1-YOLOv3。

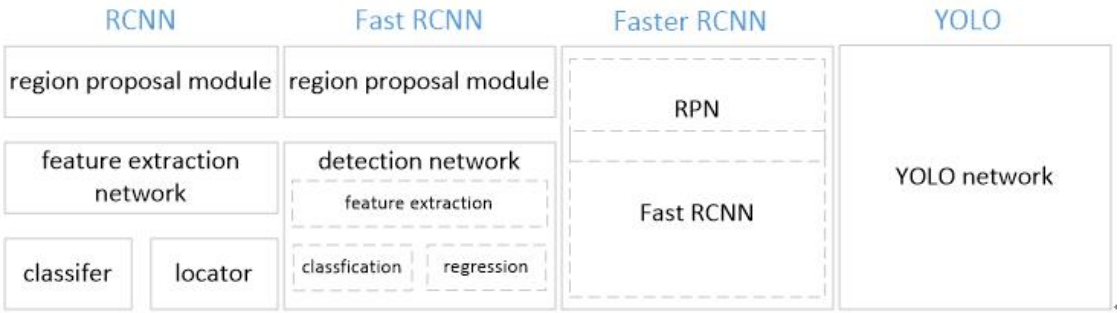
YOLO

该模型产生于 CVPR 2016 目标检测论文 YOLO: Unified, Real-Time Object Detection

YOLO 将物体检测作为回归问题求解。基于一个单独的 end-to-end 网络，完成从原始图像的输入到物体位置和类别的输出。从网络设计上，YOLO 与 rcnn、fast rcnn 及 faster rcnn 的区别如下：

1. YOLO 训练和检测均是在一个单独网络中进行。YOLO 没有显示地求取 region proposal 的过程。而 rcnn/fast rcnn 采用分离的模块（独立于网络之外的 selective search 方法）求取候选框（可能会包含物体的矩形区域），训练过程因此也是分成多个模块进行。Faster rcnn 使用 RPN（region proposal network）卷积网络替代 rcnn/fast rcnn 的 selective search 模块，将 RPN 集成到 fast rcnn 检测网络中，得到一个统一的检测网络。尽管 RPN 与 fast rcnn 共享卷积层，但是在模型训练过程中，需要反复训练 RPN 网络和 fast rcnn 网络（注意这两个网络核心卷积层是参数共享的）。

2. YOLO 将物体检测作为一个回归问题进行求解，输入图像经过一次 inference，便能得到图像中所有物体的位置和其所属类别及相应的置信概率。而 rcnn/fast rcnn/faster rcnn 将检测结果分为两部分求解：物体类别（分类问题），物体位置即 bounding box（回归问题）。



2.12 3D 目标检测

在 2D Object Detection 的基础上又提出了新的要求 3D Object Detection。问题的具体描述检测环境中的三维物体，并给出物体的 Bounding Box。相比于 2D，3D 的 Bounding Box 的表示除了多了一个维度的位置和尺寸，还多了三个角度。可以想象，一架飞机的 Bounding Box 的尺寸的是固定的，飞机的姿态除了位置之外，还有俯仰角、偏航角和翻滚角三个角度。

目前对于 3D Object Detection 有迫切需求的产业是自动驾驶产业，因为要想安全的自动驾驶，需要周围障碍物的三维位姿，在图片中的二维位姿不带深度信息，没有办法有效避免碰撞。所以 3D Object Detection 的数据集大多也是自动驾驶数据集，类别也主要是车辆和行人等，比较常用的有 KITTI 和 kaist。由于自动驾驶针对车辆，所以障碍物的高度的检测对于安全行驶并没有十分重要，而障碍物都在陆地上，所以也不存在俯仰角和翻滚角两个角度。所以有些 3D Object Detection 方法将这三值忽略了。

目前 3D 目标检测正处于高速发展时期，目前主要是综合利用单目相机、双目相机、多线激光雷达来进行 3D 目标检测，从目前成本上讲，激光雷达>双目相机>单目相机，从目前的准确率上讲，激光雷达>双目相机>单目相机。但是随着激光雷达的不断产业化发展，成本在不断降低，目前也出现一些使用单目相机加线数较少的激光雷达进行综合使用的技术方案。进行 3D 目标检测从使用的数据进行划分主要可以分为以下几类:

1. 激光

   Voxelnet，Voxelnet 把激光点云在空间中均匀划分为不同的 voxel，再把不同 voxel 中的点云通过提出的 VFE（Voxel Feature Encoding）层转换为一个统一的特征表达，最后使用 RPN（Region Proposal Network）对物体进行分类和位置回归

2. 单目相机

   以开源的 Apollo 为例，Apollo 中使用的 YOLO 3D，在 Apollo 中通过一个多任务网络来进行车道线和场景中目标物体检测。其中的 Encoder 模块是 Yolo 的 Darknet，在原始 Darknet 基础上加入了更深的卷积层同时添加反卷积层，捕捉更丰富的图像上下文信息。高分辨多通道特征图，捕捉图像细节，深层低分辨率多通道特征图，编码更多图像上下文信息。和 FPN（Feature Paramid Network）类似的飞线连接，更好的融合了图像的细节和整体信息。Decoder 分为两个部分，一部分是语义分割，用于车道线检测，另一部分为物体检测，物体检测部分基于 YOLO，同时还会输出物体的方向等 3D 信息。

3. 激光+单目相机

AVOD，AVOD 输入 RGB 图像及 BEV（Bird Eye View），利用 FPN 网络得到二者全分辨率的特征图，再通过 Crop 和 Resize 提取两个特征图对应的区域进行融合，挑选出 3D proposal 来进行 3D 物体检测

2.12 VGG19

VGG 是 Oxford 的 Visual Geometry Group 的组提出的。该网络是在 ILSVRC 2014 上的相关工作，主要工作是证明了增加网络的深度能够在一定程度上影响网络最终的性能。VGG 有两种结构，分别是 VGG16 和 VGG19，两者并没有本质上的区别，只是网络深度不一样。

VGG16 相比 AlexNet 的一个改进是采用连续的几个 3x3 的卷积核代替 AlexNet 中的较大卷积核（11x11，7x7，5x5）。对于给定的感受野（与输出有关的输入图片的局部大小），采用堆积的小卷积核是优于采用大的卷积核，因为多层非线性层可以增加网络深度来保证学习更复杂的模式，而且代价还比较小（参数更少）。

简单来说，在 VGG 中，使用了 3 个 3x3 卷积核来代替 7x7 卷积核，使用了 2 个 3x3 卷积核来代替 5*5 卷积核，这样做的主要目的是在保证具有相同感知野的条件下，提升了网络的深度，在一定程度上提升了神经网络的效果。

比如，3 个步长为 1 的 3x3 卷积核的一层层叠加作用可看成一个大小为 7 的感受野（其实就表示 3 个 3x3 连续卷积相当于一个 7x7 卷积），其参数总量为 3x(9xC^2)，如果直接使用 7x7 卷积核，其参数总量为 49xC^2，这里 C 指的是输入和输出的通道数。很明显，27xC2 小于 49xC2，即减少了参数；而且 3x3 卷积核有利于更好地保持图像性质。

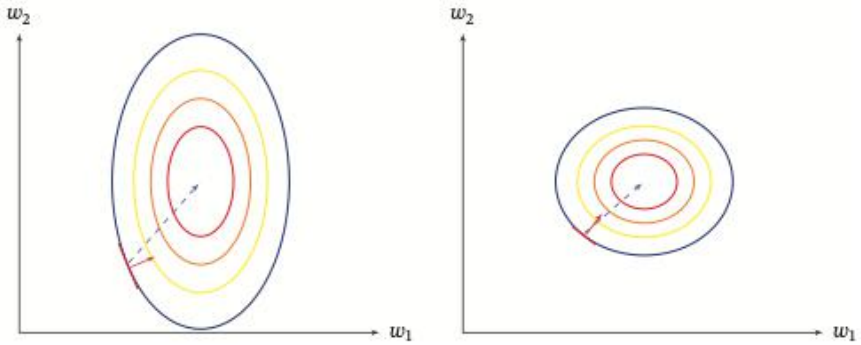
2.13 数据预处理

一般而言，样本特征由于来源以及度量单位不同，它们的尺度(Scale)(即 取值范围)往往差异很大.以描述长度的特征为例，当用“米”作单位时令其值为 x ，那么当用“厘米”作单位时其值为 $100x$.不同机器学习模型对数据特征尺度的 敏感程度不一样.如果一个机器学习算法在缩放全部或部分特征后不影响它的 学习和预测，我们就称该算法具有尺度不变性(Scale Invariance).比如线性分 类器是尺度不变的，而最近邻分类器就是尺度敏感的.当我们计算不同样本之间 的欧氏距离时，尺度大的特征会起到主导作用.因此，对于尺度敏感的模型，必须 先对样本进行预处理，

将各个维度的特征转换到相同的取值区间，并且消除不同 特征之间的相关性，才能获得比较理想的结果.

从理论上，神经网络应该具有尺度不变性，可以通过参数的调整来适应不 同特征的尺度.但尺度不同的输入特征会增加训练难度.假设一个只有一层的 网络 $y = \tanh(w_1 x_1 + w_2 x_2 + b)$，其中 $x_1 \in [0, 10]$，$x_2 \in [0, 1]$.之前我们提 到 tanh 函数的导数在区间 [−2, 2] 上是敏感的，其余的导数接近于 0.因此，如果 $w_1 x_1 + w_2 x_2 + b$ 过大或过小，都会导致梯度过小，难以训练.为了提高训练效率， 我们需要使 $w_1 x_1 + w_2 x_2 + b$ 在 [−2, 2] 区间，因此需要将 w_1 设得小一点，比如在 [−0.1, 0.1] 之间.可以想象，如果数据维数很多时，我们很难这样精心去选择每一 个参数.因此，如果每一个特征的尺度相似，比如 [0, 1] 或者 [−1, 1]，我们就不太 需要区别对待每一个参数，从而减少人工干预.

除了参数初始化比较困难之外， 不同输入特征的尺度差异比较大时， 梯度下降法的效率也会受到影响.图 7.9 给出了数据归一化对梯度的影响.其中， 图 7.9a 为未归一化数据的等高线图.尺度不同会造成在大多数位置上的梯度方向 并不是最优的搜索方向.当使用梯度下降法寻求最优解时，会导致需要很多次迭 代才能收敛.如果我们把数据归一化为相同尺度，如图 7.9b 所示，大部分位置的 梯度方向近似于最优搜索方向.这样，在梯度下降求解时，每一步梯度的方向都 基本指向最小值，训练效率会大大提高.



(a) 未归一化数据的梯度　　(b) 归一化数据的梯度

- 归一化(Normalization)方法泛指把数据特征转换为相同尺度的方法，比 如把数据特征映射到 [0, 1] 或 [−1, 1] 区间内，或者映射为服从均值为 0、方差为 1 的标准正态分布.归一化的方法有很多种，比如之前我们介绍的 Sigmoid 型函数 等都可以将不同尺度的特征挤压到一个比较受限的区间.这里，我们介绍几种在 神经网络中经常使用的归一化方法.

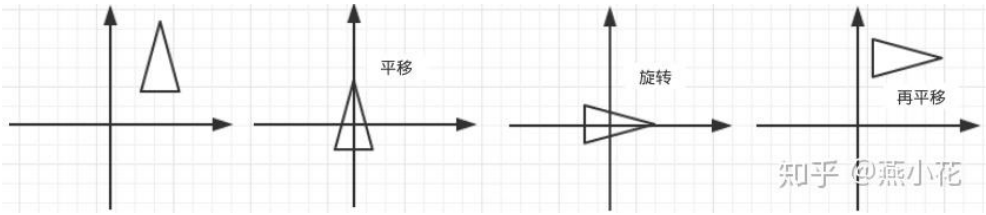
- 标准化(Standardization)也叫 Z 值归一化(Z-Score Normalization)，来源于统计上的标准分数.将每一个维特征都调整为均值为 0，方差为 1.

2.14 数据增广

数据增广是深度学习中常用的技巧之一，主要用于增加训练数据集，让数据集尽可能的多样化，使得训练的模型具有更强的泛化能力．现有的各大深度学习框架都已经自带了数据增广，但是平时在用的使用只是直接调用了对应的接口函数，而没有进行详细的分析．在实际应用中，并非所有的增广方式都适用当前的训练数据，你需要根据自己的数据集特征来确定应该使用哪几种数据增广方式

目前数据增广主要包括：水平/垂直翻转，旋转，缩放，裁剪，剪切，平移，对比度，色彩抖动，噪声等

所有的数据增广在操作的时候默认是以图像中心点进行的．从数学角度来看，任何操作都可以分成以下几个步骤：1). 首先将旋转点移动到原点处 ； 2). 执行如 2 所描述的绕原点的旋转；3). 再将旋转点移回到原来的位置



平移前和平移后的坐标关系为：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

图像平移

平移是指所有的像素在 x 和 y 方向各平移和，平移变换对应的数学矩阵为

$$H = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

图像翻转(图像镜像)

图像翻转包括水平翻转和垂直翻转．水平翻转的变换矩阵为：

$$H = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

垂直翻转的变换矩阵为：

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

图像旋转

图像旋转是指以某个点(默认为图像中心点)为中心进行任意角度的旋转，其变换矩阵为：

$$H = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

图像缩放

图像缩放是指对当前图像进行任意尺度的缩放，其变换矩阵为：

$$H = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

图像裁剪

深度学习的裁剪的常用做法是将图片缩放到原图的 1.1 倍，然后在缩放后的图像上进行裁剪操作

组合变换

在深度学习中的数据增广一般会采用多种增广方式的组合，这里就会涉及到矩阵乘法运算，根据其运算的规则，可以知道**不同的组合顺序结果是不一样的**。

2.15 迁移学习和预训练

标准机器学习的前提假设是训练数据和测试数据的分布是相同的.如果不 满足这个假设，在训练集上学习到的模型在测试集上的表现会比较差.而在很多 实际场景中，经常碰到的问题是标注数据的成本十分高，无法为一个目标任务准 备足够多相同分布的训练数据.因此，如果有一个相关任务已经有了大量的训练 数据，虽然这些训练数据的分布和目标任务不同，但是由于训练数据的规模比较 大，我们假设可以从中学习某些可以泛化的知识，那么这些知识对目标任务会有 一定的帮助.如何将相关任务的训练数据中的可泛化知识迁移到目标任务上，就 是迁移学习(Transfer Learning)要解决的问题.

具体而言，假设一个机器学习任务 $\mathcal{T}$ 的样本空间为 $\mathcal{X} \times \mathcal{Y}$ ，其中 $\mathcal{X}$ 为输入空 间，$\mathcal{Y}$ 为输出空间，其概率密度函数为 $p(\boldsymbol{x}, y)$.为简单起见，这里设 $\mathcal{X}$ 为 D 维实数 空间的一个子集，$\mathcal{Y}$ 为一个离散的集合.

一个样本空间及其分布可以称为一个领域(Domain):$\mathcal{D} = (\mathcal{X}, \mathcal{Y}, p(\boldsymbol{x}, y))$. 给定两个领域，如果它们的输入空间、输出空间或概率分布中至少一个不同，那 么这两个领域就被认为是不同的.从统计学习的观点来看，一个机器学习任务 $\mathcal{T}$ 定义为在一个领域 $\mathcal{D}$ 上的条件概率 $p(y|\boldsymbol{x})$ 的建模问题.

迁移学习是指两个不同领域的知识迁移过程，利用源领域(Source Domain) $\mathcal{D}_S$ 中学到的知识来帮助目标领域(Target Domain)$\mathcal{D}_T$ 上的学习任务.源领域 的训练样本数量一般远大于目标领域.

2014 年 Bengio 等人在 NIPS 上发表论文 How transferable are features in deep neural networks，研究深度学习中各个 layer 特征的可迁移性（或者说通用性）文章中进行了如下图所示的实验，有四种模型

- Domain A 上的基本模型 BaseA
- Domain B 上的基本模型 BaseB
- Domain B 上前 n 层使用 BaseB 的参数初始化（后续有 frozen 和 fine–tuning 两种方式）
- Domain B 上前 n 层使用 BaseA 的参数初始化（后续有 frozen 和 fine–tuning 两种方式）

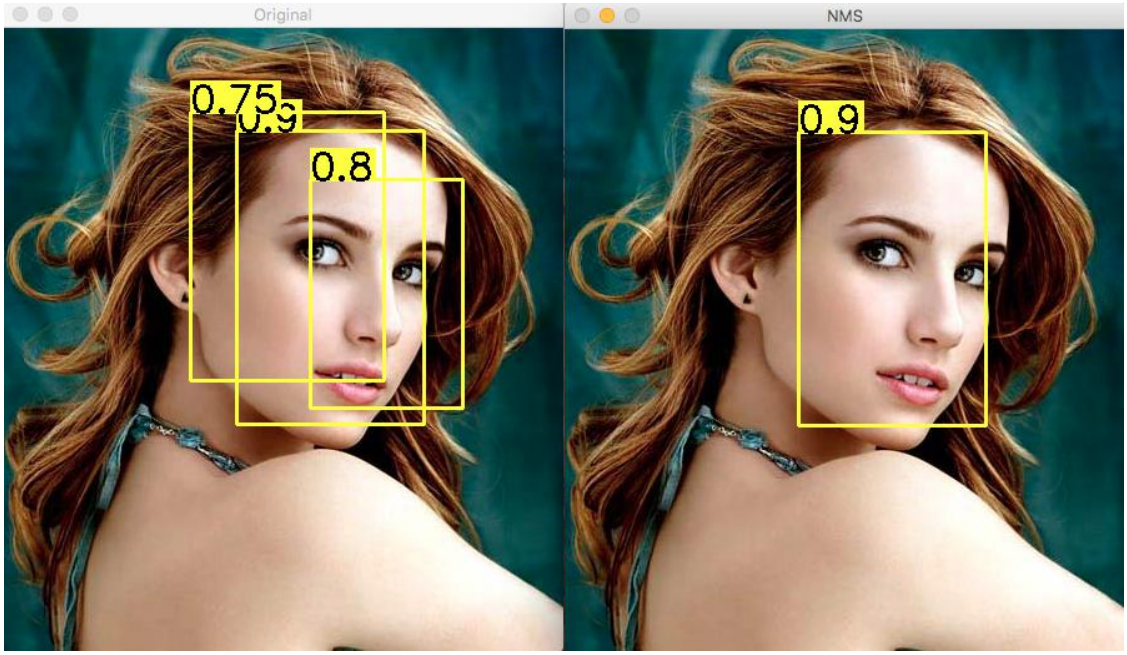
将深度学习应用在图像处理领域中时，会观察到第一层（first–layer）中提取的 features 基本上是类似于 Gabor 滤波器(Gabor filters)和色彩斑点(color blobs)之类的。
通常情况下第一层与具体的图像数据集关系不是特别大，而网络的最后一层则是与选定的数据集及其任务目标紧密相关的；文章中将第一层 feature 称之为一般(general)特征，最后一层称之为特定(specific)特征

- 特征迁移使得模型的泛化性能有所提升，即使目标数据集非常大的时候也是如此。
- 随着参数被固定的层数 n 的增长，两个相似度小的任务之间的 transferability gap 的增长速度比两个相似度大的两个任务之间的 transferability gap 增长更快 两个数据集越不相似特征迁移的效果就越差
- 即使从不是特别相似的任务中进行迁移也比使用随机 filters（或者说随机的参数）要好
- 使用迁移参数初始化网络能够提升泛化性能，即使目标 task 经过了大量的调整依然如此。

2.16 非极大值抑制（Non-Maximum Suppression）

非极大值抑制，简称为 NMS 算法，英文为 Non-Maximum Suppression。其思想是搜素局部最大值，抑制极大值。NMS 算法在不同应用中的具体实现不太一样，但思想是一样的。非极大值抑制，在计算机视觉任务中得到了广泛的应用，例如边缘检测、人脸检测、目标检测（DPM，YOLO，SSD，Faster R-CNN）等。

以目标检测为例：目标检测的过程中在同一目标的位置上会产生大量的候选框，这些候选框相互之间可能会有重叠，此时我们需要利用非极大值抑制找到最佳的目标边界框，消除冗余的边界框。Demo 如下图：



左图是人脸检测的候选框结果，每个边界框有一个置信度得分(confidence score)，如果不使用非极大值抑制，就会有多个候选框出现。右图是使用非极大值抑制之后的结果，符合我们人脸检测的预期结果。

2.17 最小二乘法

最小二乘法是一种在误差估计、不确定度、系统辨识及预测、预报等数据处理诸多学科领域得到广泛应用的数学工具

最小二乘法（又称最小平方法）是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小 。

最小二乘法还可用于曲线拟合，其他一些优化问题也可通过最小化能量或最大化熵用最小二乘法来表达

最小二乘法是解决曲线拟合问题最常用的方法。

第三章 实现过程分析

3.1 确定实现方法

在本项目开展的前期，本人依照项目设想、项目目的和项目要求开展了一系列的调研，阅读了若干相关领域的文献，了解了相关的实现方法，最终确定了项目的实现方法，该过程描述如下：

1. 根据论文 End-to-end real-time 3D Oriented Object Bounding Box Detection from LiDAR Point Cloud[3]理解了三维目标检测的问题定义及可能的解决方法，了解了常用于三维目标检测的标准数据集 Kitti，学习到了将 LiDAR 点云数据转换为鸟瞰图（BEV）后再做处理的这种常用的方法。但是由于直接训练三维数据需要的计算资源要求太高，无奈只能放弃。

2. 根据 2020 CVPR best paper: Unsupervised Learning of Probably Symmetric Deformable 3D Objects from Images in the Wild[4]中将二维图像转换为三维图像的方法，想到了可以利用单目相机获取的数据做三维检测，尽管该论文中解决的问题与本项目不同，但提供了另一种解决思路。

3. 发现百度开源平台 Apollo 中实现了单目相机数据完成三维检测的技术，学习相关资料后了解该技术需要对图像做图像分割，利用语义信息及其他预处理方法进行模型训练，由于其功能在开源平台中的集成度太高，实现过程中有大量步骤与本项目设想不符，遂放弃。

4. 学习论文 3D Bounding Box Estimation Using Deep Learning and Geometry[1]，发现其实现三维重建的方法不需要任何的预处理技术，而是利用几何学姿态限制和回归网络帮助实现由二维检测到三维检测的转换，由于其中涉及到的知识和《计算机视觉》理论课程中学习到的相似，具备实现本项目希望达到的目的，于是确定为以该论文的工作作为理论基础。

3.2 系统整体结构

整个三维目标识别系统由三个主要的模块组成：

1. **2D 图像目标检测网络**

   通过效果已经非常好的经典网络例如 Faster RCNN、SSD 及 YOLO 系列完成 2D 目标检测，得到 2D Bounding Box，用作后续的根据

二维特征的几何学限制解求 3D Bounding Box，原论文用的网络是 MS-CNN，本项目实践是利用的是预训练好的 YOLO 网络

2. **目标大小姿态估计网络**

整个项目需要训练的部分，通过这个部分需要得到除去 2D Bounding Box 之外的全部自由量，首先利用 VGG19 提取由 2D 图像检测模块画出的 Crop 部分的特征，然后用回归问题的模型训练一个模型可以提取该部分的回归参数。

3. **目标 3D 中心点解算模块**

利用上述两个模块得到的全部自由量，以及事先定义好的几何学姿态限制（若干方程组），由最小二乘法求得目标的 3D Bounding Box，并将其画出来

3.3 确定回归参数

按照原论文所描述的，要描述一个三维检测结果，需要根据透射投影原理，若我们知道二维图像上一个像素点的位置为 $x = (x, y, 1)$，我们需要将其映射到一个世界坐标系下的空间三维坐标点 $X = (X, Y, Z, 1)$，则我们由公式

$$x = K[R|T]X$$

可以知道在相机的内参数矩阵 K 已知的情况下，我们至少要知道 R 和 T 两个矩阵，才能确定 3D Bounding Box 的中心点的坐标，同时，要画出 3D bbox 我们还需要知道 bbox 的长宽高三个属性，为了简化问题，通常我们将问题规划为求解三个矩阵值如下：

$$D = [L, W, H]$$

$$T = [X, Y, Z]$$

$R = [\theta, \varphi, \phi]$（分别为仰角、偏航角和旋转角）

且通常我们认为对于 T 矩阵，我们平移仅在平面上完成，其 Z 值常为 0。且对于汽车行驶这一应用场景，通常仰角和偏航角也可以认为为 0。

由于 R 矩阵对我们识别的效果影响最大，因此我们需要选择它进行回归

而对于平移矩阵 T 和形状矩阵 D，我们选择 D 矩阵进行回归，同时我们回归的是 D 与其统计平均值的残差，这是因为对于相同类别的目标来说，它们的方差通常特别小，因此我们回归残差可以直接用 L2 损失，这样可以达到更好的效果。

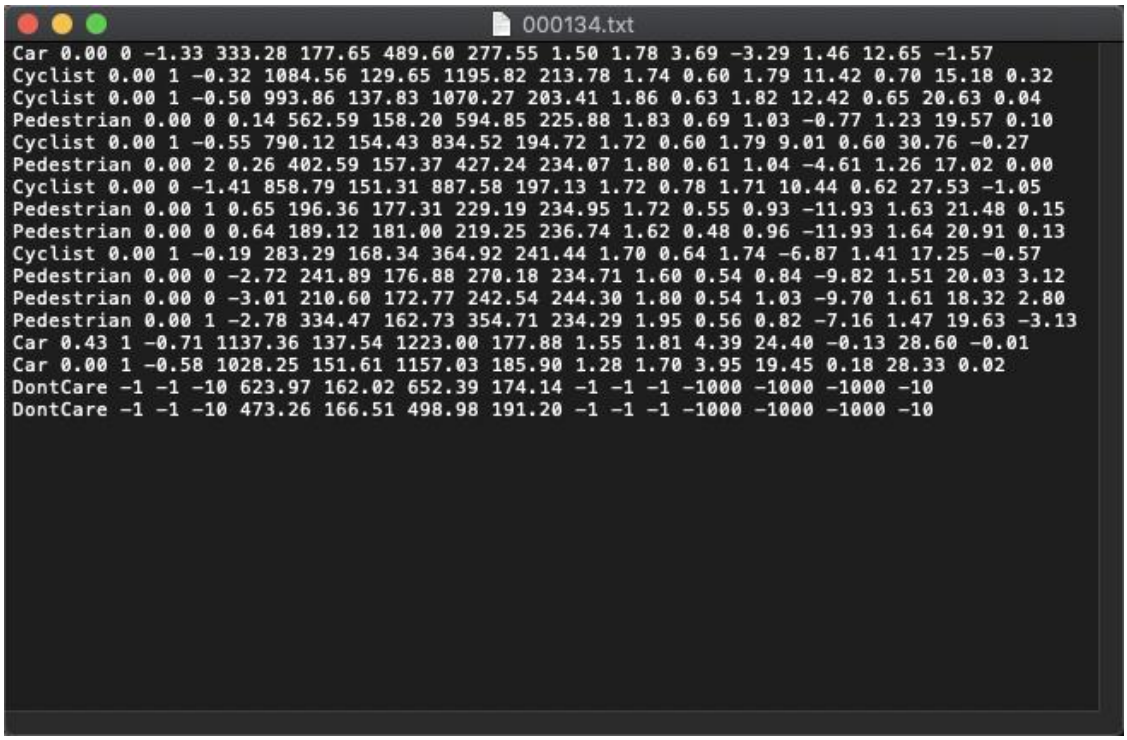
3.3 回归模型训练

3.3.1 Kitti *数据集*

KITTI 数据集由德国卡尔斯鲁厄理工学院和丰田美国技术研究院联合创办，是目前国际上最大的自动驾驶场景下的计算机视觉算法评测数据集。该数据集用于评测立体图像(stereo)，光流(optical flow)，视觉测距(visual odometry)，3D 物体检测(object detection)和 3D 跟踪(tracking)等计算机视觉技术在车载环境下的性能。KITTI 包含市区、乡村和高速公路等场景采集的真实图像数据，每张图像中最多达 15 辆车和 30 个行人，还有各种程度的遮挡与截断。整个数据集由 389 对立体图像和光流图，39.2 km 视觉测距序列以及超过 200k 3D 标注物体的图像组成[1]，以 10Hz 的频率采样及同步。总体上看，原始数据集被分类为'Road', 'City', 'Residential', 'Campus' 和 'Person'。对于 3D 物体检测，label 细分为 car, van, truck, pedestrian, pedestrian(sitting), cyclist, tram 以及 misc 组成。

例如，其 training set 中的第 134 号数据如下：

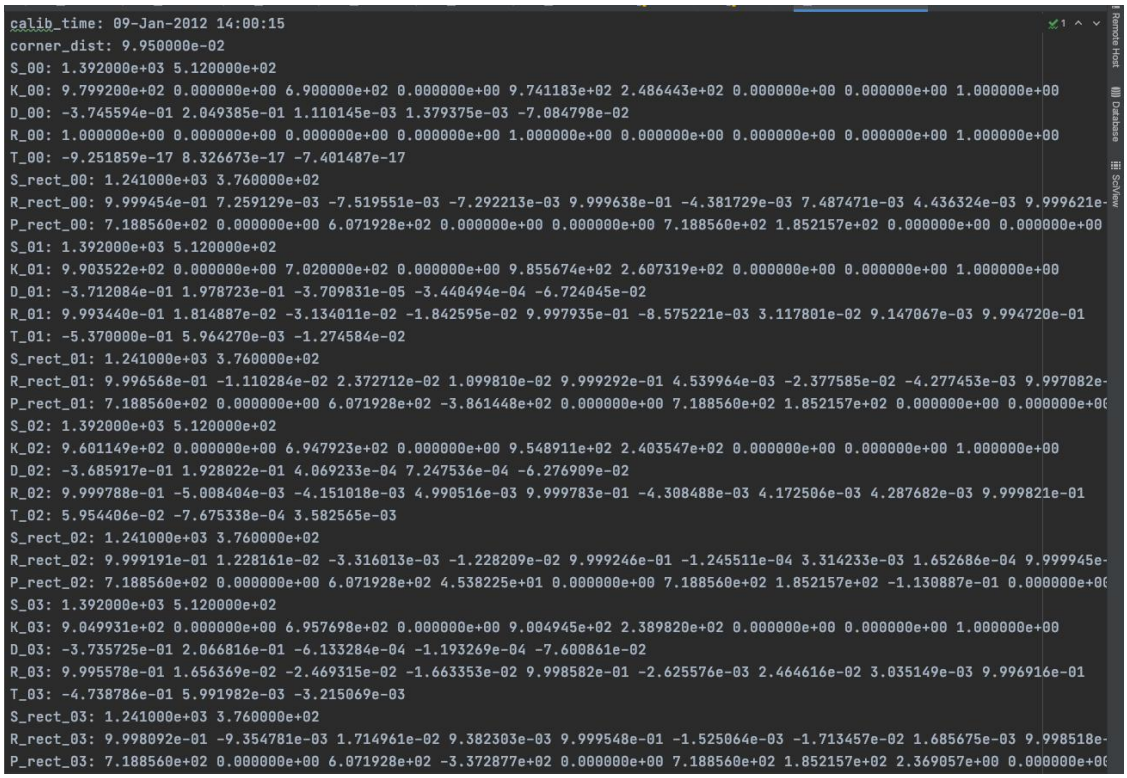
其对应的 label 文件如下：



其中每一行代表一个目标，第一列表示目标的类别，后面的列代表该目标的一些标定参数，其中项目中使用到的如下：

- 第 4 列：旋转矩阵 R 中的角度 alpha 角
- 第 5～8 列：2D Bounding Box 的标定值
- 第 9～11 列：目标 3D Bounding Box Dimension 的标定值
- 第 12～14 列：目标 3D Bounding Box Location 的标定值

另外，需要按照透射投影公式求解，还需要知道相机的内参数，Kitti 数据集提供了相机的内参数如下：



为了方便起见，我们训练和验证时只取了一个相机的内参数 P 值 **P_rect_02**，读取相机内参数的代码在/library/File.py 中，其中核心部分如下：

```
def get_P(cab_f):
    for line in open(cab_f):
        if 'P_rect_02' in line:
            cam_P = line.strip().split(' ')
            cam_P = np.asarray([float(cam_P) for cam_P in cam_P[1:]])
            return_matrix = np.zeros((3,4))
            return_matrix = cam_P.reshape((3,4))
            return return_matrix

    # try other type of file
    return get_calibration_cam_to_image
```

在一次运行的过程中，打印出从文件读取到的相机参数矩阵如下：

```
projection matrix as follow:
 [[ 7.188560e+02  0.000000e+00  6.071928e+02  4.538225e+01]
 [ 0.000000e+00  7.188560e+02  1.852157e+02 -1.130887e-01]
 [ 0.000000e+00  0.000000e+00  1.000000e+00  3.779761e-03]]
```

可以看到和我们在理论课上学习的参数矩阵的大小相同，且有若干位置为常数值

3.4.2 *数据预处理*

将原始图片数据按照经过 2D 目标检测模块得到的 Bounding Box 剪裁后的部分 resize 成 3x224x224 大小后经过一个固定参数的 normalize，这是为了能够按照规定的大小输入给 VGG19，提取特征。预处理部分的代码在 /torch_lib/Dataset.py 中，其中核心部分如下：

```
    def format_img(self, img, box_2d):

        # Should this happen? or does normalize take care of it. YOLO doesnt like
        # img=img.astype(np.float) / 255

        # torch transforms
        normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                         std=[0.229, 0.224, 0.225])
        process = transforms.Compose ([
            transforms.ToTensor(),
            normalize
        ])

        # crop image
        pt1 = box_2d[0]
        pt2 = box_2d[1]
        crop = img[pt1[1]:pt2[1]+1, pt1[0]:pt2[0]+1]
        crop = cv2.resize(src = crop, dsize=(224, 224), interpolation=cv2.INTER_CUBI
C)
```

```
        # recolor, reformat
        batch = process(crop)

        return batch
```

3.4.3 Multibins

类似于 YOLO 等 2D 检测模型中用到的 anchor 机制，本项目中实施了 Multibins 用于角度值的回归中，可以有助于弱化 L2 损失中模型忽略某一 mode 的现象，回归角度的时候，选取最大 condidence 的 bin 角度作为偏移起点，仅回归偏移量。对于划分的 bin 的数量这个超参数，原论文作者研究了不同数量下的表现，发现在 Kitti 数据集上 bin=2 表现最好，本项目选取 bin=2

Multibins 部分的代码在/torch_lib/Dataset.py，其核心部分为：

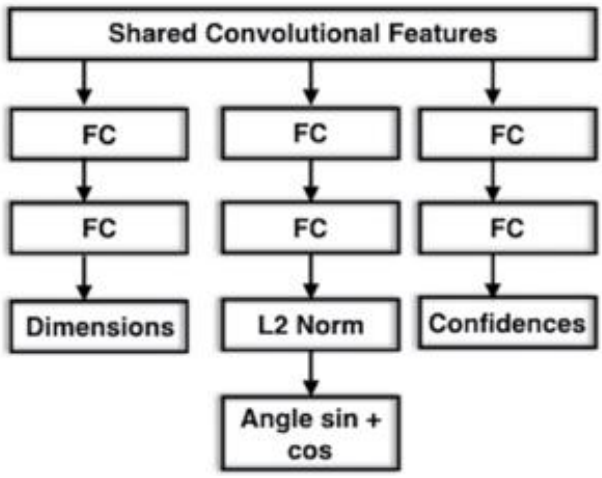
```
def generate_bins(bins):
    angle_bins = np.zeros(bins)
    interval = 2 * np.pi / bins
    for i in range(1,bins):
        angle_bins[i] = i * interval
    angle_bins += interval / 2 # center of the bin

    return angle_bins



self.overlap = overlap
# ranges for confidence
# [(min angle in bin, max angle in bin), ... ]
self.bin_ranges = []
for i in range(0,bins):
                self.bin_ranges.append(( (i*self.interval - overlap) % (2*np.pi), \
                        (i*self.interval + self.interval + overlap) % (2*np.pi)) )
```

```
# Run.py
        argmax = np.argmax(conf)
        orient = orient[argmax, :]
                # get cos(theta) ans sin(theta)
        cos = orient[0]
        sin = orient[1]
        alpha = np.arctan2(sin, cos)
        alpha += angle_bins[argmax]
        alpha -= np.pi
```

其中运行一次得到的划分 bin 的角度如下：

```
bins angle:
1.5707963267948966
4.71238898038469
```

3.4.4 *回归任务模型*



经过 VGG19 提取后，即可将特征向量作为输入输入到我们定义的回归模型中，可以观察到这个模型使用了大量的全连接层。

模型定义在/torch_lib/Model.py 中，其中核心代码如下：

```
class Model(nn.Module):
    def __init__(self, features=None, bins=2, w = 0.4):
        super(Model, self).__init__()
        self.bins = bins
        self.w = w
        self.features = features
        # for every bin, orientation has sin and cos two output
        self.orientation = nn.Sequential(
                    nn.Linear(512 * 7 * 7, 256),
                    nn.ReLU(True),
                    nn.Dropout(),
                    nn.Linear(256, 256),
                    nn.ReLU(True),
                    nn.Dropout(),
                    nn.Linear(256, bins*2) # to get sin and cos
                )
        self.confidence = nn.Sequential(
                    nn.Linear(512 * 7 * 7, 256),
                    nn.ReLU(True),
                    nn.Dropout(),
                    nn.Linear(256, 256),
                    nn.ReLU(True),
                    nn.Dropout(),
                    nn.Linear(256, bins),
                    # nn.Softmax()
                    #nn.Sigmoid()
                )
        self.dimension = nn.Sequential(
```

```
            nn.Linear(512 * 7 * 7, 512),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(512, 512),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(512, 3)
        )

    def forward(self, x):
        x = self.features(x) # 512 x 7 x 7
        x = x.view(-1, 512 * 7 * 7)
        orientation = self.orientation(x)
        orientation = orientation.view(-1, self.bins, 2)
        orientation = F.normalize(orientation, dim=2)
        confidence = self.confidence(x)
        dimension = self.dimension(x)
        return orientation, confidence, dimension
```

其中运行一次利用 torchsummary 打印模型结构如下：

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
       BatchNorm2d-2         [-1, 64, 224, 224]             128
              ReLU-3         [-1, 64, 224, 224]               0
            Conv2d-4         [-1, 64, 224, 224]          36,928
       BatchNorm2d-5         [-1, 64, 224, 224]             128
              ReLU-6         [-1, 64, 224, 224]               0
         MaxPool2d-7         [-1, 64, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]          73,856
       BatchNorm2d-9        [-1, 128, 112, 112]             256
```

ReLU–10	[–1, 128, 112, 112]	0
Conv2d–11	[–1, 128, 112, 112]	147,584
BatchNorm2d–12	[–1, 128, 112, 112]	256
ReLU–13	[–1, 128, 112, 112]	0
MaxPool2d–14	[–1, 128, 56, 56]	0
Conv2d–15	[–1, 256, 56, 56]	295,168
BatchNorm2d–16	[–1, 256, 56, 56]	512
ReLU–17	[–1, 256, 56, 56]	0
Conv2d–18	[–1, 256, 56, 56]	590,080
BatchNorm2d–19	[–1, 256, 56, 56]	512
ReLU–20	[–1, 256, 56, 56]	0
Conv2d–21	[–1, 256, 56, 56]	590,080
BatchNorm2d–22	[–1, 256, 56, 56]	512
ReLU–23	[–1, 256, 56, 56]	0
Conv2d–24	[–1, 256, 56, 56]	590,080
BatchNorm2d–25	[–1, 256, 56, 56]	512
ReLU–26	[–1, 256, 56, 56]	0
MaxPool2d–27	[–1, 256, 28, 28]	0
Conv2d–28	[–1, 512, 28, 28]	1,180,160
BatchNorm2d–29	[–1, 512, 28, 28]	1,024
ReLU–30	[–1, 512, 28, 28]	0
Conv2d–31	[–1, 512, 28, 28]	2,359,808
BatchNorm2d–32	[–1, 512, 28, 28]	1,024
ReLU–33	[–1, 512, 28, 28]	0
Conv2d–34	[–1, 512, 28, 28]	2,359,808
BatchNorm2d–35	[–1, 512, 28, 28]	1,024
ReLU–36	[–1, 512, 28, 28]	0
Conv2d–37	[–1, 512, 28, 28]	2,359,808
BatchNorm2d–38	[–1, 512, 28, 28]	1,024
ReLU–39	[–1, 512, 28, 28]	0
MaxPool2d–40	[–1, 512, 14, 14]	0
Conv2d–41	[–1, 512, 14, 14]	2,359,808

BatchNorm2d–42	[–1, 512, 14, 14]	1,024
ReLU–43	[–1, 512, 14, 14]	0
Conv2d–44	[–1, 512, 14, 14]	2,359,808
BatchNorm2d–45	[–1, 512, 14, 14]	1,024
ReLU–46	[–1, 512, 14, 14]	0
Conv2d–47	[–1, 512, 14, 14]	2,359,808
BatchNorm2d–48	[–1, 512, 14, 14]	1,024
ReLU–49	[–1, 512, 14, 14]	0
Conv2d–50	[–1, 512, 14, 14]	2,359,808
BatchNorm2d–51	[–1, 512, 14, 14]	1,024
ReLU–52	[–1, 512, 14, 14]	0
MaxPool2d–53	[–1, 512, 7, 7]	0
Linear–54	[–1, 256]	6,422,784
ReLU–55	[–1, 256]	0
Dropout–56	[–1, 256]	0
Linear–57	[–1, 256]	65,792
ReLU–58	[–1, 256]	0
Dropout–59	[–1, 256]	0
Linear–60	[–1, 4]	1,028
Linear–61	[–1, 256]	6,422,784
ReLU–62	[–1, 256]	0
Dropout–63	[–1, 256]	0
Linear–64	[–1, 256]	65,792
ReLU–65	[–1, 256]	0
Dropout–66	[–1, 256]	0
Linear–67	[–1, 2]	514
Linear–68	[–1, 512]	12,845,568
ReLU–69	[–1, 512]	0
Dropout–70	[–1, 512]	0
Linear–71	[–1, 512]	262,656
ReLU–72	[–1, 512]	0
Dropout–73	[–1, 512]	0

```
            Linear-74                  [-1, 3]           1,539
================================================================
Total params: 46,123,849
Trainable params: 46,123,849
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 351.66
Params size (MB): 175.95
Estimated Total Size (MB): 528.18
----------------------------------------------------------------
Model summary:  None
```

3.4.4 *损失函数*

原论文中将损失函数定义如下：

$$L = \alpha \times L_{dims} + L_{\theta}.$$

其中：

$$L_{\theta} = L_{conf} + w \times L_{loc}$$

$$L_{loc} = -\frac{1}{n_{\theta^*}} \sum cos(\theta^* - c_i - \Delta\theta_i)$$

$$L_{dims} = \frac{1}{n} \sum (D^* - \bar{D} - \delta)^2$$

对于 dimension，由于其方差较小，直接用 L2 损失（MSE），而对于 confidence，我们将其视为一个分类任务，只需要找到最大的 confidence 对应的 bin，因此我们使用交叉熵损失，而对于回归角度的损失函数，我们需要自定义一个损失函数。定义损失函数的部分代码如下：

```
conf_loss_func = nn.CrossEntropyLoss().cuda()
# dimension regression loss function is L2 loss
dim_loss_func = nn.MSELoss().cuda()
# orientation regression loss function is self-designed loss
orient_loss_func = OrientationLoss
```

```
def OrientationLoss(orient_batch, orientGT_batch, confGT_batch):

    batch_size = orient_batch.size()[0]
    indexes = torch.max(confGT_batch, dim=1)[1]

    # extract just the important bin
    orientGT_batch = orientGT_batch[torch.arange(batch_size), indexes]
    orient_batch = orient_batch[torch.arange(batch_size), indexes]

    theta_diff = torch.atan2(orientGT_batch[:,1], orientGT_batch[:,0])
    estimated_theta_diff = torch.atan2(orient_batch[:,1], orient_batch[:,0])

    return -1 * torch.cos(theta_diff - estimated_theta_diff).mean()
```

3.4.5 *训练*

训练过程保存在/Train.py 文件中，其核心部分如下：

```
for epoch in range(first_epoch + 1, epochs + 1):
    curr_batch = 0
    passes = 0
    for local_batch, local_labels in generator:
        # get the ground-truth of the training batch
```

```
            truth_orient = local_labels['Orientation'].float().cuda()
            truth_conf = local_labels['Confidence'].long().cuda()
            truth_dim = local_labels['Dimensions'].float().cuda()

            # get the output of the model
            local_batch = local_batch.float().cuda()
            [orient, conf, dim] = model(local_batch)

            orient_loss = orient_loss_func(orient, truth_orient, truth_conf)
            dim_loss = dim_loss_func(dim, truth_dim)

            truth_conf = torch.max(truth_conf, dim=1)[1]
            conf_loss = conf_loss_func(conf, truth_conf)

            # paper's loss function here
            loss_theta = conf_loss + w * orient_loss
            loss = alpha * dim_loss + loss_theta

        opt_SGD.zero_grad()
        loss.backward()
        opt_SGD.step()

        if passes % 10 == 0:
            print("--- epoch %s | batch %s/%s --- [loss: %s]" % (epoch, curr_batch, to
tal_num_batches, loss.item()))
            passes = 0

        passes += 1
        curr_batch += 1

    # save after every 10 epochs
```

```
if epoch % 10 == 0:
    name = model_path + 'epoch_%s.pkl' % epoch
    print("====================")
    print("Done with epoch %s!" % epoch)
    print("Saving weights as %s ..." % name)
    torch.save({
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': opt_SGD.state_dict(),
        'loss': loss
    }, name)
    print("====================")
```

训练完成后得到最终的模型保存在/weights/epoch_10.pkl

3.4 2D 检测模块实现（YOLO）

不同于原论文使用的 MS-CNN 模型，本项目的 2D 检测模块使用了预训练的 YOLO 模型，其工作方式定义在/yolo/yolo.py 中，核心部分的代码为：

```
def detect(self, image):
    # assert image is opencv
    (H, W) = image.shape[:2]

    # get the output layer
    ln = self.net.getLayerNames()
    ln = [ln[i[0] - 1] for i in self.net.getUnconnectedOutLayers()]

    # prepare input
    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB=True, crop=
False)
```

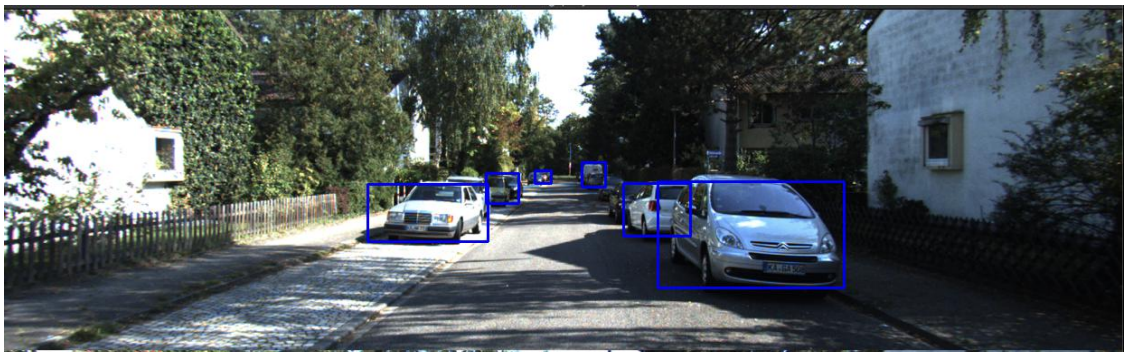
```
# feed the image and get output
self.net.setInput(blob)
output = self.net.forward(ln)

detections = []

boxes = []
confidences = []
class_ids = []

for output in output:
    for detection in output:
        # detections[5:] for all the cate scores
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]

        # if reach the threshold
        if confidence > self.confidence:
            # get the box as image pixel format (detection[0:4])
            # detection[4] is for confidence
            box = detection[0:4] * np.array([W, H, W, H])
            # detection[0:4] is for (Xc, Yc, w, h)
            (centerX, centerY, width, height) = box.astype("int")

            # use the center (x, y)-coordinates to derive the top and
            # and left corner of the bounding box
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))

            # update our list of bounding box coordinates, confidences,
```

```
                # and class IDs

                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    # NMS for drop repeat boxes
    idxs = cv2.dnn.NMSBoxes(boxes, confidences, self.confidence, self.threshold)

    if len(idxs) > 0:
        for i in idxs.flatten():
            # transform [x, y, w, h] to the coordinate of top_left and bottom_right corn
er
            top_left = (boxes[i][0], boxes[i][1])
            bottom_right = (top_left[0] + boxes[i][2], top_left[1] + boxes[i][3])

            box_2d = [top_left, bottom_right]
            class_ = self.get_class(class_ids[i])
            if class_ == "person":
                class_ = "pedestrian"

            detections.append(Detection(box_2d, class_))

    return detections
```

其作用就是检测出一张输入图片中的目标，并且返回其二维 Bounding Box 的值，利用如下图像作为输入

可以得到其检测到的结果如下：

```
2D detection results:
total object nums:  6
class:  car 2D_box:  [(727, 189), (933, 306)]
class:  car 2D_box:  [(689, 191), (763, 249)]
class:  car 2D_box:  [(406, 192), (539, 255)]
class:  car 2D_box:  [(537, 180), (573, 213)]
class:  car 2D_box:  [(643, 168), (669, 195)]
class:  car 2D_box:  [(591, 177), (609, 191)]
```



可以看到 YOLO 模型准确的检测出了输入图片中的所有汽车目标，并且成功的画出了 2D bbox

3.5 3D 解算模块实现

原论文作出了一个大胆的假设，三维框的顶点投影到图上应该包含在图像目标 2 维框内，故 8 个点投影出的 x 的最小值应该等于 2D 框的最小的 x，即左上点的 x 坐标；8 个点投影出的 y 的最小值应该等于 2D 框的最小的 y，即左上点的 y 坐标；8 个点投影出的 x 的最大值应该等于 2D 框的最大的 x，即右下点的 x 坐标；8 个点投影出的 y 的最大值应该等于 2D 框的最大的 y，即右下点的 y 坐标。故，共有 4 个方程，此方程组称为几何姿态限制

3D 解算模块主要是根据输入到回归模型中得到的若干参数和输入到 2D 目标检测模块得到的 2D Bounding Box 的值利用数学的方法以及事先规定几何姿态限制方程组解算出目标在三维的 Location（T 矩阵）参数，根据这些参数画出 3D Bounding Box

其中数学主要利用的是最小二乘法，主要实现在/library/Math.py 中，其核心部分代码如下：

```
def calc_location(dimension, proj_matrix, box_2d, alpha, theta_ray):
    #global orientation
    orient = alpha + theta_ray
    R = rotation_matrix(orient)

    # format 2d corners
    xmin = box_2d[0][0]
    ymin = box_2d[0][1]
    xmax = box_2d[1][0]
    ymax = box_2d[1][1]

    # left top right bottom
    box_corners = [xmin, ymin, xmax, ymax]

    # get the point constraints
    constraints = []

    left_constraints = []
    right_constraints = []
    top_constraints = []
    bottom_constraints = []

    # using a different coord system
    dx = dimension[2] / 2
```

```
dy = dimension[0] / 2
dz = dimension[1] / 2

# below is very much based on trial and error

# based on the relative angle, a different configuration occurs
# negative is back of car, positive is front
left_mult = 1
right_mult = -1

# about straight on but opposite way
if alpha < np.deg2rad(92) and alpha > np.deg2rad(88):
    left_mult = 1
    right_mult = 1
# about straight on and same way
elif alpha < np.deg2rad(-88) and alpha > np.deg2rad(-92):
    left_mult = -1
    right_mult = -1
# this works but doesnt make much sense
elif alpha < np.deg2rad(90) and alpha > -np.deg2rad(90):
    left_mult = -1
    right_mult = 1

# if the car is facing the oppositeway, switch left and right
switch_mult = -1
if alpha > 0:
    switch_mult = 1

# left and right could either be the front of the car ot the back of the car
# careful to use left and right based on image, no of actual car's left and right
for i in (-1,1):
```

```
    left_constraints.append([left_mult * dx, i*dy, -switch_mult * dz])
for i in (-1,1):
    right_constraints.append([right_mult * dx, i*dy, switch_mult * dz])

# top and bottom are easy, just the top and bottom of car
for i in (-1,1):
    for j in (-1,1):
        top_constraints.append([i*dx, -dy, j*dz])
for i in (-1,1):
    for j in (-1,1):
        bottom_constraints.append([i*dx, dy, j*dz])

# now, 64 combinations
for left in left_constraints:
    for top in top_constraints:
        for right in right_constraints:
            for bottom in bottom_constraints:
                constraints.append([left, top, right, bottom])

# filter out the ones with repeats
constraints = filter(lambda x: len(x) == len(set(tuple(i) for i in x)), constraints)

# create pre M (the term with I and the R*X)
pre_M = np.zeros([4,4])
# 1's down diagonal
for i in range(0,4):
    pre_M[i][i] = 1

best_loc = None
best_error = [1e09]
best_X = None
```

```
# loop through each possible constraint, hold on to the best guess
# constraint will be 64 sets of 4 corners
count = 0
for constraint in constraints:
    # each corner
    Xa = constraint[0]
    Xb = constraint[1]
    Xc = constraint[2]
    Xd = constraint[3]

    X_array = [Xa, Xb, Xc, Xd]

    # M: all 1's down diagonal, and upper 3x1 is Rotation_matrix * [x, y, z]
    Ma = np.copy(pre_M)
    Mb = np.copy(pre_M)
    Mc = np.copy(pre_M)
    Md = np.copy(pre_M)

    M_array = [Ma, Mb, Mc, Md]

    # create A, b
    A = np.zeros([4,3], dtype=np.float)
    b = np.zeros([4,1])

    indicies = [0,1,0,1]
    for row, index in enumerate(indicies):
        X = X_array[row]
        M = M_array[row]

        # create M for corner Xx
```

```
            RX = np.dot(R, X)
            M[:3,3] = RX.reshape(3)

            M = np.dot(proj_matrix, M)

            A[row, :] = M[index,:3] - box_corners[row] * M[2,:3]
            b[row] = box_corners[row] * M[2,3] - M[index,3]

        # solve here with least squares, since over fit will get some error
        loc, error, rank, s = np.linalg.lstsq(A, b, rcond=None)

        # found a better estimation
        if error < best_error:
            count += 1 # for debugging
            best_loc = loc
            best_error = error
            best_X = X_array

    # return best_loc, [left_constraints, right_constraints] # for debugging
    best_loc = [best_loc[0][0], best_loc[1][0], best_loc[2][0]]
    return best_loc, best_X
```

输入上述用于 2D 目标检测的图片，可以得到经过 3D 解算模块得到的针对于每个目标算出的 Location 值：

```
location matrix:
[2.837980940590991, 0.7207858062773429, 10.24307810296197]
location matrix:
[4.441824345423448, 1.2581971235054183, 27.72624469072927]
location matrix:
[-2.3947024381143365, 0.8270525758090734, 13.59973720247558]
location matrix:
[-3.99573588800902, 0.808014023797405, 53.6970183084768]
location matrix:
[3.738736498657595, -0.2786420972822299, 56.59866802196874]
location matrix:
[-0.8360116793293298, -0.12937309261292285, 75.05725547922587]
```

3.6 结果输出

上述输入图片经过训练好的回归模型后，得到的各目标 oriatation, dimension 及 confidence 值输出如下：

```
orientation:  tensor([[[ 0.9848, -0.1737],
         [ 0.9585, -0.2849]]], grad_fn=<DivBackward0>)
confidence:  tensor([[-6.5081,  6.8002]], grad_fn=<AddmmBackward>)
dimension:  tensor([[-0.0026, -0.0160,  0.0071]], grad_fn=<AddmmBackward>)
orientation:  tensor([[[ 0.8893, -0.4573],
         [ 0.9997,  0.0226]]], grad_fn=<DivBackward0>)
confidence:  tensor([[ 7.5420, -7.6291]], grad_fn=<AddmmBackward>)
dimension:  tensor([[-0.0685, -0.0169,  0.0050]], grad_fn=<AddmmBackward>)
orientation:  tensor([[[0.9986, 0.0526],
         [0.9825, 0.1865]]], grad_fn=<DivBackward0>)
confidence:  tensor([[-7.4472,  7.7878]], grad_fn=<AddmmBackward>)
dimension:  tensor([[-0.0247,  0.0656,  0.7190]], grad_fn=<AddmmBackward>)
orientation:  tensor([[[ 0.9949,  0.1005],
         [ 0.9062, -0.4229]]], grad_fn=<DivBackward0>)
confidence:  tensor([[-6.6566,  6.8472]], grad_fn=<AddmmBackward>)
dimension:  tensor([[0.0524, 0.0147, 0.4808]], grad_fn=<AddmmBackward>)
orientation:  tensor([[[ 0.9982, -0.0605],
         [ 1.0000, -0.0072]]], grad_fn=<DivBackward0>)
confidence:  tensor([[ 8.8070, -8.9927]], grad_fn=<AddmmBackward>)
dimension:  tensor([[0.3225, 0.2185, 1.0470]], grad_fn=<AddmmBackward>)
orientation:  tensor([[[ 0.9993, -0.0377],
         [ 0.9882,  0.1532]]], grad_fn=<DivBackward0>)
confidence:  tensor([[ 2.1937, -2.2028]], grad_fn=<AddmmBackward>)
dimension:  tensor([[-0.0013,  0.0823,  0.2312]], grad_fn=<AddmmBackward>)
```

将回归和解算得到的各种参数传入 Plot 模块，可以画出 3D Bounding Box, 其实现在/Library/Plotting.py，其核心部分代码如下：

```
def plot_3d_box(img, cam_to_img, ry, dimension, center):

    # plot_3d_pts(img, [center], center, calib_file=calib_file, cam_to_img=cam_to_img)

    R = rotation_matrix(ry)

    corners = create_corners(dimension, location=center, R=R)

    # to see the corners on image as red circles
    # plot_3d_pts(img, corners, center,cam_to_img=cam_to_img, relative=False)
```

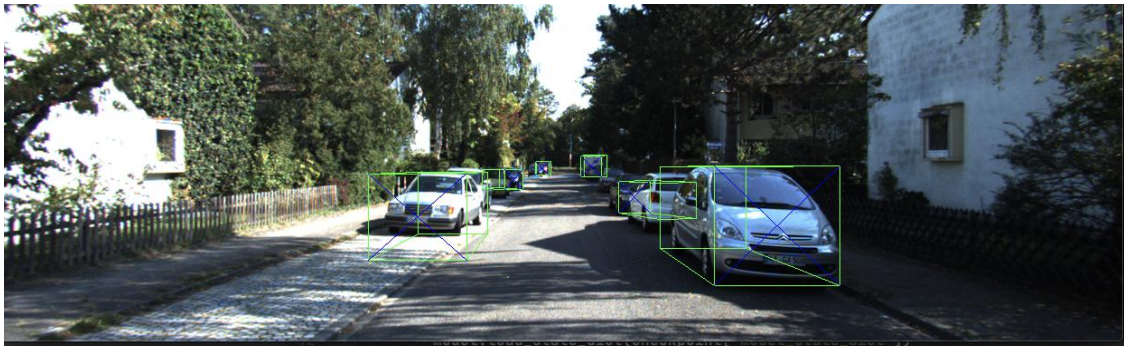
```
    box_3d = []
    for corner in corners:
        point = project_3d_pt(corner, cam_to_img)
        box_3d.append(point)

    print("3D points: ", box_3d)
    cv2.line(img, (box_3d[0][0], box_3d[0][1]), (box_3d[2][0],box_3d[2][1]), cv_colors.
GREEN.value, 1)
    cv2.line(img, (box_3d[4][0], box_3d[4][1]), (box_3d[6][0],box_3d[6][1]), cv_colors.
GREEN.value, 1)
    cv2.line(img, (box_3d[0][0], box_3d[0][1]), (box_3d[4][0],box_3d[4][1]), cv_colors.
GREEN.value, 1)
    cv2.line(img, (box_3d[2][0], box_3d[2][1]), (box_3d[6][0],box_3d[6][1]), cv_colors.
GREEN.value, 1)

    cv2.line(img, (box_3d[1][0], box_3d[1][1]), (box_3d[3][0],box_3d[3][1]), cv_colors.G
REEN.value, 1)
    cv2.line(img, (box_3d[1][0], box_3d[1][1]), (box_3d[5][0],box_3d[5][1]), cv_colors.G
REEN.value, 1)
    cv2.line(img, (box_3d[7][0], box_3d[7][1]), (box_3d[3][0],box_3d[3][1]), cv_colors.
GREEN.value, 1)
    cv2.line(img, (box_3d[7][0], box_3d[7][1]), (box_3d[5][0],box_3d[5][1]), cv_colors.
GREEN.value, 1)

    for i in range(0,7,2):
        cv2.line(img, (box_3d[i][0], box_3d[i][1]), (box_3d[i+1][0],box_3d[i+1][1]), cv_col
ors.GREEN.value, 1)

    front_mark = [(box_3d[i][0], box_3d[i][1]) for i in range(4)]
```

```
    cv2.line(img, front_mark[0], front_mark[3], cv_colors.BLUE.value, 1)
    cv2.line(img, front_mark[1], front_mark[2], cv_colors.BLUE.value, 1)
```

根据上述输入到 3D 解算模块的数据，可以得到根据其参数算出的 3D bbox 的各个点在图像中的像素位置如下：

```
3D points:  [array([928, 313], dtype=int16), array([789, 313], dtype=int16), array([928,  181], dtype=int16), array([789, 181], dtype=int16), array([824, 272], dtype=int16), arra y([729, 272], dtype=int16), array([824, 182], dtype=int16), array([729, 182], dtype=int 16)]
3D points:  [array([683, 233], dtype=int16), array([718, 233], dtype=int16), array([683,  198], dtype=int16), array([718, 197], dtype=int16), array([729, 240], dtype=int16), arr ay([769, 239], dtype=int16), array([729, 200], dtype=int16), array([769, 199], dtype=i nt16)]
3D points:  [array([515, 285], dtype=int16), array([406, 285], dtype=int16), array([515,  189], dtype=int16), array([406, 190], dtype=int16), array([538, 256], dtype=int16), arr ay([461, 256], dtype=int16), array([538, 188], dtype=int16), array([461, 188], dtype=int 16)]
3D points:  [array([577, 207], dtype=int16), array([557, 207], dtype=int16), array([577,  185], dtype=int16), array([557, 185], dtype=int16), array([551, 205], dtype=int16), arr ay([532, 206], dtype=int16), array([551, 185], dtype=int16), array([532, 185], dtype=in t16)]
3D points:  [array([641, 193], dtype=int16), array([664, 193], dtype=int16), array([641, 170], dtype=int16), array([664, 170], dtype=int16), array([645, 193], dtype=int16), arra y([670, 193], dtype=int16), array([645, 169], dtype=int16), array([670, 169], dtype=int1 6)]
3D points:  [array([591, 191], dtype=int16), array([606, 191], dtype=int16), array([591, 1 76], dtype=int16), array([606, 176], dtype=int16), array([592, 191], dtype=int16), array ([608, 191], dtype=int16), array([592, 176], dtype=int16), array([608, 176], dtype=int1 6)]
```

可以看到，对于每个目标，均产生了 8 个点，对应 3D bbox 中的 8 个顶点，根据这些点可以在二维图像中画出 3D bbox：



3.7 验证

随机选取了训练集中的若干数据和训练集中的若干数据以及我自己拍摄的若干数据进行验证：

3.7.1 *训练集数据*


SPACE for next image, any other key to exit

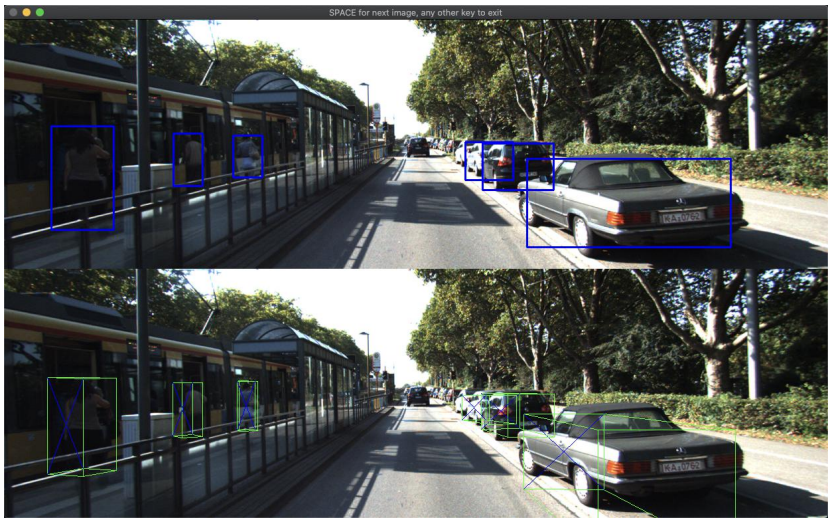

SPACE for next image, any other key to exit
Löwen Apotheke


SPACE for next image, any other key to exit
SONNE

3.7.2 *测试集数据*

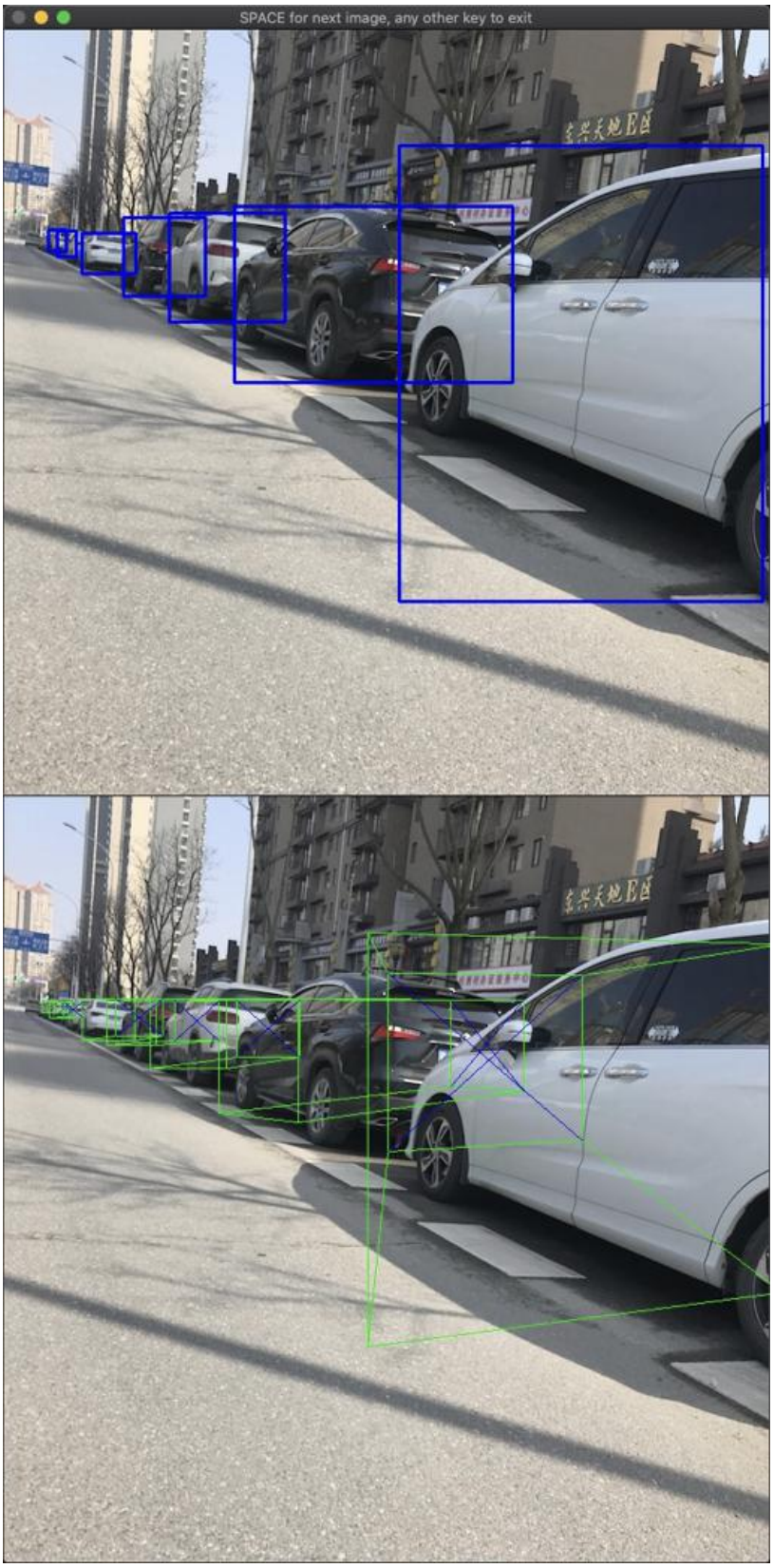

SPACE for next image, any other key to exit


SPACE for next image, any other key to exit


SPACE for next image, any other key to exit

3.7.3 *自己拍摄的数据*


SPACE for next image, any other key to exit
浙江大学
浙江大学


SPACE for next image, any other key to exit

第四章 结果分析

4.1 系统在训练数据集上的表现

由上述过程可以看出，模型在训练集和测试集上的表现都比较好，可以准确的识别出大部分的目标，成功的画出二维以及三维的 Bounding Box，但同时，其也存在有部分目标识别不出来的情况，经过对比和分析可以发现这部分目标的特征比较不明显，**它们部分是显示不完整的汽车（遮盖或者过近），部分是太过于小的目标，或者是和环境的颜色太过相近的目标**，这些目标不容易被目标检测网络发现。这也凸显了目前目标检测技术的几个技术盲点，怎么去解决这些问题是现在非常值得研究的课题。

4.2 系统在真实数据上的表现

相对于模型在测试集和训练集上的表现，其在真实数据上的表现似乎更加不尽人意。

但其实这并不能说明系统在真是数据集上工作得不好，也不能说明该模型的对于 3D 目标姿态估计的拟合能力欠佳。这是因为，分析整个预测的过程我们可以知道，由于目标的 2D Bounding Box 是由预训练的 YOLO 模型给出的，这部分 YOLO 工作得非常出色，可以看到在真实数据上基本都准确的画出了 2D Bounding Box，而对于 3D Bounding Box，它是由系统根据回归得到的参数以及 2D Bounding Box 和对应的集合学限制通过解求模块算出来的，这个计算的过程中需要用到拍摄图像数据的相机的内参数，由于真实数据的采集都是利用本人的手机采集的，对于手机的摄像头，无法获取其准确的内参数，因此可以看到系统算出的 3D Bounding Box 的效果并不好，这是因为相机内参数的不确定导致的计算误差。

同时可以发现，其对行人的识别准确率要远远高于汽车，这很可能是由于对于行人来说，旋转角这个回归参数比较统一，行人一般是直立行走，因此识别的比较准确，而对于汽车这种可能在平面上发生各种角度旋转的目标，模型则有可能不能准确的识别出其三维的姿态，但是对于二维 Bounding Box 还是画的比较准确的。倘若我们使用相机准确的内参数以及利用该相机拍摄的真实数据，系统表现的误差还可能是由于以下几条原因：

- 模型的拟合能力不够，还需要更多的数据进行训练，可以进一步对数据进行增广，加强数据

- 输入图片的尺寸不符，训练及测试用的数据统一为 1242*375 的大小，而真实数据的尺寸往往不确定，这可能在剪裁或者提取特征的时候发生信息损失
- 回归角度时的定义或者部分代码出错，导致回归出的角度值在真实数据上出现较大偏差
- 模型在训练数据集上出现了严重的过拟合现象

第五章 总结

5.1 心得体会

- 困难

  由于本项目所用到的方法、技术都是比较先进、以前从来没有接触过的，因此在项目的实现上遇到了许多麻烦，其中比较突出的麻烦一是由于知识储备不足，遇到问题没有很好的办法解决，只能不断的尝试，同时在做出结果来的时候，由于缺乏理论知识的支撑，对结果的合理分析也变得十分困难。当然还有资源上的困难，没有 GPU 供我们使用导致大多数时候我只能使用预训练的模型以及已经训练好的模型，这对我向项目中加入创新性的工作造成了很大的困难。

- 理论课程的重要性

  本项目让我真正意识到了数据决定训练效果的上限，很多时候训练的效果不好就是因为数据集不够干净，不够规整或者不够丰富，因此，对数据集的前期处理就显得非常重要了，我想这也就是我们为什么要学习《计算机视觉》这门课程，通过计算机视觉理论课程中学习到的有关图像的处理以及三维信息的表示和处理的方法有助于我们理解训练过程中间结果的意义，有助于我们理解和优化工作。

- 前沿技术并不遥远

  本次是兴趣驱动让我选择了这个课题作为课程项目，在做之前原本以为三维目标检测是遥不可及的技术，是非常前沿的技术，以我们现在的能力是没有办法理解和实现的。经过一段时间的了解和调研之后，才发现以我们目前掌握的计算机视觉和机器学习的相关知识，已经可以基本理解发表在计算机视觉顶级会议上的工作了，只是如果自己想要从事相关研究并且取得一定成果的话，还需要很长一段时间的学习，加强自身的知识储备。

- 收获

  收获肯定是非常丰富的，首先这个项目的体量是非常大的，从前期的调研到理解论文到理解每一行代码，为代码加上注释再到训练模型、在真实数据上验证，对结果进行分析。整个过程对我理解计算机视觉的相关原理和方法有非常大的帮助，也真切体会到了计算机视觉在前沿技术中的广泛应用，其在相关研究领域发挥了非常重要的作用。利用计算机视觉的相关技术，我们能实现让计算机真正拥有人的眼睛的强大能力，相信在不远的将来我们一定能看到计算机视觉的强大力量。

5.2 后续工作

本项目实现的系统识别能力有限，按照原论文作者的描述以及我个人不成熟的设想，本项目的后续研究工作可以围绕以下几点展开。

- 加入相机内参数解算模块

  目前为止，系统中所用到的相机内参数都是需要在运行前测量好的，作为存储文件的形式传递给系统使用，后续还可以加入一个模块用于解算真实数据集上的相机内参数，这样就不必要事先知道拍摄数据的相机参数，而可以实现完全自动化，系统的会更加包容和灵活。

- 扩充数据集

  按照原论文作者描述的，目前用于该模型训练的数据集的数据过于少，或许可以通过某种方法整合其他标准数据集的数据一并进行训练

- 半监督/非监督学习

  另外一种解决小数据集的办法是，开发半监督或无监督模型，就像论文 Unsupervised Learning of Probably Symmetric Deformable 3D Objects from Images in the Wild[4]所做的那样，说不定可以利用相似的技术将无监督的训练方法利用到本项目想解决的任务中，这样的话，每一辆车的行车记录仪上的数据都可以用来训练，训练数据的获取会异常容易

- 视频数据的 3Dbox 估计

  由于视频数据的连续性，我们可以利用当时桢的信息对下一桢和以后的桢进行预测，这些运动预测信息有助于提高模型的准确率

5.3 致谢

本项目能够顺利展开，首先要感谢《计算机视觉》这门课程的开设，感谢翟瑞芳老师悉心准备的理论课程，在理论课上我学习到了很多有关计算机视觉特别是 3D 视觉的相关技术和方法，由此对本项目的课题即 3D 目标检测产生了兴趣，同时也感受到了计算机视觉的无限可能和强大力量，这使得我将计算机视觉作为未来想从事的领域方向之一，并为之付诸行动。

同时，向原论文作者以及同领域内开展相关研究的其他团队致谢，他们的工作非常有价值，值得我们反复推敲和研究，也给我们提供了解决相关问题的思路和技术。

最后，向互联网上无私的贡献者以及给本项目提供理论或者技术上支持的其他同学致谢，是他们的开源代码及技术支持帮助本项目的顺利开展。

参考文献

Arsalan Mousavian, Dragomir Anguelov, John Flynn and Jana Kosecka: 3D Bounding Box Estimation Using Deep Learning and Geometry in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [1]

Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi: You Only Look Once: Unified, Real-Time Object Detection in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [2]

Waleed Ali, Sherif Abdelkarim, Mohamed Zahran, Mahmoud Zidan and Ahmad El Sallab: YOLO3D: End-to-end real-time 3D Oriented Object Bounding Box Detection from LiDAR Point Cloud in 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [3]

Shangzhe Wu, Christian Rupprecht and Andrea Vedaldi: Unsupervised Learning of Probably Symmetric Deformable 3D Objects from Images in the Wild in 2020 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [4]