机器学习纳米学位毕业项目

DeepTesla

浦轶豪

2018 年 2 月 2 日

I. 问题的定义

1.1 项目概述

自动驾驶是汽车产业与人工智能、物联网、高性能计算等新一代信息技术深度融合的产物，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。自动驾驶涉及的软硬件有传感器、高精度地图、V2X(车辆与周围的移动交通控制系统实现交互的技术)以及 AI 算法，其中 AI 算法处于相对最重要的地位。目前在算法方面已经有了大量研究成果并已开始成熟。

本项目旨在采用端到端的实现方式实现自动驾驶 AI 算法中的行驶中方向盘角度预测。项目需要利用 MIT 6.S094 这门公开课中的 Tesla 数据集训练深度学习模型[1]，根据车辆的前置相机所拍摄的路况图像，实现对车辆转向角度的预测。其中数据分为相机视频以及对应车辆转向角度的真实值。整个项目涉及到深度学习、计算机视觉相关领域的内容。

1.2 问题陈述

本项目为自动驾驶相关领域的其中一个小部分——行驶中方向盘角度预测。问题的关键在于通过数据集中的视频以及标注好的方向盘角度来预测一段全新的视频中每一帧方向盘应该转动的角度。

实现流程如下：1）分析数据并对其进行合理的预处理，2）单独拿出其中一部分作为测试集，并将剩余的数据以顺序和打乱顺序两种形式划分为训练集和验证集，3）使用 NVIDIA 提出的模型作为基准模型进行尝试测试，4）自己设计模型进行尝试测试，5）优化模型使之尽可能达到更高的准确度。

图 1 输出结果视频的其中一帧

最后得到的结果为测试集中视频进行处理的一段如图 1 格式的输出视频。其中包括真实数据与预测数据的对比显示。图 1 中间为方向盘角度对比的方向盘图形显示。其中右侧的方向盘的颜色深浅表示预测的准确度，越绿则越准确，相对越红则越不准确。底部右侧则显示真实数据与预测数据曲线图的对比，其中蓝色的是真实数据，橙色的是预测数据。

项目期望中间方向盘能尽可能保持绿色或偏绿色的状态，而在曲线图中预测数据的曲线图应尽可能与真实数据的曲线图一致。

1.3 评价指标

本项目属于回归模型，因此采用 MSE(Mean Square Error)作为损失函数对模型进行评估。以 y_作为角度预测值，y 作为角度准确值，n 为样本数量，使用 MSE（Mean Squared Error）作为损失函数，对模型准确度进行衡量。对每条数据的偏差均算入评估能够保证评估结果讲整个数据集包括在内，并且取平均保证了数据的平衡。

$$MSE = \frac{1}{\mathrm{n}}\sum_{i=0}^{n}(y\_ - y)^2$$

II. 分析

2.1 数据的探索

原始数据总共分为两部分：

1）tesla 在两种不同驾驶模式（human driving 和 autopilot）下的前置相机录制的视频（其中一个 csv 文件表明各段视频所属类别），视频为 mkv 格式。共有十段视频，第一段 50 秒，第二段 120 秒，其余均为 90 秒。视频分 1280x820。图 2 为视频中其中一帧画面。

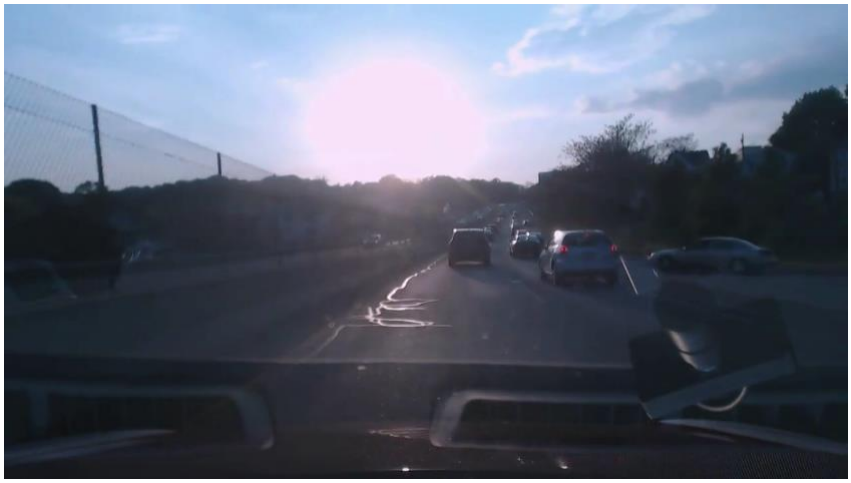

图 2 数据集包含视频中的其中一帧

2）视频对应的行驶过程中的控制信号，使用 CSV 格式，具体内容以及样例如下：

ts_micro frame_index wheel

1464305394391807 | 0 | -0.5

1464305394425141 | 1 | -0.5

1464305394458474 | 2 | -0.5

（其中，ts_micro 是时间戳，frame_index 是帧编号，wheel 是转向角度（以水平方向为基准，+为顺时针，-为逆时针））

将视频的每一帧作为输入的特征，而对于标签文件，仅采用 wheel 一列作为标签，舍弃其余对训练无用的信息。由于第十段视频将作为测试集使用，因此仅对剩余数据进行分析。视频方面道路位置基本处于同一位置，基本没有特别异常的数据。其中天空和车前盖的部分并不会提供任何有用的信息，因此需要在预处理的时候进行裁剪。表 1 对前 9 个 csv 文件的所有数据进行了统计。九个文件总共包含 24300 条数据，针对 wheel 进行分析，发现其平均值位-0.173333，趋于 0。其中视频中汽车有时会通过非常大角度的弯道，因此有时角度会相对 0 有较大的偏差。

表 1 前 9 个 csv 文件中所有数据统计

	frame	frame_index	ts_micro	wheel
count	5400.000000	18900.00000	2.430000e+04	24300.000000
mean	1616.166667	1349.50000	1.464381e+15	-0.173333
std	1120.991308	779.44343	1.437537e+11	4.594643
min	0.000000	0.00000	1.464304e+15	-18.000000
25%	674.750000	674.75000	1.464304e+15	-2.000000
50%	1349.500000	1349.50000	1.464305e+15	0.000000
75%	2549.250000	2024.25000	1.464306e+15	1.500000
max	3899.000000	2699.00000	1.464650e+15	15.000000

2.2 探索性可视化

在对表 1 进行观察的同时，图 3 对标签的分布进行了可视化。其中横轴为方

向盘转动角度，纵轴为在一定间距内角度出现的次数，总共进行了 100 等分。途中可以发现最大的几个区间均位于 0 附近，且越往两侧偏则包含的数据越少。可发现绝大多数的数据均处于-5 到 5 之间。

在视频中真实的行驶过程中，大多数情况下运行在直行路线中，方向盘角度不会有很大的偏转，且多数弯道设计均趋于平稳，使得即使是转弯也无需转动很大幅度的方向盘。

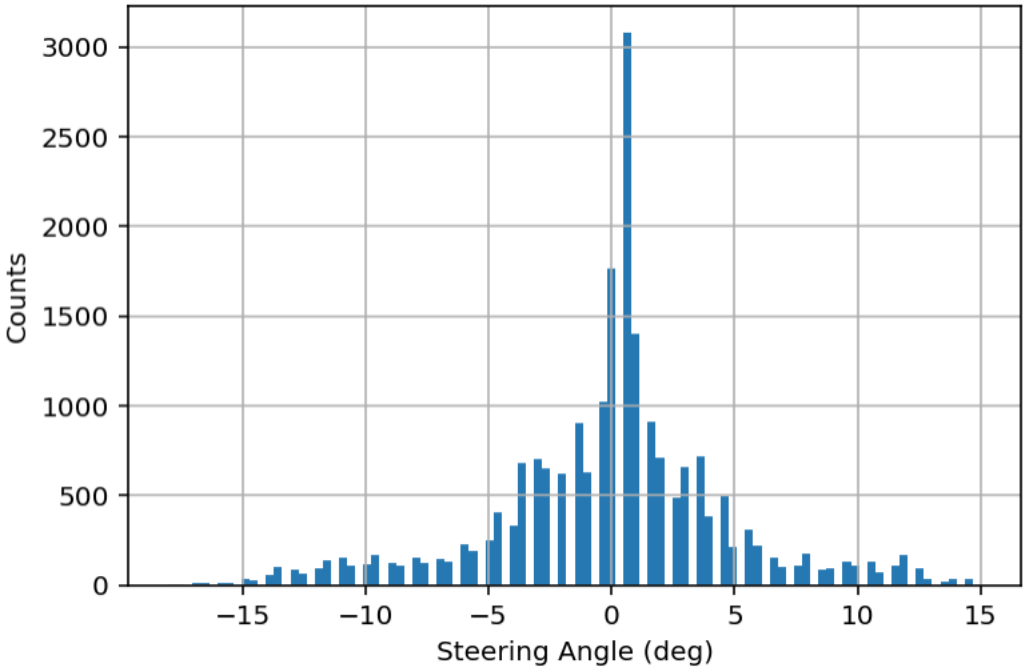

图 3 方向盘角度频率统计

2.3 算法和技术

为了实现一个端到端的实现方式，卷积神经网络能够很好地胜任这项任务。卷积神经网络与普通神经网络的区别在于，卷积神经网络包含了一个由卷积层和子采样层构成的特征抽取器。在卷积神经网络的卷积层中，一个神经元只与部分邻层神经元连接。在 CNN 的一个卷积层中，通常包含若干个特征平面(featureMap)，每个特征平面由一些矩形排列的的神经元组成，同一特征平面的神经元共享权值，这里共享的权值就是卷积核。卷积核一般以随机小数矩阵的形式初始化，在网络的训练过程中卷积核将学习得到合理的权值。共享权值（卷积核）带来的直接好处是减少网络各层之间的连接，同时又降低了过拟合的风险。子采样也叫做池化（pooling），通常有均值子采样（mean pooling）和最大值子采样（max pooling）两种形式。子采样可以看作一种特殊的卷积过程。卷积和子采样大大简化了模型复杂度，减少了模型的参数。[2]

图 4 是一个比较常见的卷积神经网络的基本结构，其中包括卷积层、激活层、池化层、以及全连接层，对于分类模型最后会使用 softmax 层作为输出层，而本项目的输出为线性输出，因此使用带一个神经元的线性输出层。

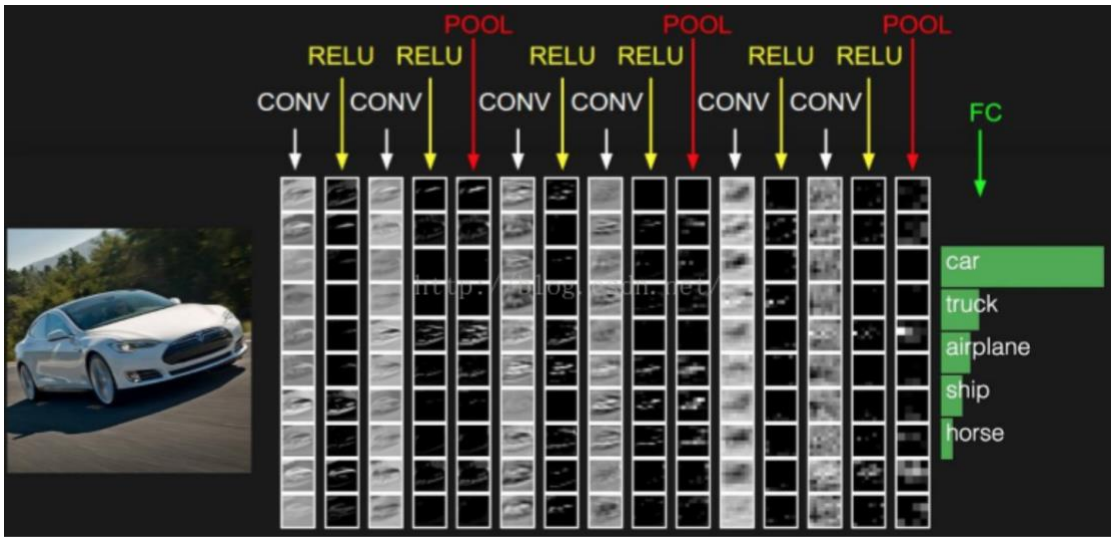

图 4 卷积神经网络基本结构示例

在卷积神经网络架构的基础上采用了近年来提出的较为有效的小技巧来提高模型的准确率。在网络内加入 dropout 层[3]以防止过拟合；加入 batchnorm 层[4]提供一个更快的收敛，使模型能够更快达到一个较好的输出效果；使用 ReLU[5]、ELU[6]等激活函数使得计算更快，且增加一定的准确率。

在整个算法中影响结果的部分的有两大部分 1）模型的设计架构，其中包括卷积层、池化层、dropout 层、batchnorm 层、全连接层等的位置摆放和数量，2）超参数的选择，其中包括卷积层核大小，步幅，边距的选择，激活函数的选择，dropout 层保留系数的设定，全连接层的大小选择，学习率的设定，训练代数的选择，批训练每批的数量。

2.4 基准模型

基准模型的选择采用了 NVIDIA 论文中提出的模型架构[1]（图 5），在此基础上对其中的细节进行了一定的变动。

原始的基准模型为一个较为简易的卷积神经网络架构。经过预处理的输入特征先经过正则化，之后使用两个卷积核为 5x5 的卷积层，两个卷积核为 3x3 的卷积层，其过滤器数分别为 24，36，48，64，展开后经过四层神经元数量分别为 1164，100，50，10 的全连接层，最后输出结果。

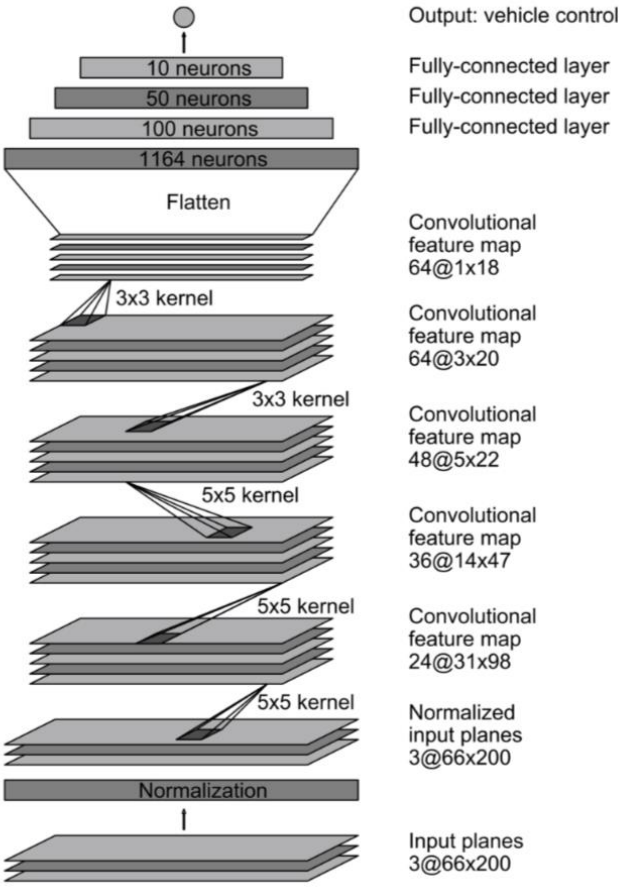

图 5 NVIDIA 提出的模型

在为其前三层全连接层后均加入 dropout 层后在经过 20 代的运算后进行曾是，以训练集顺序和随机打乱两种方式得到的模型，其测试 MSE 分别为 5.22 和 3.84。图 6 图 7 展示了基准模型的测试结果。在仅仅 20 代的训练结果下，其表

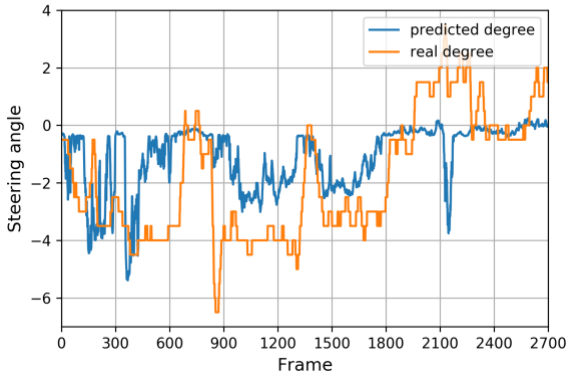

图 6 随机打乱模型的测试结果

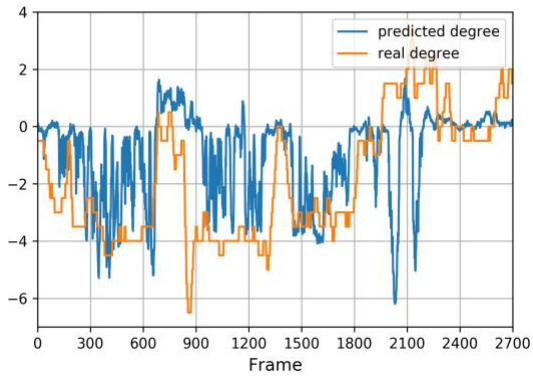

图 7 顺序模型的测试结果

现结果并没有十分理想。但是作为基准模型来说还是有一定的参考价值的。

III. 方法

3.1 数据预处理

视频处理的过程中使用 opencv 读取视频的每一帧进行预处理并且将其进行保存。由于视频的每一帧均为 720x1280 的大小作为输入数据实在过大会导致模型运算极其缓慢，预处理中对其进行了一定的缩放。同时，考虑到视频中每一帧中天空和车盖部分属于无用信息，因此特别的将其进行裁剪，保留图片中纵轴方向 350 到 553 的部分，裁剪其他内容。最后将裁剪后的数据进行正则化进入[0,1]

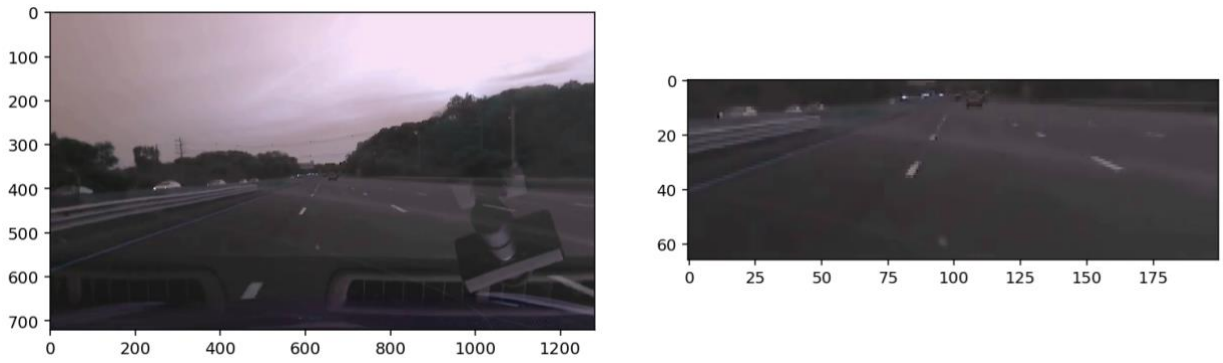

图 8 图片预处理前后对比

的区间中。图片由 720x1280 的图片提取有用信息后得到了一副 66x200 的图片作为输入特征，预处理效果如图 8。

对于输出的标签，由于其范围并没有特别大，且数据较为集中，因此判定其无需做预处理操作。

在进行训练前，首先将获取经过预处理的特征和标签。由于数据量不是很大，完全能够在内存中存下且训练模型的时候不会内存溢出，因此将所有训练数据均存入内存中以备用。

考虑到数据的随机打乱与否，本实验中同时对两种方式进行试验。因此将数据以随机和不随机打乱分别以 8：2 的比例分为训练集和验证集。以上完成所有的输入数据操作。

3.2 执行过程

在本次试验中建立的模型为经典的深度卷积神经网络。与基准模型不同，在模型架构中有所添加。初始的模型的结构如图 9，过程如下 1）通过卷积核为 5x5，过滤器数为 16，步幅为 1，边距方式为 valid 的卷积层并经过 ReLU 函数激活， 2）通过卷积核为 5x5，过滤器数为 24，步幅为 1，边距方式为 valid 的卷积层并经过 ReLU 函数激活，3）通过卷积核为 3x3，过滤器数为 32，步幅为 1，边距方式为 valid 的卷积层并经过 ReLU 函数激活，4）使用 2x2，步幅 2 进

行池化，5）通过卷积核为 3x3，过滤器数为 48，步幅为 1，边距方式为 valid 的卷积层并经过 ReLU 函数激活，6）使用 2x2，步幅 2 进行池化，7）通过卷积核为 3x3，过滤器数为 64，步幅为 1，边距方式为 valid 的卷积层并经过 ReLU 函数激活，8）使用 2x2，步幅 2 进行池化，9）展开，10）通过单元数为 512 的全连接层，并使用 ReLU 激活函数激活，11）通过保留系数为 0.2 的 dropout 层，12）通过单元数为 256 的全连接层，并使用 ReLU 激活函数激活，13）通过保留系数为 0.2 的 dropout 层，14）通过单元数为 128 的全连接层，并使用 ReLU 激活函数激活，15）通过保留系数为 0.2 的 dropout 层，16）通过单元数为 32 的全连接层，并使用 ReLU 激活函数激活，17）通过保留系数为 0.5 的 dropout 层，18）输出单元。

```
def get_my_model():
    input_shape = (66, 200, 3)
    model = Sequential()
    model.add(Conv2D(filters=16, kernel_size=5, strides=1, padding='valid', activation='relu',
                     input_shape=input_shape))
    model.add(Conv2D(filters=24, kernel_size=5, strides=1, padding='valid', activation='relu'))
    model.add(Conv2D(filters=32, kernel_size=3, strides=1, padding='valid', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='valid'))
    model.add(Conv2D(filters=48, kernel_size=3, strides=1, padding='valid', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='valid'))
    model.add(Conv2D(filters=64, kernel_size=3, strides=1, padding='valid', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='valid'))
    model.add(Flatten())
    model.add(Dense(units=512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=256, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=128, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=32, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(units=1))
    adam = optimizers.Adam(lr=1e-3)
    model.compile(loss='mse', optimizer=adam)
    return model
```

图 9 新模型的设计代码

在本模型中采用一种金字塔式的设计模式使模型能一步步逐渐获取图片的特征。前三层 Dropout 使用了一个较小的保留系数 0.2 以减少过拟合的影响，最后使用了 0.5 的 dropout 层为了减少欠拟合。模型使用 Adam 优化器，使用 1e-3 的学习速率进行学习。

图 10 展示了使用此模型进行训练的过程曲线，x 轴代表代数，y 轴代表 loss 的值。其中包括数据集打乱与顺序情况下训练集和验证集的 loss。在这过程中可以发现两种情况下均能很好的进行收敛，但是在打乱的数据集下，验证集下的收敛明显较为迅速，而顺序的数据集下，验证集下的的 loss 在 20 代内还处于一个较大值，且与训练集下的 loss 差距较大。最后一代中顺序情况的训练集 loss 为 8.4897，验证集 loss 为 12.2665；打乱情况的训练集 loss 为 7.9813，验证集 loss 为 8.8558。在测试集下两者的数据为顺序情况 MSE=4.05，打乱情况 MSE=5.80。

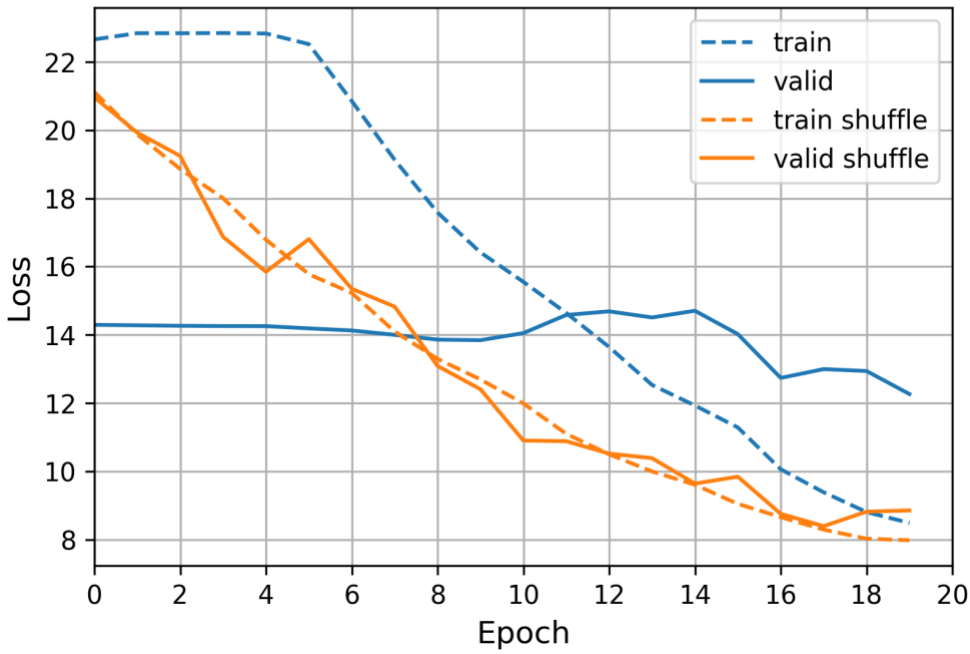

图 10 数据集打乱与顺序情况下训练 20 代的 loss 对比

这个初步设计的模型得到的最终效果与基准模型相比甚至还有些不如，在这里考虑到的模型的可能存在问题如下：1）训练代数过少，模型尚未进行充分训练，2）采用的 dropout 保留系数过下，模型存在欠拟合。

3.3 完善

在初步设计的雏形上经过一系列的分析和思考，决定对模型在各细节上完善以得到较为准确的模型。完善后的模型整体架构如图 11 中代码所示。完善后的模型在原有的金字塔结构的基础上添加了更多的层数，并且使得层与层之间的递进更为缓和。

值得提出的是，所有的激活函数由 ReLU 改为了 ELU，同时在激活曾前均采用 batchnorm 层以达到更快的收敛速度。

在模型架构上，卷积层每层均采用了 he_normal 初始化权值以达到一个更快的收敛速度，同时能够防止梯度爆炸和梯度消失。前四层核大小均为 3x3，且过滤器数量为 16，32，48，64 逐步递增。并在前四层每一层后均加入与此前相同的池化层。最后又加入了一层不带池化层，过滤器数量为 84 的卷积层。

在全连接层方面，层数也变多了，5 层全连接层的神经元数量分别为 1024，512，128，32，16。其后的 dropout 层的保留系数则由内至外呈现一个递增的趋势，分别为 0.2，0.2，0.5，0.5，0.8。最后输出层保持不变。

```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=3, strides=1, padding='valid',
                 kernel_initializer='he_normal',
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='valid'))
model.add(Conv2D(filters=32, kernel_size=3, strides=1, padding='valid',
                 kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='valid'))
model.add(Conv2D(filters=48, kernel_size=3, strides=1, padding='valid',
                 kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='valid'))
model.add(Conv2D(filters=64, kernel_size=3, strides=1, padding='valid',
                 kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Conv2D(filters=84, kernel_size=3, strides=1, padding='valid',
                 kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Flatten())
model.add(Dense(units=1024, kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dropout(0.2))
model.add(Dense(units=512, kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dropout(0.2))
model.add(Dense(units=128, kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dropout(0.5))
model.add(Dense(units=32, kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dropout(0.5))
model.add(Dense(units=16, kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Activation('elu'))
model.add(Dropout(0.8))
model.add(Dense(units=1, kernel_initializer='he_normal'))
adam = optimizers.Adam(lr=1e-3)
model.compile(loss='mse', optimizer=adam)
```

图 11 完善后的模型 keras 设计代码

在同样的 20 代的训练次数下，完善后的模型明显呈现出一个更快的收敛速度以及更高的准确率。图 12 中可以看到在顺序和打乱两种情况下，经过 20 代的训练无论是训练集的 loss 还是验证集的 loss 均收敛到一个相对于基准模型和初始模型更具优势的数值。

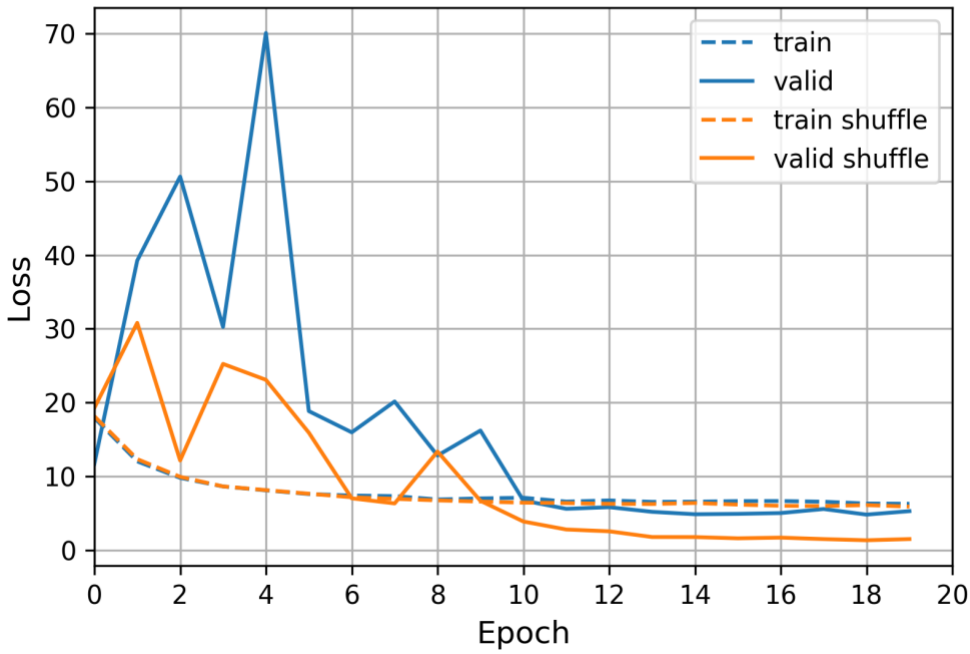

图 12 完善后的模型在两种情况下训练的折线图

IV. 结果

4.1 模型的评价与验证

在经过一系列模型的测试后，最终采用以打乱数据作为训练集的完善后的模型作为最终的模型。

此模型经过 20 代的训练后训练集的 MSE= 5.89，测试集的 MSE= 1.49。达到了一个相对较低的数值。对测试集进行预测后得到了一个相对最低的 MSE= 1.93。模型在验证集和测试集上的结果差距并不大，由此可以看出模型具有一定的鲁棒性，泛用性有一定的保证。

除了 MSE 还可视化讲测试集的预测数据与真实数据进行对比，在图 13 中我们可以发现在大多数情况下预测数据与真实数据的误差均能够处于一个较小的范围内，虽然预测数据一帧帧之间的波动较大，但整体的趋势上与真实数据基本符合。

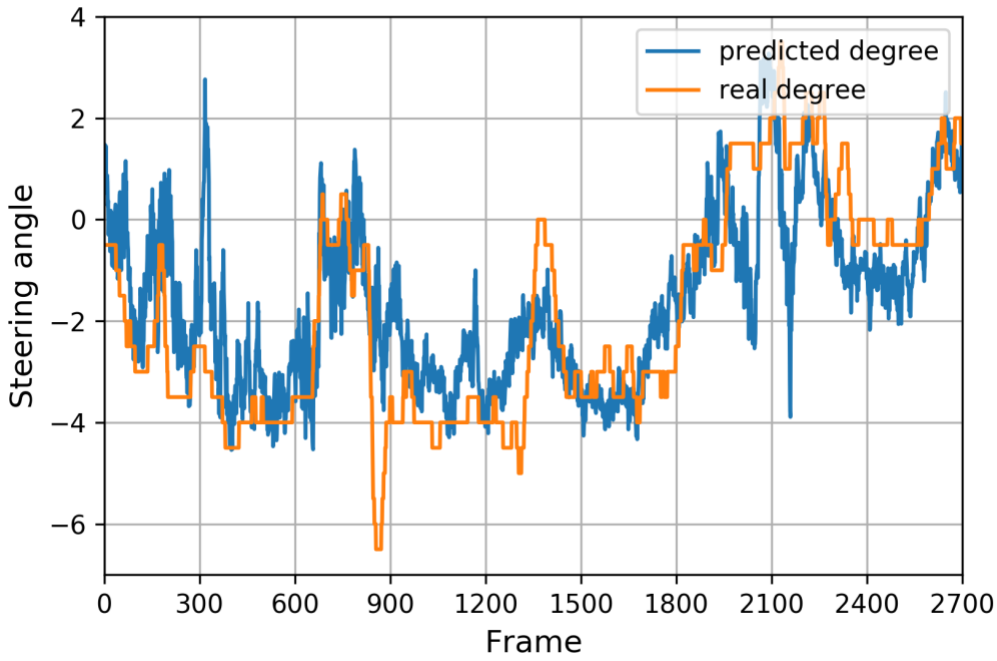

图 13 最终模型预测与真实值对比。

4.2 合理性分析

回看基准模型，表 2 给出了在 20 代后两个模型数据的对比。其中训练 loss 和验证 loss 均为最后一代训练结束后得出的数据。其中可以发现，在不对数据进行打乱的情况下，验证集的收敛效果均不是很理想，特别是基准模型的验证 loss 达到 13.95 之高。而在打乱数据的情况下，基准模型能够较好地进行拟合，并且测试能够得到较为理想的结果。

表 2 基准模型与完善模型对比

	shuffle	tain loss	val loss	test loss
benchmark model	FALSE	6.4684	13.9502	5.223405
	TRUE	6.064	8.011	3.837142
refined model	FALSE	6.2695	5.272	5.330589
	TRUE	5.8863	1.4854	1.931393

而最终模型无论是在哪个数据上均占有极大的优势，通过这些数据对比可以发现其拟合效果是最好的。

V. 项目结论（大概 1-2 页）

5.1 对项目的思考

项目的完成过程并不是特别复杂，主要分为了两部分，1）数据处理，2）模型设计及训练验证。这两部分可以相对较为独立地分割开。

在第一部分的进行过程中最无法确定的是如何分隔数据集。最终采取了对两种方式同时放入模型训练验证。

有了第一部分提供的数据，在模型设计和超参数的选择则采用了多次尝试的方式，其中过拟合和欠拟合的情况均出现过。在通过对训练曲线的分析判断属于那种问题，并针对此类问题采用常用的解决方式。调参也是项目最为困难的部分。

最终模型虽然得出了一个较为准确的结果，但是此模型的准确定还有待提升，有时存在的部分较大幅度的误差会导致在通用场景下可能会引发一系列问题。

5.2 需要作出的改进

由于设备的限制，本项目中所有的模型均只训练了 20 代，这个数字可能相对较小，可以采用一个较大代数对模型进行训练，这样也有极大可能训练出一个更为准确的模型。

除此之外，本项目还存在一个系统性的问题。实现中使用的是较为经典的深度卷积神经网络，而本项目的数据集存在一个经典卷积网络所无法获取的特征——时间顺序关联。由于视频是在时间上顺序进行的，因此相邻帧之间必然存在一定的关系，而在这一点上并没有进行特别的分析。在本项目中考虑的帧与帧之间不存在任何关联，这一点导致了结果存在一定的误差以及时间顺序下预测值存在较大的波动。使用 LSTM 有可能能够达到一定的缓和效果。

参考文献

[1] Bojarski M, Del Testa D, Dworakowski D, et al. End to end learning for self-driving cars[J]. arXiv preprint arXiv:1604.07316, 2016.

[2] http://blog.csdn.net/yunpiao123456/article/details/52437794

[3] Tinto V. Dropout from higher education: A theoretical synthesis of recent research[J]. Review of educational research, 1975, 45(1): 89-125.

[4] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//International conference on machine learning. 2015: 448-456.

[5] Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines[C]//Proceedings of the 27th international conference on machine learning (ICML-10). 2010: 807-814.

[6] Clevert D A, Unterthiner T, Hochreiter S. Fast and accurate deep network learning by exponential linear units (elus)[J]. arXiv preprint arXiv:1511.07289, 2015.