学习目标：学习集成学习方法以及交叉验证情况下的模型集成、学会使用深度学习模型的集成学习、以及对5月31号安神的直播做了一些总结

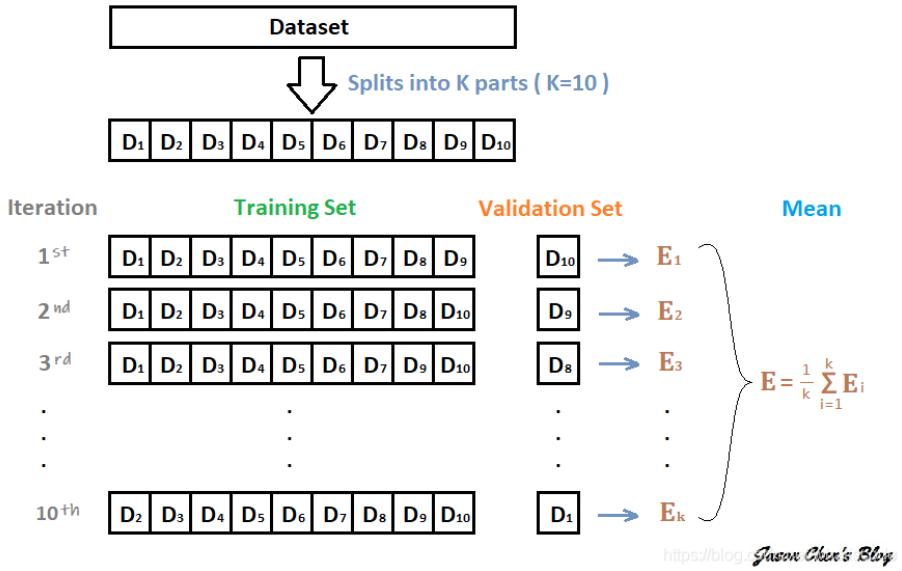
# 一、模型集成

## 1 机器学习中的集成学习方法

在机器学习中的集成学习可以在一定程度上提高预测精度，常见的集成学习方法有Stacking、Bagging和Boosting，同时这些集成学习方法与具体验证集划分联系紧密。

由于深度学习模型一般需要较长的训练周期，如果硬件设备不允许建议选取**留出法**，如果需要追求精度可以使用交叉验证的方法。

下面假设构建了10折交叉验证，训练得到10个CNN模型。




那么在10个CNN模型可以使用如下方式进行集成：

- 对预测的结果的概率值进行平均，然后解码为具体字符；
- 对预测的字符进行投票，得到最终字符。

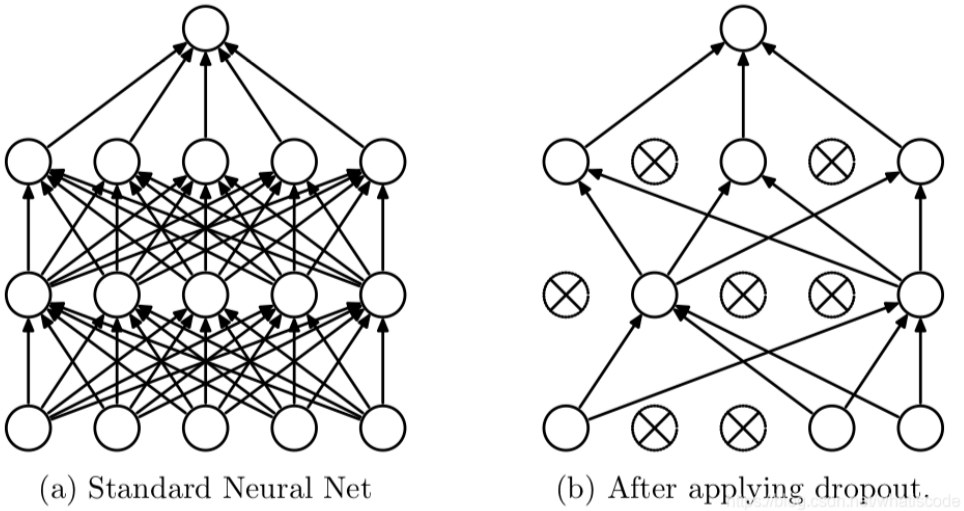
## 2 深度学习中的集成学习

此外在深度学习中本身还有一些集成学习思路的做法，值得借鉴学习：

### 2.1 Dropout

Dropout可以作为训练深度神经网络的一种技巧。在每个训练批次中，通过随机让一部分的节点停止工作。同时在预测的过程中让所有的节点都其作用。

Dropout经常出现在在现有的CNN网络中，可以有效的缓解模型过拟合的情况，也可以在预测时增加模型的精度。



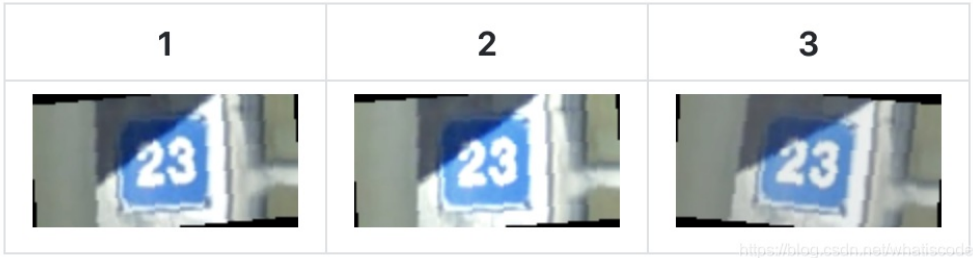

加入Dropout后的网络结构如下：

```
# 定义模型
class SVHN_Model1(nn.Module):
    def __init__(self):
        super(SVHN_Model1, self).__init__()
        # CNN提取特征模块
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2)),
            nn.ReLU(),
            nn.Dropout(0.25),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2)),
            nn.ReLU(),
            nn.Dropout(0.25),
            nn.MaxPool2d(2),
        )
        #
        self.fc1 = nn.Linear(32*3*7, 11)
        self.fc2 = nn.Linear(32*3*7, 11)
        self.fc3 = nn.Linear(32*3*7, 11)
        self.fc4 = nn.Linear(32*3*7, 11)
        self.fc5 = nn.Linear(32*3*7, 11)
        self.fc6 = nn.Linear(32*3*7, 11)

    def forward(self, img):
```

```
        feat = self.cnn(img)
        feat = feat.view(feat.shape[0], -1)
        c1 = self.fc1(feat)
        c2 = self.fc2(feat)
        c3 = self.fc3(feat)
        c4 = self.fc4(feat)
        c5 = self.fc5(feat)
        c6 = self.fc6(feat)
        return c1, c2, c3, c4, c5, c6
```

### 2.2 TTA

测试集数据扩增（Test Time Augmentation，简称TTA）也是常用的集成学习技巧，数据扩增不仅可以在训练时候用，而且可以同样在预测时候进行数据扩增，对同一个样本预测三次，然后对三次结果进行平均。



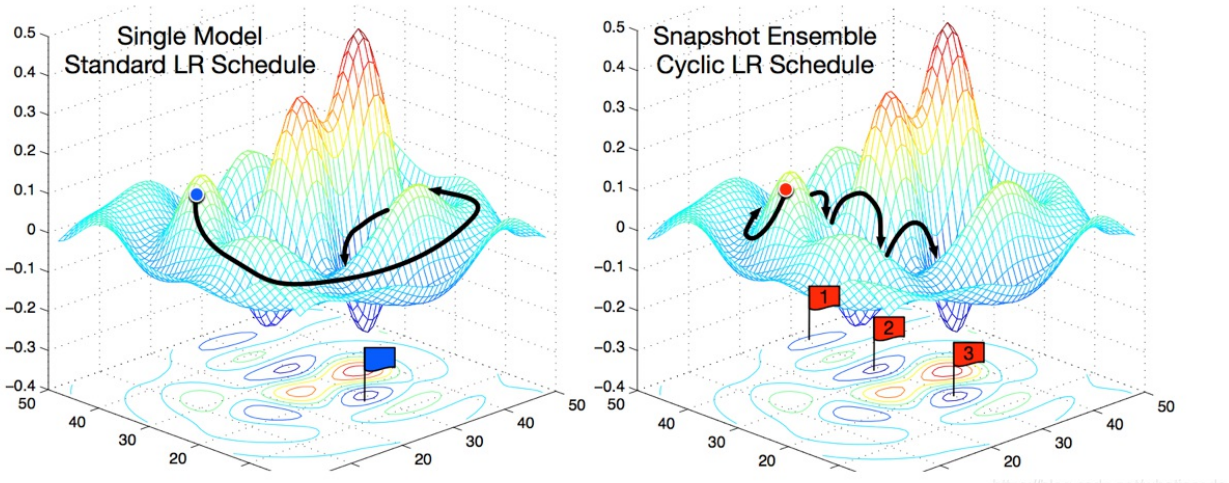


```
def predict(test_loader, model, tta=10):
    model.eval()
    test_pred_tta = None
    # TTA 次数
    for _ in range(tta):
        test_pred = []

        with torch.no_grad():
            for i, (input, target) in enumerate(test_loader):
                c0, c1, c2, c3, c4, c5 = model(data[0])
                output = np.concatenate([c0.data.numpy(), c1.data.numpy(),
                    c2.data.numpy(), c3.data.numpy(),
                    c4.data.numpy(), c5.data.numpy()], axis=1)
                test_pred.append(output)

        test_pred = np.vstack(test_pred)
        if test_pred_tta is None:
            test_pred_tta = test_pred
        else:
            test_pred_tta += test_pred

    return test_pred_tta
```

### 2.3 Snapshot

本章的开头已经提到，假设我们训练了10个CNN则可以将多个模型的预测结果进行平均。但是加入只训练了一个CNN模型，如何做模型集成呢？

在论文Snapshot Ensembles中，作者提出使用cyclical learning rate进行训练模型，并保存精度比较好的一些checkopint，最后将多个checkpoint进行模型集成。





由于在cyclical learning rate中学习率的变化有周期性变大和减少的行为，因此CNN模型很有可能在跳出局部最优进入另一个局部最优。在Snapshot论文中作者通过使用表明，此种方法可以在一定程度上提高模型精度，但需要更长的训练时间。

| | Method | C10 | C100 | SVHN |
|---|---|---|---|---|
| ResNet-110 | Single model | 5.52 | 28.02 | 1.96 |
| | NoCycle Snapshot Ensemble | 5.49 | 26.97 | 1.78 |
| | SingleCycle Ensembles | 6.66 | 24.54 | 1.74 |
| | Snapshot Ensemble ($\alpha_0 = 0.1$) | 5.73 | 25.55 | **1.63** |
| | Snapshot Ensemble ($\alpha_0 = 0.2$) | **5.32** | **24.19** | 1.66 |
| Wide-ResNet-32 | Single model | 5.43 | 23.55 | 1.90 |
| | Dropout | 4.68 | 22.82 | 1.81 |
| | NoCycle Snapshot Ensemble | 5.18 | 22.81 | 1.81 |
| | SingleCycle Ensembles | 5.95 | 21.38 | 1.65 |
| | Snapshot Ensemble ($\alpha_0 = 0.1$) | **4.41** | **21.26** | 1.64 |
| | Snapshot Ensemble ($\alpha_0 = 0.2$) | 4.73 | 21.56 | **1.51** |



### 3 结果后处理

在不同的任务中可能会有不同的解决方案，不同思路的模型不仅可以互相借鉴，同时也可以修正最终的预测结果。

在本次赛题中，可以从以下几个思路对预测结果进行后处理：

- 统计图片中每个位置字符出现的频率，使用规则修正结果；
- 单独训练一个字符长度预测模型，用来预测图片中字符个数，并修正结果。

### 4 总结

在本次Task中主要了解了深度学习模型做集成学习的各种方法，并给出了部分代码。以下几点需要注意：

- 集成学习只能在一定程度上提高精度，并需要耗费较大的训练时间，因此建议先使用提高单个模型的精度，再考虑集成学习过程；
- 具体的集成学习方法需要与验证集划分方法结合，Dropout和TTA在所有场景有可以起作用。

## 二、5月31日安神直播笔记

安神的这次直播主要分为以下四个环节：

1. 模型训练与验证
2. 调参流程
3. 模型集成
4. Q&A环节

其中模型训练与验证和调参流程在上一个学习笔记中已经提到了，模型集成在这个学习笔记中被提到。然后**调参流程**环节的部分思想来自于：http://karpathy.github.io/2019/04/25/recipe/，强烈推荐大家去看看这篇博客。

### 1 模型训练与验证

#### 1.1 为什么要设置验证集

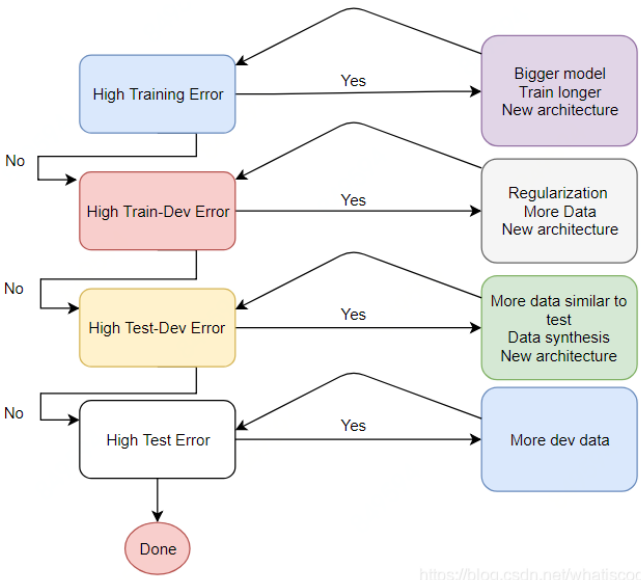
用于调整模型超参数，有效判断模型状态，防止过拟合

CNN模型的拟合能力很强，甚至会强行记住训练样本的一些无关紧要的细节来达到loss的不断下降，因此一味追求训练集loss的下降，可能导致模型在测试集的泛化效果较差

两种常见的验证集划分方法：**留出法**、K折交叉验证法

关于验证集更详细的内容可以回顾一下上一个学习笔记：https://blog.csdn.net/whatiscode/article/details/106449565

### 2 调参流程



以上这幅图非常重要。分别给出了以下四种情况下我们应该怎么样去改进模型，而不是瞎忙活：

（1）在训练集上的误差比较高（欠拟合）

- 更大的模型（增加模型的复杂度）
- 训练更久（增加epoch的次数）
- 尝试新的网络结构

（2）在训练集上已经训练得很好了，但是在验证集上效果不好（过拟合）

- 正则化（给loss加上惩罚、dropout、batch-normalization）
- 更多的数据（数据增强）
- 尝试新的网络结构

（3）在验证集上效果较好，但是在测试集上效果不好（实际中很难出现）

- 原因：可能验证集与测试集的分布出现了差异（测试集应该能反映想要应用场景的效果）
- 获取更多与测试集相似的数据
- 分析数据的分布
- 尝试新的网络结构

（4）测试集上效果不好

- 获取更多的验证集数据

**2.1 观察数据**

在你开始训练，甚至是编写任何代码之前，你需要大量的观察数据

观察什么？

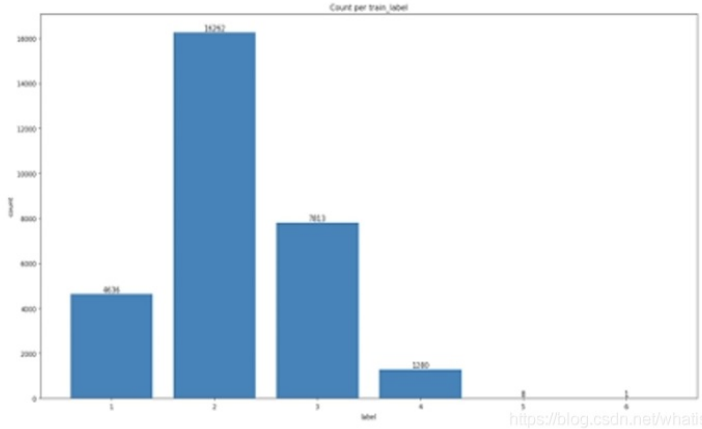
- 了解问题的背景，数据的样式，标注信息的格式等等。
- 观察数据的分布，类别是否存在偏差？
- 是否有脏数据？是否有大量重复数据？

这个过程实际上会让我们对问题有很好的理解，并且设计出合理的训练框架来求解（该课题中的baseline已经帮我们跳过了这一步骤）

对于街景字符识别项目需要思考的问题：

- 不同字符数量图片的占比，是否存在偏差？
- 是否有脏数据？是否有大量重复数据？
- 局部信息足够吗？是否需要上下文信息？
- 位置信息重要吗？
- 数据适合什么样的数据增强？什么样的数据增强不适用？

数据集中的图片字符个数分布：





通过上图我们可以看出，数据集中包含5个和6个字符的图片非常少，所以我们可以吧定长字符的个数直接设置为4，从而提高模型的效果。这也反映了观察并分析原始数据的重要性。

**2.2 搭建初始框架**

基于对数据的清晰认识，就可以搭建**baseline**训练框架了，需要遵循以下原则：

- 固定随机种子。包括python、numpy、torch、tensorflow的随机种子，这对复现遇到的bug并查找原因非常重要。
- Don't be a hero。baseline尽可能简单，无论你是不是新手，你的脑海中都可能已经浮现出了很多关于优化模型的想法了。对于baseline，不要做任何花里胡哨的操作，只使用最有把握的官方实现的网络，只使用最基本的数据增强。
- 将输入进网络的数据进行可视化。这可以很直观的看出预处理环境是否有bug
- 设置合理的评价指标，并打印或绘制足够的信息来监控训练状态。

**2.3 让baseline走上正轨**

有了最基本的baseline训练&验证框架，就可以开始实验了。这里有几个基本参数的设置经验，可以让baseline快速走上正轨：

- 选择一个合适的初始学习率。一个合适的初始学习率，应该让训练初期的每一个batch的loss非常快速的下降。通常你应该首先尝试，1e-2，1e-3这样的数值。
- 优化方法的选择：Adam is all you need。Adam具有很强的适应性，能够自适应的调整学习率的大小来完成快速收敛，这是其内在原理决定的。使用Adam优化器通常会让模型收敛更快，并且让初始学习率的可选范围变得更广。尽管对于CNN的训练来说，一个基于SGD的训练在效果上通常可以略微的超过Adam，但是其往往会花费更多的时间，所以绝大多数情况都推荐使用Adam。
- 学习率阶段性下降策略。训练时，当验证集的损失函数开始不再慢慢变低，而是开始震荡了，那么是时候考虑降低学习率了。
- 使用预训练模型。

**2.4 调参原则（重要）**

安神原话："我知道大家最不缺的就是奇思妙想，各种超参数，各种idea都想要去尝试。即便是刚刚入门的同学，也能把欠拟合-过拟合的概念以及他们可能的解决方案说得头头是道，所以在这里不再赘述。但我所要强调的是，调参的过程中，你要**时刻保持思路的清晰**，知道自己在做什么，为什么要这么做。

通常我在工作或者比赛中，会问自己这样几个问题：

- 是否理清了当前问题的主要矛盾？（只针对当前最核心的问题进行解决）
- 是否每个实验都能带来正向提升？
- 当前进行的实验是否能带来确定性的结论？（单一变量原则：即一次实验只做一处改变）

“

实战调参介绍：https://github.com/datawhalechina/dive-into-cv-pytorch/tree/master/beginner/chapter02_image_classification_introduction/2.5_SVHN_in_action

## 3 模型集成：榨出最后一点果汁

当我们已经对单模型进行了充分的调参达到不错的效果之后，我们还可以使用集成学习来尽可能的“压榨”出最后一点成绩的提升。

事实上，是非常不推荐大家通过过多的模型集成来上分的，因为这在实际应用中毫无意义。大多数正式比赛也都会通过对模型集成，以及使用模型的大小，推理速度进行限制。

关于模型集成的内容在笔记前面已经介绍了，所以不再重复。

## 4 Q&A

由于问题太多且太碎，所以只列出了前两个比较重要的问题：

1. 如何学习复杂模型，例如字符识别模型CTPN，目标检测模型faster-rcnn?

   CV的入门门槛很低，但是后面要做出很好的东西需要非常全面的能力和编程水平；主要通过看论文，读源码，复现，debug等过程一步步来提高自己，并没有太好的捷径。

2. 如何复现目标检测模型，例如faster-rcnn?

   先不要尝试自己去复现，先要去找别人比较高新的项目，然后去读他们的代码并且搞懂，搞懂之后呢，可以先尝试写一个最简单的逻辑框架，把复杂的东西全去掉，然后找到一个比较小的训练集，只要模型能够收敛，基本上你就掌握得差不多了。

## 5 小结

非常感谢安神的分享。通过本次的直播，学习到了很多关于模型训练以及调参的技巧，还有关于模型集成的内容。不过纸上得来终觉浅，绝知此事要躬行，还是要通过多实际来不断加深对这些技巧的理解，并将其真正转化为自己的经验。

正如安神所说，CV入门门槛很低，但想要有所突破还是需要全面提高自己的理论知识和编程能力。任何事情都没有捷径，踏踏实实看论文，读代码才是最好的方法。愿我们都能够在自己选择的道路上不断前行，不断进步。

最后帮安神打个广告，希望安神能够早日推出目标检测的入门教程。

《动手学CV-Pytorch版》开源项目地址：https://github.com/datawhalechina/dive-into-cv-pytorch