

本科生晋升GM记录 & kaggle比赛进阶技巧分享

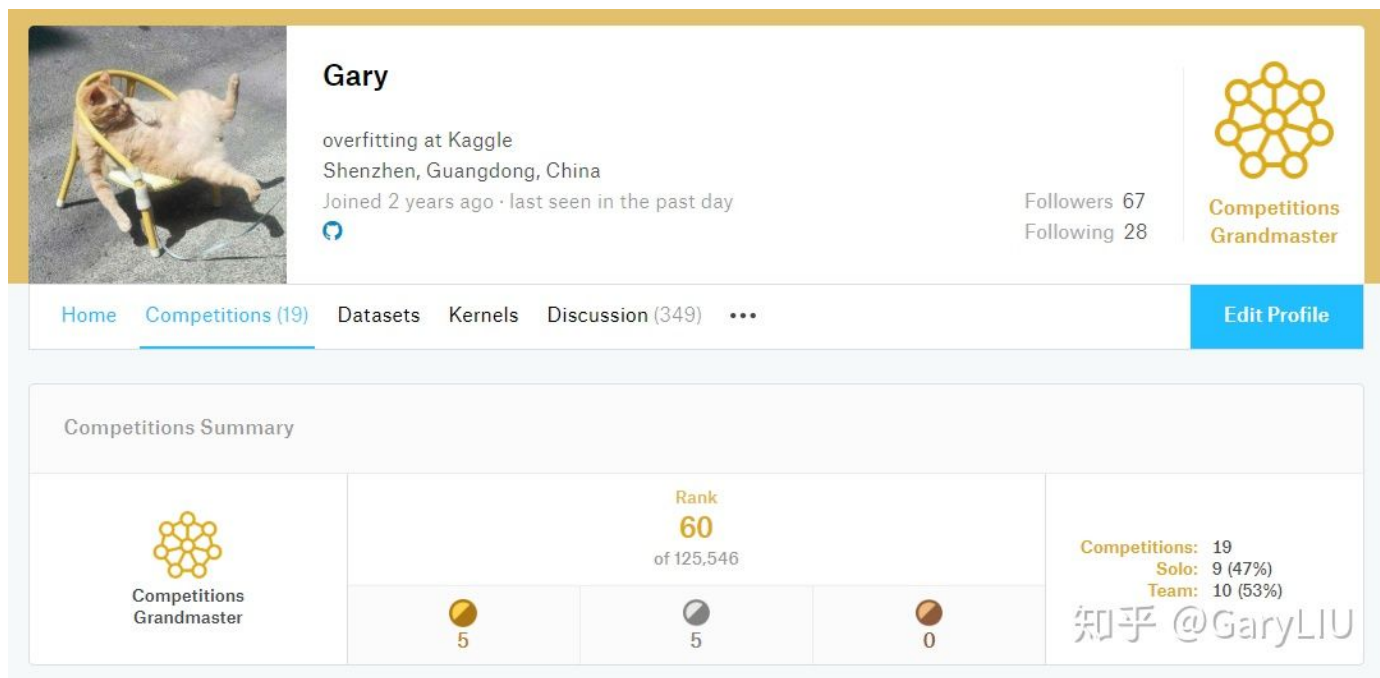
 **Gary**
kaggle GrandMaster

已关注





Sherlock、湃森、Amusi、pprp、小占同学等 947 人赞同了该文章

Kaggle profile: kaggle.com/garybios

rank: 60/125547



The screenshot shows Gary's Kaggle profile. At the top, there's a header with a profile picture of a dog, the name 'Gary', and the title 'overfitting at Kaggle'. Below this, it says 'Shenzhen, Guangdong, China' and 'Joined 2 years ago · last seen in the past day'. To the right, it shows 'Followers 67' and 'Following 28'. A 'Competitions Grandmaster' badge is also visible. Below the header is a navigation bar with links to 'Home', 'Competitions (19)', 'Datasets', 'Kernels', 'Discussion (349)', and an 'Edit Profile' button. The main content area is titled 'Competitions Summary' and displays a table with Gary's performance metrics. The table shows a rank of 60 out of 125,546, with 5 gold medals, 5 silver medals, and 0 bronze medals. It also lists 'Competitions: 19', 'Solo: 9 (47%)', and 'Team: 10 (53%)'. A watermark '知乎 @GaryLIU' is visible in the bottom right corner of the summary section.

Competitions Summary			
 Competitions Grandmaster	Rank 60 of 125,546		Competitions: 19 Solo: 9 (47%) Team: 10 (53%)
	 5	 5	 0

前言

个人其实从入坑kaggle到最近拿到了GM，其实可以分成三个阶段。

Phase 1

去年9月中的时候，刚上大四不久，之前一直对热衷于DL的我，其实都只是在自学看书，学习一些理论知识，但动手实践非常少，框架也只是会一些Tensorflow/Keras/sklearn之类的。某一天无意只是想查一些项目的开源code，自己对着学习一下，就查到了kaggle，之前在kaggle也是有账号的，但只是为了下载Dog vs Cat的数据。当时入坑的是TGS Salt Identification，一开始只能尝试着用keras去把开源kernel的代码跑一下，最后发现自己连改Unet换backbone都不会（之前都是白学了），就死皮赖脸的在讨论区发帖/水贴（参加过这个比赛的老师估计对我当时都有点印象），但幸运的是得到了许多非常热心的回答。慢慢地，对LB上分上了瘾。从一点都不会用pytorch到学会用pytorch魔改Unet模型，自己一路上分到top20，真的是非常刺激。GPU机器都是自己花在校生活费在vast.ai上租的单卡1080ti。

有了第一次在TGS获取的top2%银牌，开始在一些老师脑海中留下了些印象，到了第二个比赛Doodle Recognition Challenge，当时就和杜老师在知乎上认识了，并且和吴老师也一起组上了队伍。当然，当时的我还是对很多都非常不懂的，只是在队伍里打打下手。拿下第二个银牌升级为



Expert title后，进入了华人Kagglers大群（当时expert是进群门槛，现在已经是master了），里面全是GM/Master/Expert大神，在群里向各位老师学习。后面寒假实习面试（鹅厂），遇到了Chen Joya师兄，虽然当时不match实习岗位，但是师兄挺欣赏我（师兄真的非常非常好人，感谢），私下给我提供4 * p40+4 * v100和我一起打比赛。这个时候就是在Human Protein Atlas Image Classification和action（国内第一位本科应届生Grand Master）他们一起打了，拿下第一枚金牌，我们的solution: [kaggle Human Protein Atlas 比赛总结](#)。

Phase 2

自己当时完全想不到自己有机会拿下金牌和Master title，在比赛里也认识到了涛哥（史上最快晋升GM的男人），杨老师等。在此阶段，和涛哥/action他们继续组队，或者和群里一些老师的讨论，主要是吸收/学习他们身上的一些调参和对数据的理解的经验。同时，自己也在相关岗位实习，做research，每天更是阅读许多论文，自己的能力在此阶段成长了许多。也在此阶段拿下了两块金牌，不过时间不等人，这个时候我刚好大学毕业了。

Phase 3

当觉得自己学习到一个程度，认为自己能够独当一面，可独立解决大部分问题的时候，这时我选择了solo，但发现solo确实是比组队辛苦很多的，无队友讨论，需要自己去挖掘idea等等。幸运的是，在此阶段拿下了一枚solo金和一枚team金，成功在一年后晋升自己以前想都不敢想的Grand Master。

分享初衷

1. 鉴于国内竞赛的讨论氛围确实是非常的糟糕，同时不少小伙伴的英语阅读能力并非那么好，在kaggle经常阅读不太懂许多solution，或者不知道如何去使用kaggle，如何在这个平台上进行学习。
2. 然后，希望能够分享自己在此期间学习到的一些皮毛知识给大家，让更多的小白能够很好的入门，也能拿到一枚金牌，回馈给社区。

All tricks:

1. 以下所有的一些方法，都是一些top选手常用的，如果你使用得当，银牌是非常稳的，然后自己再加把劲点，金牌都非常有机会的。可能不够全面或者有错误的，希望各位小伙伴一起观摩观摩，如有补充的，麻烦留下评论，或者私信我，我修改补充上去。
2. 基本都是CV任务下的经验，其它类型的任务可适当借鉴下。

常用框架

编程语言：python（☆☆☆☆☆）



炼丹框架：Pytorch (☆☆☆☆☆)、Keras(☆☆☆☆)、Mxnet(☆☆☆)、Tensorflow (☆☆)

必备框架：Apex (☆☆☆☆☆)、Numpy、Opencv、PIL、Scikit-learn、albumentations、imgaug等

1. 个人看来，Pytorch的易用性其实是已经超越了另外几个框架，搭建/魔改模型都非常方便；
2. Apex可以让你只写几行代码，就可以轻松使用float16或者混合精度来训练模型，显存减少将近一半的情况下，训练速度也得到大幅度提升。
3. 自写数据增强库，推荐使用Opencv；如果是封装好的数据增强库，推荐albumentations或imgaug（或torchvision.transforms），基本想得到的transform方式都包含。

apex使用example

```
from apex import amp
import apex
...

model = resnet18()
optimizer = Adam(...)
model, optimizer = amp.initialize(model, optimizer, opt_level="O1", verbosity=0)
...
logits = model(inputs)
train_loss = criterion(logits, truth)
with amp.scale_loss(train_loss, optimizer) as scaled_loss:
    scaled_loss.backward()
optimizer.step()
```

实验pipeline(baseline)

1. 建议baseline模型从resnet18/34 or efficientnet-B0，小模型迭代快，实验进程也可以快速推进；
2. Adam优化器，SGD（可选，但是SGD没有Adam那么好调，所以baseline可以不选，后面细致调参的时候再转SGD也行。）；
3. loss function: 多分类（cross entropy），多标签分类（binary cross entropy），语义分割（binary cross entropy，如果是多类语义分割，可选cross entropy）；
4. metric: 这个就有点难度，一般根据比赛评测指标来选择；
5. 数据增强：可以为空，因为是baseline，需要数据增强后面是可以逐渐试错找到有效的再添加上去。

如何提升搭建baseline的能力

对于参赛者而言，往往具有一个好的baseline，或有着一套属于自己风格的pipeline，其实是已经成功了一半。好的baseline等价于一个好的起点，后面的改进遇到的阻碍相对也会少非常多。



但是，要把baseline写好，其实是需要不少场比赛下来的经验或者是已经有过相关项目的工作经验。

1. 对于初学者，每场比赛，都会有许多kagglers将自己的baseline开源到kernel上，你所要做的是，不是直接copy，而是去学习，从多个kernel中取出其比较精妙的部分，再组合成自己的baseline；
2. 在以前相关类型的比赛里，模仿top solution codes，整理出一个baseline。
3. 多次实践下来，你基本掌握了自己动手写baseline的能力。

调参技巧

调参是比赛环节里最重要的一步（日常工作也都是一样离不开调参的），好的learning rate和learning rate schedule对比不合适的，得到的结果也是千差万别，optimizer的选取或许多比较fancy的finetune技巧也是类似的结果。

1. **Adam**: $\text{init_lr}=5\text{e-}4(3\text{e-}4)$ (☆☆☆☆☆)， $3\text{e-}4$ 号称是Adam最好的初始学习率，有理有据，请看下图；SGD就更考验调参功力，这里就不详说（因为我也一般般）。



Andrej Karpathy ✓
@karpathy



3e-4 is the best learning rate for Adam, hands down.

♡ 409 11:01 AM - Nov 24, 2016



💬 124 people are talking about this

知乎 @GaryLiu

2. lr schedule

- ReduceLROnPlateau, $\text{patience}=4$ (5), $\text{gamma}=0.1$, 这是我常用的一套组合，并不是最好的；
- StepLR, 个人比较喜欢用这个，自己设定好在哪个epoch进行学习率的衰减，个人比较喜欢用的衰减步骤是 $[5\text{e-}4(3\text{e-}4), 1\text{e-}4, 1\text{e-}5, 1\text{e-}6]$ ，至于衰减位置，就需要自己有比较好的直觉，或者就是看log调参，对着2.1上训练的valid loss走势，valid loss不收敛了，咱就立刻进行衰减；
- CosineAnnealingLR+Multi cycle,这个相较于前两个，就不需要太多的调参，可以训练多个cycle，模型可以找到更多的局部最优，一般推荐 $\text{min_lr}=1\text{e-}6$ ，至于每个cycle多少epoch这个就说不准了，不同数据不太一样。

3. **finetune**，微调也是有许多比较fancy的技巧，在这里不做优劣比较，针对分类任务说明。

- **微调方式一**，最常用，只替换掉最后一层fc layer，改成本任务里训练集类别数目，然后不做其余特殊处理，直接开始训练；
- **微调方式二**，在微调一的基础上，freeze backbone的参数，只更新（预训练）新的fc layer的参



数（更新的参数量少，训练更快）到收敛为止，之后再放开所有层的参数，再一起训练；

- **微调方式三**，在微调方式二预训练fc layer之后或者直接就是微调方式一，可选择接上差分学习率（discriminative learning rates）即更新backbone参数和新fc layer的参数所使用的学习率是不一致的，一般可选择差异10倍，理由是backbone的参数是基于imagenet训练的，参数足够优秀同时泛化性也会更好，所以是希望得到微调即可，不需要太大的变化。

```
optimizer = torch.optim.Adam([{'params': model.backbone.parameters(), 'lr': 3e-5},  
{'params': model.fc.parameters(), 'lr': 3e-4},    ])
```

- **微调方式四**，freeze浅层，训练深层（如可以不更新resnet前两个resnet block的参数，只更新其余的参数，一样是为了增强泛化，减少过拟合）。

4. **Find the best init_lr**，前面说到3e-4在Adam是较优的init_lr，那么如何寻找最好的init_lr？

- 出自fastai, lr_find(), 其原理就是选取loss function仍在明显降低的较大的学习速率，优劣性其实也是相对而言，不一定是最好的。

5. **learning rate warmup**，理论解释可以参 [zhihu.com/question/3380...](https://www.zhihu.com/question/3380...)

6. 如果模型太大的同时你的GPU显存又不够大，那么设置的batch size就会太小，如何在有限的资源里提升多一点？

- 梯度累计（gradient accumulation），其实就是积累多个batch的梯度之后，再进行梯度的回传做参数的更新，变相的增大了训练的batch size，但缺点是对Batch Normalization没影响的。。
- 如果你卡多，这时可以使用多卡并行训练，但要使用syncbn（跨卡同步bn），即增大了batch size，又对Batch Normalization起到相同的作用。

分类赛技巧

1. label smoothing

分类任务的标签是one-hot形式，交叉熵会不断地去拟合这个真实概率，在数据不够充足的情况下拟合one-hot容易形成过拟合，因为one-hot会鼓励正确类别与所属类别之间的差异性尽可能大，但其实有不少类别之间是极为相似的。label smoothing的做法其实就是将hard label变成soft label。

2. topk-loss(OHEM)

OHEM最初是在目标检测上提出来的，但其实思想是所有领域任务都通用的。意思就是提取当前batch里top k大的loss的均值作为当前batch的loss，进行梯度的计算和回传。其insight也很简单，就是一中hard mining的方法，一个batch里会有easy sample和hard sample，easy sample对网络的更新作用较小（loss值小，梯度也会小），而hard sample的作用会更大（loss值大，梯度值也会大），所以topk-loss就是提取hard sample。




```
loss = criterion(logits, truth)
loss,_ = loss.topk(k=..)
loss = loss.mean()
```

3. weighted loss

weighted loss其实也算是一种hard mining的方法，只不过这种是人为地认为哪种类别样本更加hard，哪种类别样本更加easy。也就是说人为对不同类别的loss进行进行一个权重的设置，比如0,1类更难，设置权重为1.2，2类更容易，设置权重为0.8。。

```
weights = [1.2, 1.2, 0.8]
class_weights = torch.FloatTensor(weights).to(device)
criterion = torch.nn.CrossEntropyLoss(weight=class_weights)
```

4. dual pooling

这种是在模型层进行改造的一种小trick了，常见的做法：global max/average pooling + fc layer，这里试concat(global max-pooling, global average pooling) + fc layer，其实就是为了丰富特征层，max pooling更加关注重要的局部特征，而average pooling试更加关注全局的特征。不一定有效，我试过不少次，有效的次数比较少，但不少人喜欢这样用。

```
class res18(nn.Module):
    def __init__(self, num_classes):
        super(res18, self).__init__()
        self.base = resnet18(pretrained=True)
        self.feature = nn.Sequential(
            self.base.conv1,
            self.base.bn1,
            self.base.relu,
            self.base.maxpool,
            self.base.layer1,
            self.base.layer2,
            self.base.layer3,
            self.base.layer4
        )
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)
        self.reduce_layer = nn.Conv2d(1024, 512, 1)
        self.fc = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )
    def forward(self, x):
        bs = x.shape[0]
        x = self.feature(x)
```



```

x1 = self.avg_pool(x).view(bs, -1)
x2 = self.max_pool(x).view(bs, -1)
x = torch.cat([x1, x2], dim=1)
x = self.reduce_layer(x).view(bs, -1)
logits = self.fc(x)
return logits

```

5. margin-based softmax

- 在人脸识别领域，基于margin的softmax loss其实就是对softmax loss的一系列魔改（large margin softmax、NormFace、AM-softmax、CosFace、ArcFace等等），增加类间margin，当然也有其它的特点，如weight norm和基于余弦角度的优化等等。其共同目标都是为了获得一个更加具有区分度的feature，不易过拟合。
- 一个比较多同学忽略的点是，如果使用了margin-based softmax，往往连同开源repo里默认的超参数也一起使用了，比如 $s=32.0$ ， $m=0.5$ ，但其实这两个参数的设定都是有一定的缘由，比如 s 值象征着超球体的体积，如果类别数较多，那么 s 应该设置大点。如果你没有很好的直觉，那grid search一波，搜索到适合的 s 和 m 值也不会花很多时间。

6. Lovasz loss

- 这个loss本来是出于分割任务上的，其优化的是IOU，但你如果仔细观察lovasz传入的logit和truth，可以发现是和multi label classification类似，logit和truth都是由多个1值的one-hot形式。
- 所以在多标签分类任务上，其实是可以利用lovasz loss来进行优化的，出自（Bestfitting）
(kaggle.com/c/human-prot...)

分类赛技巧（openset/检索）

1. **BNNeck**(出自罗浩博士的Bag of Tricks and A Strong Baseline for Deep Person Re-identification)，知乎链接[一个更加强力的ReID Baseline](#)，其实就是在feature层和fc layer之间增加一层Batch Normalization layer，然后在retrieval的时候，使用BN后的feature再做一个l2 norm，也就是retrieval with Cosine distance。

```

class res50(torch.nn.Module):
    def __init__(self, num_classes):
        super(res50, self).__init__()
        resnet = resnet50(pretrained=True)
        self.backbone = torch.nn.Sequential(
            resnet.conv1,
            resnet.bn1,
            resnet.relu,
            resnet.layer1,
            resnet.layer2,
            resnet.layer3,
            resnet.layer4

```



```

    )
    self.pool = torch.nn.AdaptiveMaxPool2d(1)
    self.bnneck = nn.BatchNorm1d(2048)
    self.bnneck.bias.requires_grad_(False) # no shift
    self.classifier = nn.Linear(2048, num_classes, bias=False)
def forward(self, x):
    x = self.backbone(x)
    x = self.pool(x)
    feat = x.view(x.shape[0], -1)
    feat = self.bnneck(feat)
    if not self.training:
        return nn.functional.normalize(feat, dim=1, p=2)
    x = self.classifier(feat)
    return x

```

2. margin-based softmax (上面已经说到)

3. **triplet loss + softmax loss**, 结合metric learning, 对feature进行多个loss的优化, triplet loss也是可以有许多的花样, Batch Hard Triplet Loss, 是针对triplet loss的一种hard mining方法。

4. **IBN**, 切换带有IBN block的backbone, 搜图 (open-set) 往往test和train是不同场景下的数据, IBN block当初提出是为了提高针对不同场景下的模型泛化性能, 提升跨域 (cross domain) 能力, 在reid下的实验, IBN表现优异。

5. center loss

6. **Gem, generalized mean pooling**, 出自Fine-tuning CNN Image Retrieval with No Human Annotation, 提出的是一种可学习的pooling layer, 可提高检索性能, 代码出自github.com/tuananh1007/...

```

class GeM(nn.Module):
    def __init__(self, p=3, eps=1e-6):
        super(GeM, self).__init__()
        self.p = Parameter(torch.ones(1)*p)
        self.eps = eps
    def forward(self, x):
        return LF.gem(x, p=self.p, eps=self.eps)
    def __repr__(self):
        return self.__class__.__name__ + '(' + 'p=' + '{:.4f}'.format(self.p.data.tolist()[0])

```

7. **global feature + local features** 将全局特征和多个局部特征一起融合, 其实就是一种暴力融合特征的方法, 对提升精度有一定的帮助, 就是耗时相对只使用global feature来说很多点, 此种方法可参考在reid常用的PCB(Beyond Part Models: Person Retrieval with Refined Part



Pooling)或MGN(Learning Discriminative Features with Multiple Granularities for Person Re-Identification)方法

8. **re-ranking**，是一种在首次获取检索图的候选图里做一次重新排序，获得更加精准的检索，相对比较耗时间，不适合现实场景，适合比赛刷精度。

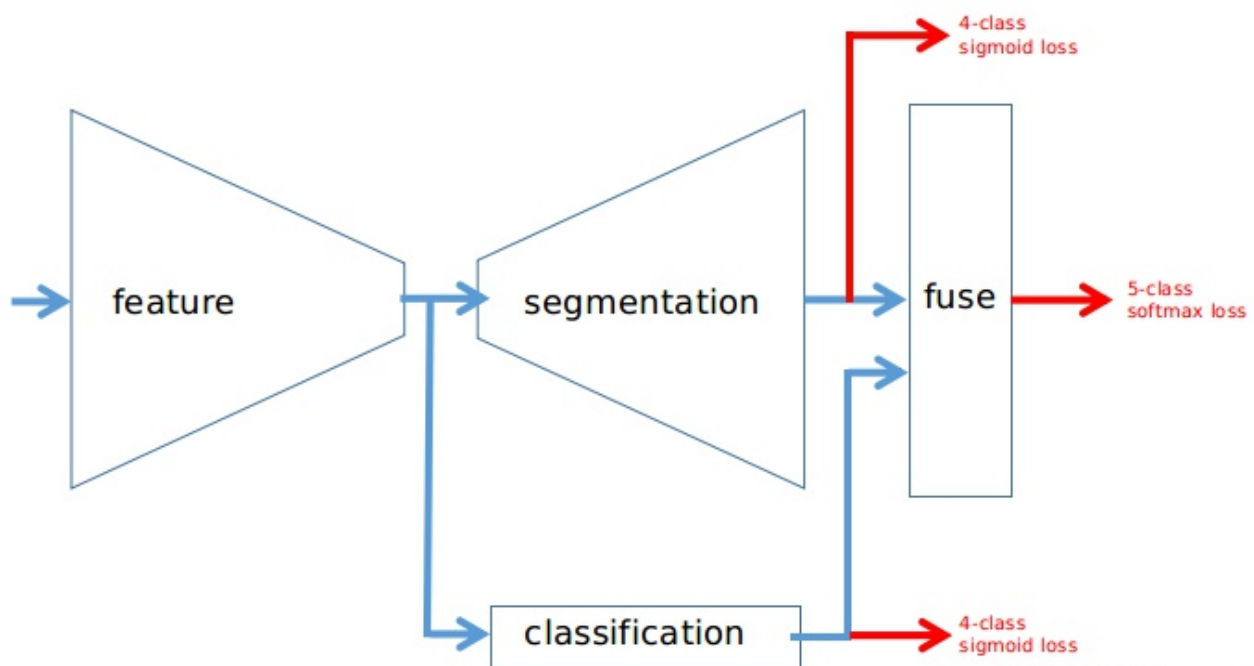
分割赛技巧

1. **Unet** Unet可以说是在kaggle的语义分割赛里的一个较优的选择，许多top solution都是使用了Unet，FPN也是一个非常不错的选择。

2. Unet的魔改

现在有个开源库其实是已经集成了许多不同分割网络，表现也是相对不错的，如果觉得自己修改比较困难，或者自己改得不够好，可以尝试使用这个库**segmentation_models_pytorch**

- 很多top solution都是修改Unet的Decoder，最常见的就是增加scse block和Hypercolumn block，也有一些是使用了CBAM (Convolutional Block Attention Module, bestfitting比较喜欢用) 或BAM (Bottleneck attention module)，这些注意力block一般是放在decoder不同stage出来的feature后面，因为注意力机制往往都是来优化feature的。
- dual head(multi task learning)，也就是构造一个end2end带有分割与分类的模型。同时，多任务学习往往会降低模型过拟合的程度，并可以提升模型的性能。



出自蛙神-Heng CherKeng

```
import segmentation_models_pytorch as smp
class Res34_UNET(nn.Module):
    def __init__(self, num_classes):
        super(Res34_UNET, self).__init__()
        self.model = smp.Unet(encoder_name='resnet34', encoder_weights='imagenet', cla
        self.avgpool = nn.AdaptiveAvgPool2d((1,1))
```

```
self.cls_head = nn.Linear(512, num_classes, bias=False)
```

```
def forward(self, x):  
    global_features = self.model.encoder(x)  
    cls_feature = global_features[0]  
    cls_feature = self.avgpool(cls_feature)  
    cls_feature = cls_feature.view(cls_feature.size(0), -1)  
    cls_feature = self.cls_head(cls_feature)  
    seg_feature = self.model.decoder(global_features)  
    return seg_feature, cls_feature
```

3. **lovasz loss** 之前在TGS Salt Identification的适合，lovasz对分割的效果的表现真的是出类拔萃，相比bce或者dice等loss可以提高一个档次。但是最近的分割赛这个loss的表现就一般，猜测是优化不同metric，然后不同loss就会带来不同的效果，又或者是数据的问题。

4. **dice loss for postive, bce loss for negtive** 主要就是将分割任务划分两个任务：1. 分割任务，2. 分类任务 dice loss可以很好的优化模型的dice score，而bce loss训练出来的分类器可以很好地找出negative sample，两者结合可以达到一种非常好效果，详细解说可以参考我之前的一个solution: [Kaggle Understanding Clouds 7th place总结](#)

通用技巧

大部分都是牺牲推断速度来提高精度的方法，适合比赛，不适合现实场景。

1. **TTA (Test Time Augmentation)** 一种暴力测试的方法，将有效的增强方式得到多个不同的input，然后进行infer，得到多个结果进行融合，一般会比原始input会高不少。这种方法的缘由就是希望通过对input进行不同的变换方式获取多个不同的但重要的特征，然后可以得到多个具有差异性的预测结果。
2. **多尺度训练，融合** 在训练期间，随机输入多种尺度的图像进行训练，如（128128，196196，224224，256256，384*384等等）然后测试的时候可适当的选取其中某几个尺度表现优异的预测结果出来融合，这种方法其实就是为了提升模型对尺度的变换的鲁棒性，不易受尺度变换的影响。
3. **Ensemble**
 - **Snapshot Ensembles**，这个方法常在与cycle learning rate的情况下使用，在不同cycle下，模型会产出多个不同的snapshot weight（多个不同的局部最优，具有差异性），这时可以将这几个snapshot model一起进行推断，然后将预测结果进行平均融合。
 - **SWA, Stochastic Weight Averaging**，随机权重平均，其实现原理当模型在训练时收敛到一定程度后，开始追踪每次epoch后得到的模型的平均值，有一个计算公式和当前的模型权重做一个平均得到一个最终的权重，提高泛化性能。
 - **stacking**，在分类任务里，stacking是作为一种2nd level的ensemble方法，将多个“准而不同”的基分类器的预测集成与一身，再扔进去一个简单的分类器（mlp、logit regression、simple cnn，xgboost等）让其自己学习怎么将多个模型融合的收益做到最高。一般数据没有问题的话，stacking会更加稳定，不易过拟合，融合的收益也会更高。

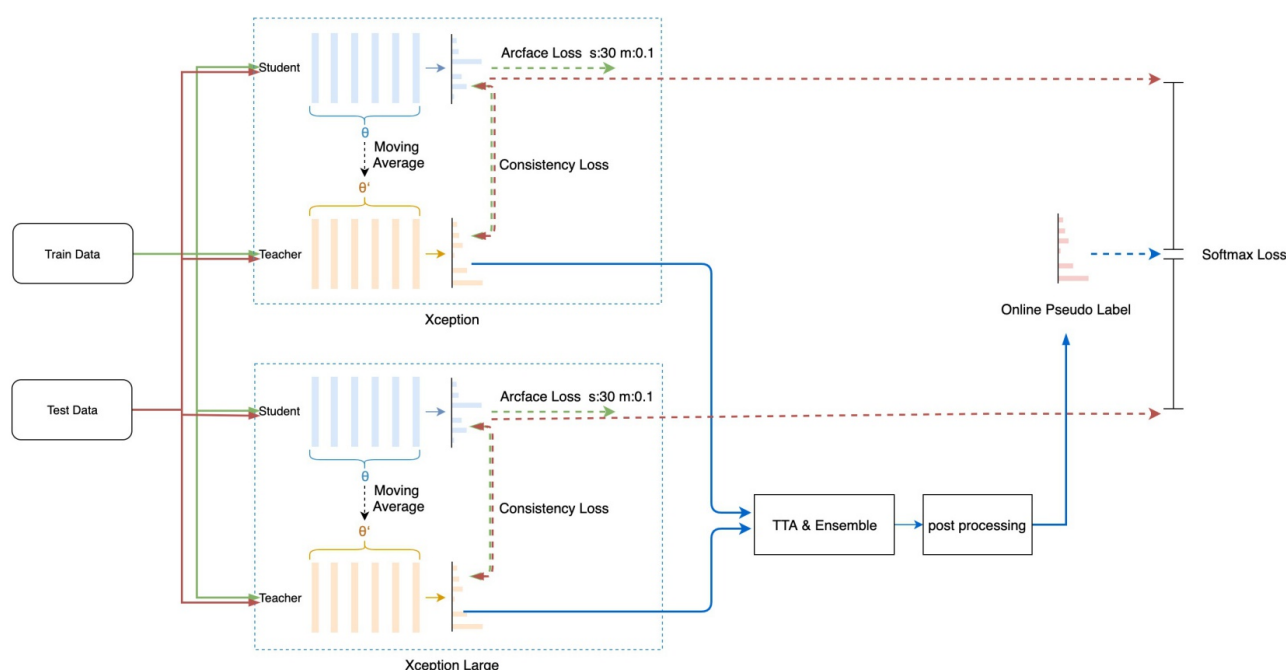


4. **设计metric loss** 许多小伙伴会有这样一个疑惑，比赛的评测metric往往和自己训练时所使用的loss优化方向不是那么一致。比如多标签分类里的metric是fbeta_score，但训练时是用了bce loss，经常可以看到val loss再收敛后会有一个反弹增大的过程，但此时val fbeta_score是还在继续提升的。这时就可以针对metric来自行设计loss，比如fbeta loss就有。

5. semi-supervised learning

- **recursive pseudo-label (伪标签)**，伪标签现在已经是kaggle赛里一个必备工具了，但是这是个非常危险的操作，如果没有筛选好的伪标签出来，容易造成模型过拟合伪标签里的许多噪声。比较安全的方法是：1. 筛选预测置信度高的样本作为伪标签，如分类里，再test里的预测概率是大于0.9的，则视为正确的预测，此时将其作为伪标签来使用。2. 帮第一次的伪标签扔进去训练集一起训练后，得到新的模型，按相同的规则再次挑一次伪标签出来。3. 如此不断循环多次，置信度的阈值可以适当作调整。
- **mean teacher**，在这里给涛哥在Recursion Cellular Image Classification第三名的方案做个广告，end2end semi-supervised learning pipeline。

End2end semi-supervised learning pipeline -- Dual Xception



- 1139 classification; 5 fold cross validation; Finetuned on 4 cell types;
- Mean teachers with online prior pseudo-labeling; The pseudo labels updates using the highest validation score teacher models during training;
- Xception above is from imagenet pretrained weights; Xception Large below is trained from scratch;

from seutao

- **knowledge distillation (知识蒸馏)**，此方法有助于提高小模型 (student) 的性能，将大模型 (teacher) 的预测作为soft label (用于学习teacher的模型信息) 与truth (hard label) 扔进去给小模型一起学习，当然两个不同label的loss权重需要调一调。当然，蒸馏的方法有很多种，这只是其中一种最简单的方法。蒸馏不一定用于训练小模型，大模型之间也是可以一同使用的。

数据增强与预处理

数据增强往往都是调出来的，可以先在本地里对图像施加不同的变换方式，用肉眼观察其是否有效（肉眼观察和模型学习到的不一定对等），之后再扔进去网络里训练验证其是否有效。

1. **h/v flip(水平垂直翻转)**，95%的情况下都是有效的，因为不怎么破坏图像空间信息。
2. **random crop/center crop and resize**，在原图进行crop之后再resize到指定的尺度。模型的感受野有限，有时会看不到图像中一些分布比较边缘或者是面积比较小的目标物体，crop过后其占比有更大，模型看到的机会也会更多。适用性也是比较大的。
3. **random cutout/erasing(随机擦除)**，其实就是为了随机擦除图像中局部特征，模型根据有限的特征也可以判断出其属性，可提高模型的泛化性。
4. **AutoAugment**，自己设定一些规则policy，让模型自己寻找合适的数据增强方式，需要消耗比较大的计算资源。
5. **mixup** 一种与数据无关的数据增强方式，即特征之间的线性插值应导致相关标签之间的线性插值，扩大训练分布。意思是两个不同的label的样本进行不同比例的线性插值融合，那么其label也应该是相同比例关系进行线性融合。（上图）
6. **Class balance** 主要就是针对数据不平衡的情况下进行的操作，一般是针对采样方法，或者在loss上做处理，如focal loss、weighted loss等。
7. **图像预处理**，许多看似有效的预处理操作，但是并不一定有效，如在医学领域的图像，许多肉眼观察良好的预处理的方式，实际上是破坏了原图真实类别关联的特征，这种方面需要相关领域知识。

如何选择更好的backbone模型

1. 对于baseline，从**resnet18/34 or efficientnet-B0**起步，把所有work的技巧（loss/augmentation/metric/lr_schedule）调好之后，这时就应该大模型（deeper）；
2. 当更好需要换模型的时候，是不是就需要自己去设计/构造新模型呢？其实在比赛的短期里，重新设计一个新的backbone出来是不提倡的，因为模型不仅要work，还要重新在imagenet上预训练，时间消耗巨大，不合适比赛；
3. 由于学术界里，sota模型多之又多，那如何选择？从个人经验总结来看，比较推荐**se-resnext50/101**、**SENet154**（太大，很少用），带有se block的resnet，都不同类型的任务都有一定的通用性，性价比也较高；efficientnet系列（最近在某些比赛里还优于se-resnext）可以上到B3,B5，有条件的B7都没问题。其他的sota模型，可以尝试Xception，inception-resnetV2等等。

建议与劝退

1. 不要过拟合public lb，要在oof上看cv。一般来说，oof cv是比较准的，除非是遇到了private、public、train数据分布都不一致的必赛，这时就需要靠人品。。
2. 如果是小白入坑，一定要自己独立去完成一个比赛，尽量地去上分，从kernel，讨论区去挖掘有用的信息，如果你没一个比较好的基础名次，自己也不会有资本去和前排的大神一起组队。
3. 劝退建议，希望大家都是在以提高自身能力/兴趣浓厚的前提下来kaggle参加比赛，不要太功利性：如出发点为了找工作想来拿牌、申请好学校offer、带队拿牌并收取佣金和各种py交易来



一起拿奖牌等等，当然前面两种无可厚非，在kaggle拿金确实是有一定的加分。我这里只是强调出发点。

4. 没有坚持的魄力，如遇到一些代码上的难题（debug不通）或者无法上分，就直接放弃比赛或者是让自己变成伸手党等等，比赛期间是一个非常好的学习的过程，而且kaggle的讨论区也是非常热闹的，只要脸皮够厚，在讨论区发一下自己相关困惑的问题（只要不是非常SB的问题），一般都有人很热心的回答你。
5. 充足的比赛时间，学生和上班族基本是一样的，白天上班上课，晚上才能有一些业余的时间来放在比赛里。我还在校时，经常在课堂里拿着ipad观察讨论区，看相关论文，也是为了能在白天积攒一些点子，晚上可以实施。我经常会在比赛的后半段期，会在凌晨调好闹钟起床，观察刚刚运行完程序的log，然后思考再做对应的修改继续跑下一次，为了就是不想浪费时间（当然可能比不上工作时加班到半夜的小伙伴）。当然不提倡，休息还是很重要的。
6. 如果不是很满足这些点，建议不要参加比赛哈，不要浪费这些时间，老老实实搬砖或者在校搞论文会更加好。

总结

1. 以上分享的应该是不够全的，比如一些视频类/目标检测类的技巧就没有出现，只是因为那些方面我还比较弱，所以就没写上去。
2. 以上分享的tricks不一定work，只是一些常用的方法，最重要的还是对数据的理解，针对这些问题来使用相对应的方法，会更加的高效同时也会得到更加多的收益。当然，这是需要长久下来积累的经验与功力。
3. 有兴趣有问题的同学，可以私信我或者留下评论，互相学习。

编辑于 01-27

