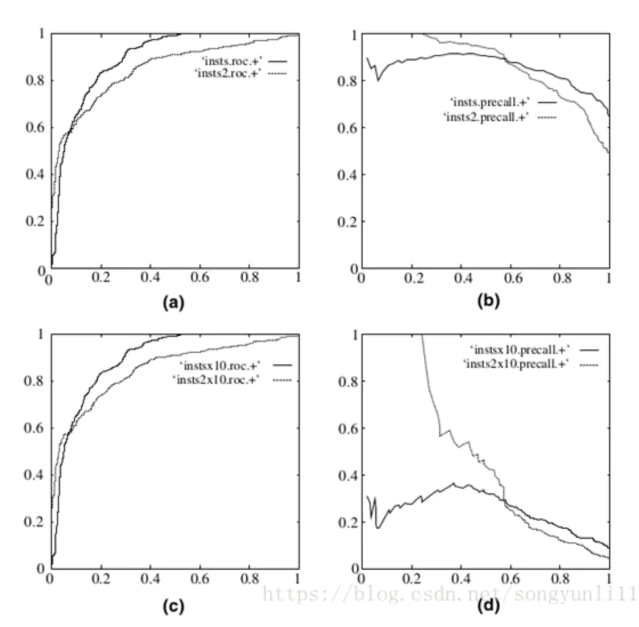
The two error functions that are most commonly used are the [Mean Squared Error (MSE)](https://en.wikipedia.org/wiki/Mean_squared_error) (usually used in regression problems) and the [cross entropy](https://www.ics.uci.edu/~pjsadows/notes.pdf) (usually used in classification problems).

若将负样本的数量增加，扩大个10倍，可以预见FP,TN都会增加，必然会影响到P,R。但ROC曲线的俩个值，FPR只考虑第二行，N若增大10倍，则FP,TN也会成比例增加，并不影响其值，TPR更是只考虑第一行，不会受到影响。这里在网上盗个图【3】，方便大家理解哈～

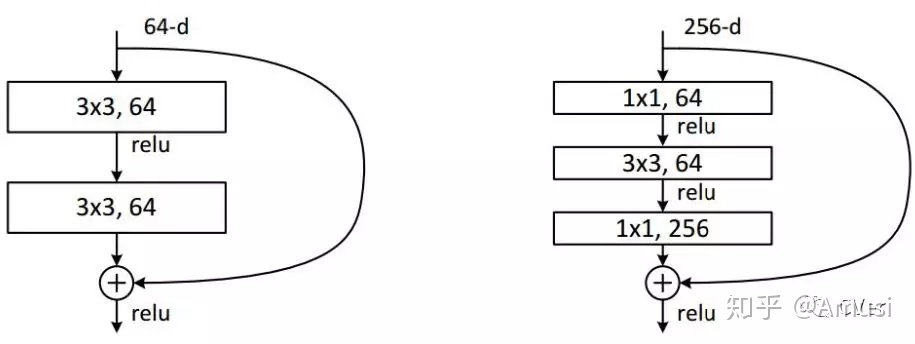


ROC曲线和P-R曲线对比图

其中第一行ab均为原数据的图，左边为ROC曲线，右边为P-R曲线。第二行cd为负样本增大10倍后俩个曲线的图。可以看出，ROC曲线基本没有变化，但P-R曲线确剧烈震荡。因此，在面对正负样本数量不均衡的场景下，ROC曲线（AUC的值）会是一个更加稳定更加全面能反映模型好坏的指标。

AlexNet使得深度学习开始被应用到各种任务。

2014年后，模型size的增大成为提高模型性能的主要研究方法（VGG、GoogLeNet）。尽管VGG取得了很好的性能，但它的计算量太大；GoogLeNet的Inception架构能够在strict constraints on memory and computational budget的情况下，取得很好的性能。另外GoogLeNet仅有5 million参数（1/12 AlexNet参数量）。VGG的参数量为AlexNet的3倍。Inception的计算量比VGG少，但有更高性能。这使得能在大数据（inference时间要少）或移动环境（内存、计算力有限）的情景下使用Inception。



CNN的结构虽然经常层数变多了，但是在性能相对一致的情况下，其实所需的参数是大大减少了的。以层数换参数的策略。结构的复杂化，不意味着参数的必然增加。 通常，光是Fully connected layer的参数量就要占到总体参数量的80%。

CNN : Normal weights

FC+ReLU : Kaiming weights

Batch normal : Uni weights

### Pro Tips

Here are some suggestions and hyperparameters to reduce training time and improve accuracy:

**Reducing learning rate in order to get lower training errors:**

* Trying learning rates of 0.02, 0.01, 0.005. For me, 0.001 does noticeably reduce training errors.
* Use [Adaptive Learning Rate method](https://wiki.tum.de/display/lfdv/Adaptive+Learning+Rate+Method) to learn fast at the beginning and slow down when plateau.

**Improving learning time and learning performance:**

* Cutting half of the filters of CONV layers, and half again, and learning time decreases while learning performance is not affected
* Replacing some FC layers with 1 more CONV layer for more efficient learning. (Need to change input image size to 256x256 for this to work.)
* Using [BatchNorm2d](https://discuss.pytorch.org/t/example-on-how-to-use-batch-norm/216) for faster convergence.
* Increasing [Dropout](https://medium.com/@zhang_yang/scaling-in-neural-network-dropout-layers-with-pytorch-code-example-11436098d426) probability from None to 0.6 to reduce overfitting, since training errors can go down below 1.0.
* Further reducing sizes of FC layers to reduce overfitting.
* Adding an augmented dataset to help avoid overfitting and improve learning performance.

像做菜一样，同一个data set，同一个model structure，不同的人做出来的结果也不一样。因为很多hyperparameter的选择，还有各种细节的处理。

Selecting the optimal number of layers is relatively straight forward. As [@Yoshua-Bengio](https://www.quora.com/profile/Yoshua-Bengio) mentioned on Quora - “You just keep on adding layers, until the test error doesn’t improve anymore”.

How about对grid search里的参数做微分BP，然后找出正确的方向，得到optimal parameters？

Try to avoid early stopping, but instead use regularization (especially dropout) to allow training for as many epochs as we want. Early stopping is the second choice.

*Learning rates for transfer learning*.

It’s common to use a smaller learning rate for ConvNet weights that are being fine-tuned, in comparison to the (randomly-initialized) weights for the new linear classifier that computes the class scores of your new dataset. This is because we expect that the ConvNet weights are relatively good, so we don’t wish to distort them too quickly and too much (especially while the new Linear Classifier above them is being trained from random initialization).

# Benefits of Batch Normalization

Batch normalization optimizes network training. It has been shown to have several benefits:

1. **Networks train faster** – Each training iteration will actually be slower because of the extra calculations during the forward pass and the additional hyperparameters to train during back propagation. However, it should converge much more quickly, so training should be faster overall. More iteration time but less iereation epochs.
2. **Allows higher learning rates** – Gradient descent usually requires small learning rates for the network to converge. And as networks get deeper, their gradients get smaller during back propagation so they require even more iterations. Using batch normalization allows us to use much higher learning rates, which further increases the speed at which networks train.
3. **Makes weights easier to initialize** – Weight initialization can be difficult, and it's even more difficult when creating deeper networks. Batch normalization seems to allow us to be much less careful about choosing our initial starting weights.
4. **Makes more activation functions viable** – Some activation functions do not work well in some situations. Sigmoids lose their gradient pretty quickly, which means they can't be used in deep networks. And ReLUs often die out during training, where they stop learning completely, so we need to be careful about the range of values fed into them. Because batch normalization regulates the values going into each activation function, non-linearlities that don't seem to work well in deep networks actually become viable again.
5. **Simplifies the creation of deeper networks** – Because of the first 4 items listed above, it is easier to build and faster to train deeper neural networks when using batch normalization. And it's been shown that deeper networks generally produce better results, so that's great.
6. **Provides a bit of regularization** – Batch normalization adds a little noise to your network. In some cases, such as in Inception modules, batch normalization has been shown to work as well as dropout. But in general, consider batch normalization as a bit of extra regularization, possibly allowing you to reduce some of the dropout you might add to a network.
7. **May give better results overall** – Some tests seem to show batch normalization actually improves the training results. However, it's really an optimization to help train faster, so you shouldn't think of it as a way to make your network better. But since it lets you train networks faster, that means you can iterate over more designs more quickly. It also lets you build deeper networks, which are usually better. So when you factor in everything, you're probably going to end up with better results if you build your networks with batch normalization.

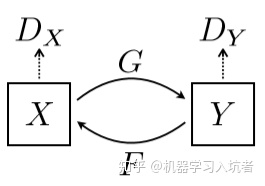
加入人工选择数据环节，去掉不合适的数据，使得训练的数据分布和未来应用时的数据分布更一致。（这个过程可否也通过机器学习来实现？）

股票历史数据也可以通过data augmentation来增强，因为历史股票路径仅仅是众多历史可能性中的一种，就像猫猫狗狗的图片中，猫猫狗狗的姿势也只是它们众多姿势中的一种一样。所以通过变换角度可以增加猫猫狗狗的图片，股票数据也可以。

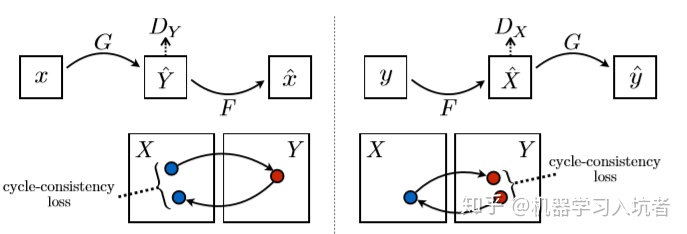


***3、cycleGAN的核心原理***

前面已经提到，cycleGAN将斑马视为一个域X，将普通马视为域Y。此外，论文中X到Y的映射用函数G：X->Y表示；Y到X的映射用 F：Y->X表示。



G和F为生成对抗网络GAN的两个生成器，同时G和F分别对应两个判别器DX和DY。DX用于区分其输入图像来自真实图像X还是F(X)；DY用于区分其输入图像来自真实图像Y还是G(X)。



cycleGAN损失函数包含两个部分，第一部分是经典GAN网络包含的对抗损失adversarial loss，第二部分是论文提出的“cycle consistency loss”，中文译为循环一致性损失，对应上图中x映射到y^再映射回x^的过程，也就是重构损失。

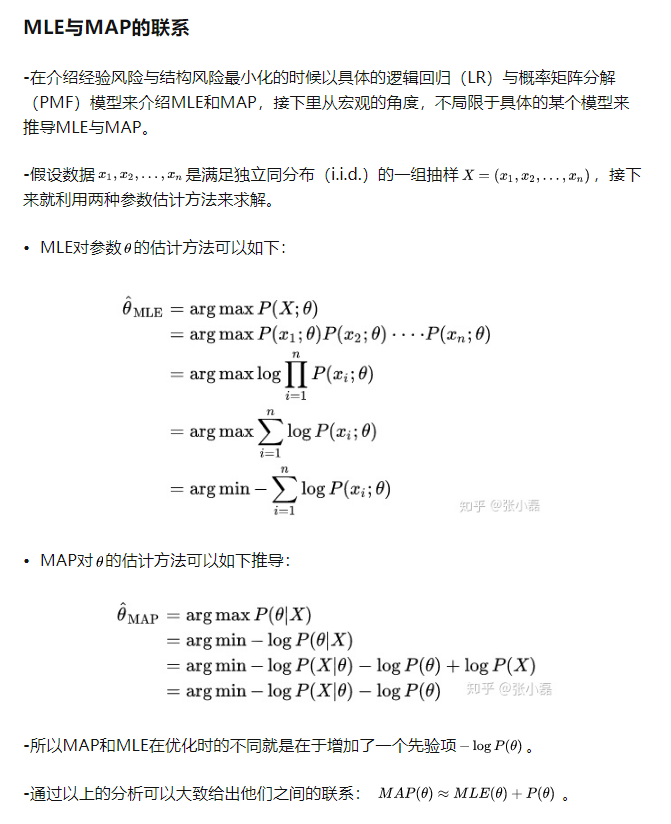
1. 没有证据表明BN的有效性 ，是因为减少了**ICS（Interval Covariate Shift）**。
2. BN work的根本原因，是因为在网络的训练阶段，其能够让优化空间（**optimization landscape**）变的平滑。
3. 其他的normalization技术也能够像BN那样对于网络的训练起到作用。

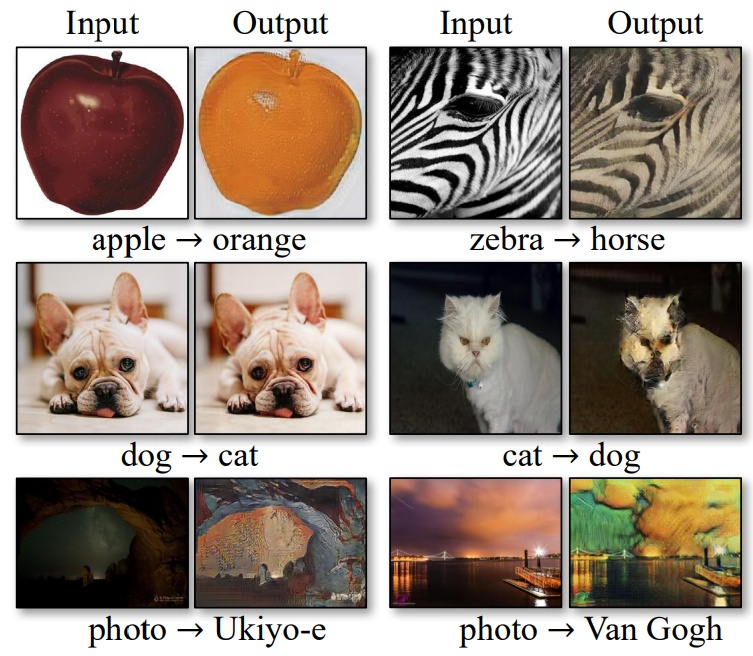
Ioffe and Szegedy在[文章](https://link.zhihu.com/?target=https%3A//arxiv.org/abs/1805.11604)中说，**BN可以防止梯度爆炸或弥散、可以提高训练时模型对于不同超参（学习率、初始化）的鲁棒性、可以让大部分的激活函数能够远离其饱和区域**。所有这些BN的性质，都可以帮助我们快速鲁棒的训练网络。但是该怎么解释呢？

可以理解为极大似然估计和最大后验概率估计得出来的差别；极大似然推倒出来的就是经验风险最小化，最大后验推出来的是结构风险最小化，当数据量很大时候，这两种的结果是一样的。

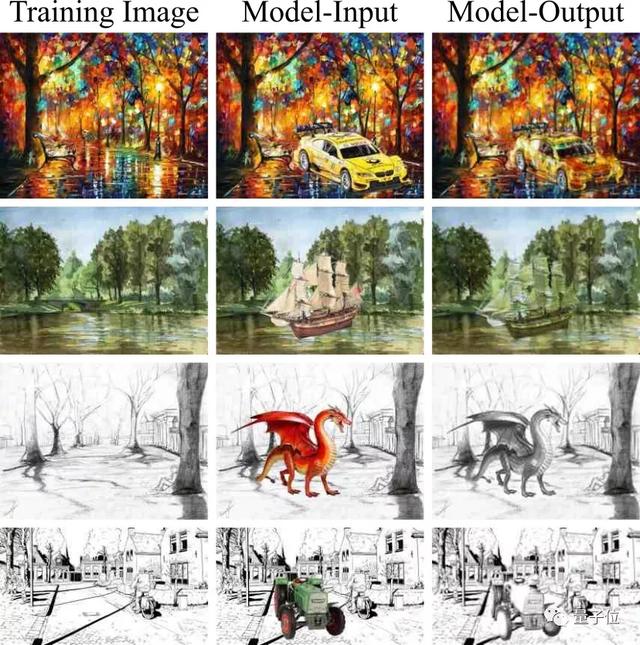
在贝叶斯[统计学](https://baike.baidu.com/item/%E7%BB%9F%E8%AE%A1%E5%AD%A6/1175)中，最大后验（Maximum A Posteriori，MAP）[估计](https://baike.baidu.com/item/%E4%BC%B0%E8%AE%A1/8678215)可以利用经验数据获得对未观测量的点态估计。它与[Fisher](https://baike.baidu.com/item/Fisher)的[最大似然估计](https://baike.baidu.com/item/%E6%9C%80%E5%A4%A7%E4%BC%BC%E7%84%B6%E4%BC%B0%E8%AE%A1/4967925" \t "_blank)（Maximum Likelihood，ML）方法相近，不同的是它扩充了优化的目标函数，其中融合了预估计量的[先验分布](https://baike.baidu.com/item/%E5%85%88%E9%AA%8C%E5%88%86%E5%B8%83/7513047" \t "_blank)信息，所以最大后验估计可以看作是正则化（regularized）的最大似然估计。

经验风险最小化与结构风险最小化是对于损失函数而言的。可以说经验风险最小化只侧重训练数据集上的损失降到最低；而结构风险最小化是在经验风险最小化的基础上约束模型的复杂度，使其在训练数据集的损失降到最低的同时，模型不至于过于复杂，相当于在损失函数上增加了正则项，防止模型出现过拟合状态。这一点也符合奥卡姆剃刀原则：如无必要，勿增实体。





CycleGan猫 → 狗 ，苹果 → 橙子 ，形状的变换不足



可以把人物加到画面里--ConSinGAN

AI or robots may be more emotional than human being.

In a number of papers, such as the one presenting SAGAN (or Self Attention GAN), it is shown that Spectral Normalization, a particular kind of normalization applied on the **convolutional kernels**, can greatly help the stability of the training.

## 1.数据增强

最明显的应用是训练模型：从我们的数据生成新样本以增强我们的数据集。我们如何检查这种增强是否真的有帮助呢？有两个主要策略：我们可以在“假”数据上训练我们的模型，并检查它在真实样本上的表现。对应的我们在实际数据上训练我们的模型来做一些分类任务，并且检查它对生成的数据的执行情况。如果它在两种情况下都能正常工作，你可以随意将生成模型中的样本添加到你的实际数据中并再次重新训练，你应该期望获得性能。

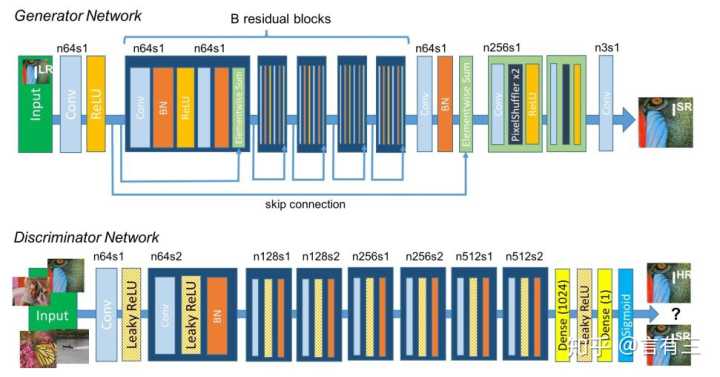
NVIDIA展示了这种方法的[惊人实例](https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1803.01229.pdf)：他们使用GAN来增加具有不同疾病的医学脑CT图像的数据集，并且表明仅使用经典数据的分类性能是78.6％的灵敏度和88.4％的特异性。通过添加合成数据增强，可以增加到85.7％的灵敏度和92.4％的特异性。

图像降噪图像在产生和传输过程中都会受到噪声的干扰，因此图像降噪是一个非常基础的问题，生成式模型GAN在捕捉噪声的分布上有天然的优势。基于深度学习的图像降噪面临的一大难题就是没有成对的真实噪声和无噪声数据，GCBD(GAN-CNN Based Blind Denoiser)方法使用GAN从真实带噪声图像中采集噪声，获得真实的成对图用于降噪模型训练。

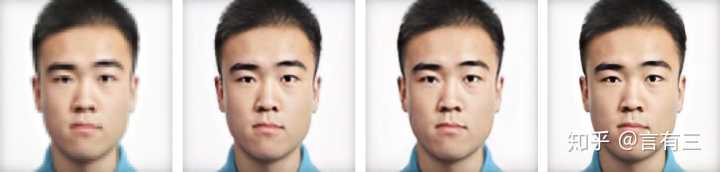
链接：https://www.zhihu.com/question/270901930/answer/1029938028

**图像超分辩**

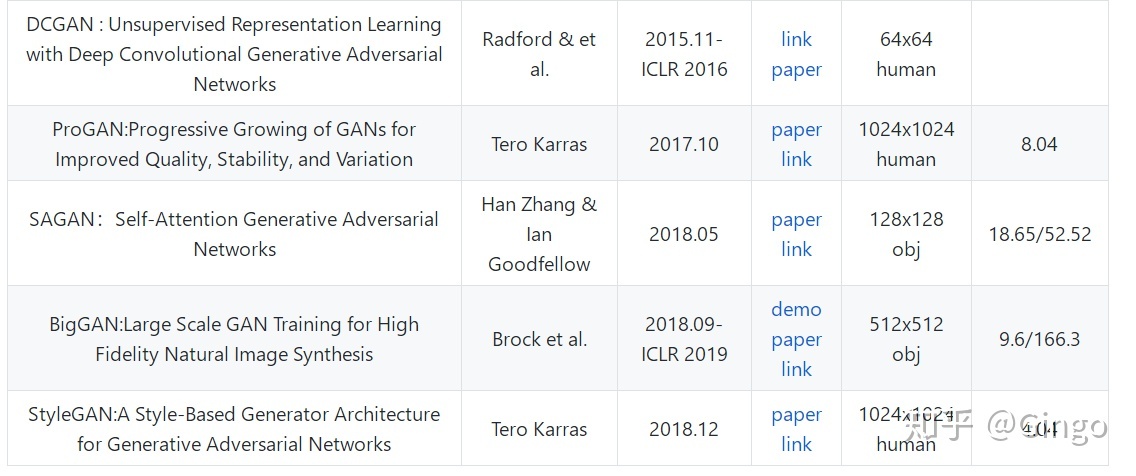
越高的分辨率能获得更清晰的成像，人们对分辨率的追求永无止境。与之同时，很多的低分辨率老照片需要修复，具有很大的人文社会价值，GAN已经在超分辨率领域颇有建树。

SRGAN

SRGAN(super-resolution generativeadversarial network)是第一个基于GAN的图像超分辩模型，能够取得8倍的高分辨率放大。



上图展示了一些结果，感兴趣的读者可以使用自己的数据进行训练。



为了很好的说明问题，研究者们两个指标来对比了实验结果，分别是FID和精度（precision）、召回率（recall）以及两者的平均数F1。

其中FID（Fréchet distance(弗雷歇距离) ）是法国数学家Maurice René Fréchet在1906年提出的一种路径空间相似形描述，直观来说是狗绳距离：主人走路径A，狗走路径B，各自走完这两条路径过程中所需要的最短狗绳长度，所以说，FID与生成图像的质量呈负相关。

