论文解析课

一、模型 Scale

`EfficientNet` 的论文，解决的是如何科学的 `Scale` 模型的问题。

什么叫做模型的 `Scale` ？当卷积模型的 `baseline` 确定后，通过增加网络的「深度」，网络的「宽度」和输入图像的「分辨率」，就能得到不同大小的神经网络。比如 `ResNet18`，`ResNet50` 和 `ResNet101`。

为了帮助大家理解，我们用机器学习中的算法来进行类比。对于 `GBDT` 算法，模型的「整体形式」是确定的（前向加法树模型），通过对 `n_estimators` 以及 `max_depth` 等超参数的选择，可以确定具体的模型「形式」。（超参数包含两类，一类是涉及模型的「形式」，一类是涉及模型的参数训练）

```
from sklearn.ensemble import GradientBoostingClassifier
GradientBoostingClassifier(n_estimators = 100,max_depth=6)
```

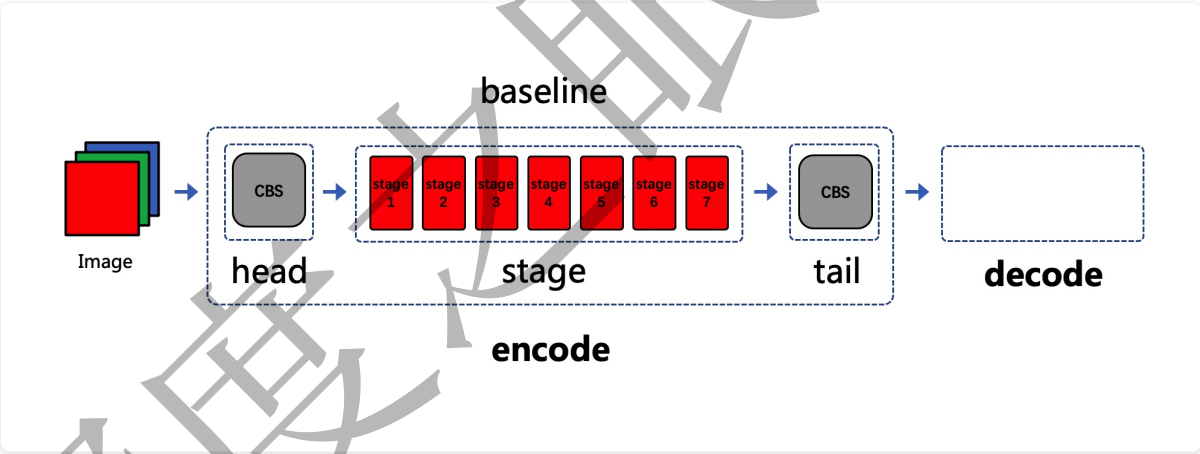
最后，再通过数据，训练得到具体每棵树的「参数」。毫无疑问，`n_estimators` 以及 `max_depth` 等超参数的确定非常的重要，在机器学习中是通过 `gridsearch` 等方法调参得到。

`GBDT` 的「整体形式」对应的就是前面说的卷积模型的 `baseline`。`GBDT` 的 `n_estimators` 和 `max_depth` 等超参数对应的就是前面说的网络的「深度」，网络的「宽度」和输入图像的「分辨率」。

> *`EfficientNet` 的论文提供了一套，针对卷积神经网络调整「超参数」的方法论。（这里超参数是指涉及模型形式的超参数，且 `baseline` 需要已知）*

二、卷积模型抽象

卷积网络模型（分类模型），可以抽象为下面形式：



其中 $stage_i$ 在代码中，经常又叫做 $block_i$。每一个 $stage_i$ 在结构的「形式」上是相同的。但由于「超参数」的不同，导致 $stage_i$ 的具体结构会有变化。

> *网络的「深度」，网络的「宽度」和输入图像的「分辨率」指的是 $stage_i$ 中 `layer` 数量，`channel` 大小，和输入张量的 `size` 大小。*

在论文中，分别用如下单词表示：

- depth
- width
- resolution

将卷积网络模型（分类模型）的 `stage` 部分进行数学抽象：

$$\odot_{i=1\cdots s}\mathcal{F}_i^{L_i}(X_{<H_i,W_i,C_i>}) \tag{1}$$

公式 (1) 中 s 表示有 s 个 `stage` 。$\mathcal{F}_i^{L_i}$ 表示第 i 个 `stage` ，使用的结构 $\mathcal{F}$ 的次数为 L_i 次。(H_i, W_i) 是输入张量的分辨率，C_i 是 `channel` 的大小。当公式 (1) 确定后，就可以调整每个 `stage` 的深度、宽度和分辨率，从而达到 `Scale` 模型的目的。

$$\odot_{i=1\cdots s}\mathcal{F}_i^{d\cdot L_i}(X_{<r\cdot H_i,r\cdot W_i,w\cdot C_i>}) \tag{2}$$

当 $\mathcal{F}_i, H_i, W_i, C_i$ 已知的时候，如何选择合适的 d, r, w 就是 `EfficientNet` 论文的核心点：

经验观察一：
随着网络「深度」，「宽度」和「分辨率」的不断放大，准确率的提升将渐渐消失。

经验观察二：
为了追求更高的准确率和效率，需要平衡，网络「深度」，「宽度」和「分辨率」。

最终结论：

> *同时放大或者缩小三个参数。*

先验的假设具有如下公式：

$$
\begin{aligned}
depth: d &= \alpha^{\phi} \\
width: w &= \beta^{\phi} \\
resolution: r &= \gamma^{\phi} \\
s.t: \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\
\alpha \geq 1, \beta^2 &\geq 1, \gamma \geq 1
\end{aligned}
$$

`s.t` 的约束关系是由于卷积中运算量和三者关系决定的。

$$
cost \approx C_i \times H \times W \times C_o
$$

选取超参数的过程：

- 固定 $\phi = 1$，找到最优的 α, β, γ
- 固定 α, β, γ，不断调整 ϕ 得到不同的 d, w, r，最终就得到了不同的 `EfficientNet-Bi` 模型。

`EfficientNet-B0` 也就是论文中的 `baseline` 模型是一切的起点，当明白了 `EfficientNet-B0` 模型的结构后，乘上对应的 d, w, r 系数，就能 `Scale` 出其余的 `EfficientNet` 模型。

三、EfficientNet-B0

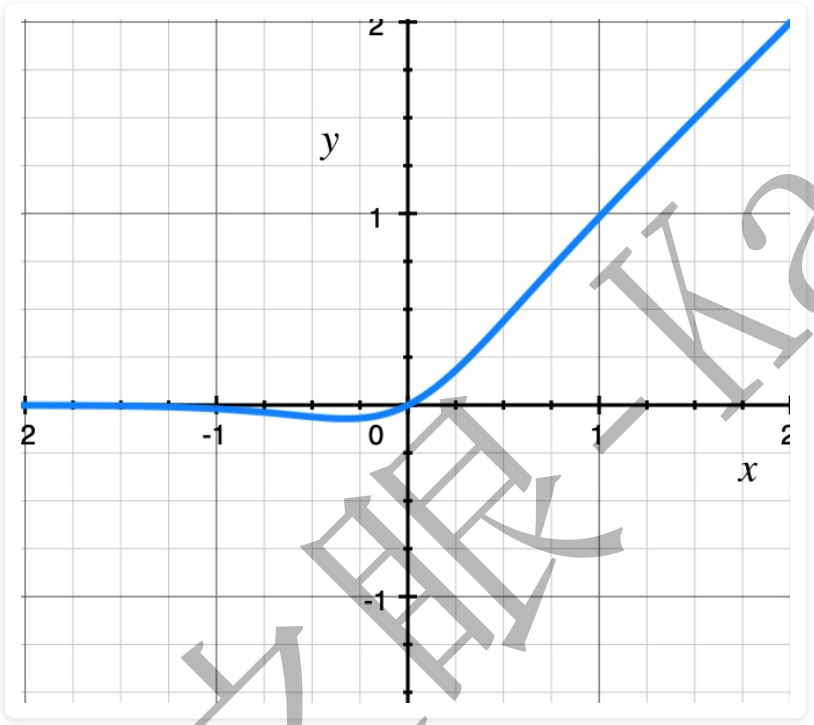
`EfficientNet-B0` 的结构是通过 NAS 技术搜索而出，下面我们来看一下 `EfficientNet-B0` 的结构。了解它的结构之前，需要一些预备知识。

- `Swish`
- `Depthwise convolutions`
- `Channel attention`
- `Skip connection`

Swish:

$$x \cdot sigmoid(\beta x)$$

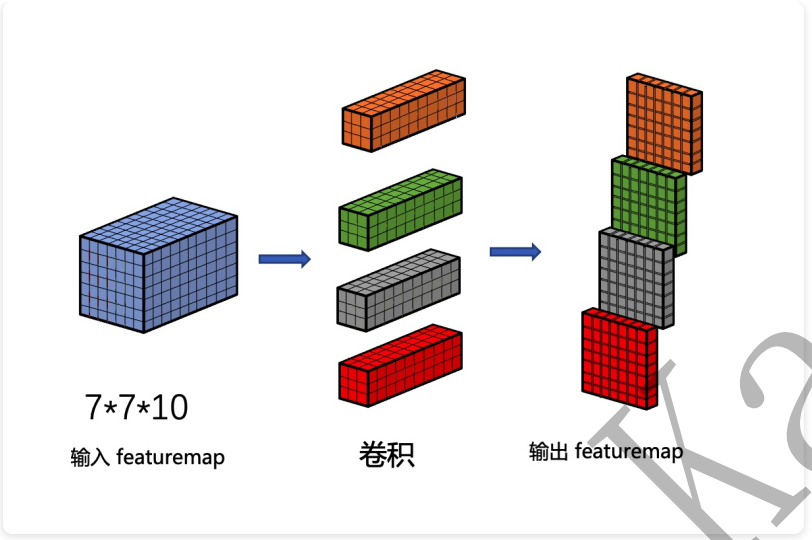
当 $\beta = 0$ 时，`Swish` 近似于线性函数，$beta \to \infty$ 时，近似于 `relu` 。



在 *torch* 中 `Swish` 叫做 `Silu` 。

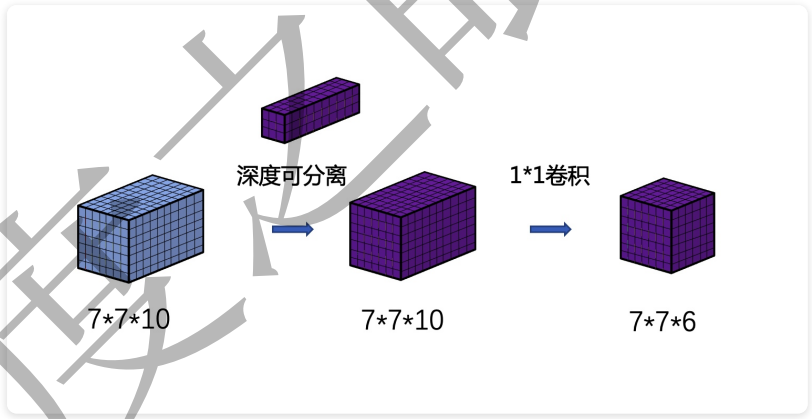
Depthwise convolutions:

深度可分离卷积，目的是为了减少卷积的参数数量。



以上图为例，传统卷积的参数数量为 $3 \times 3 \times 10 \times 4$，需要参数 360。

如果使用深度可分离卷积，则如下图所示：

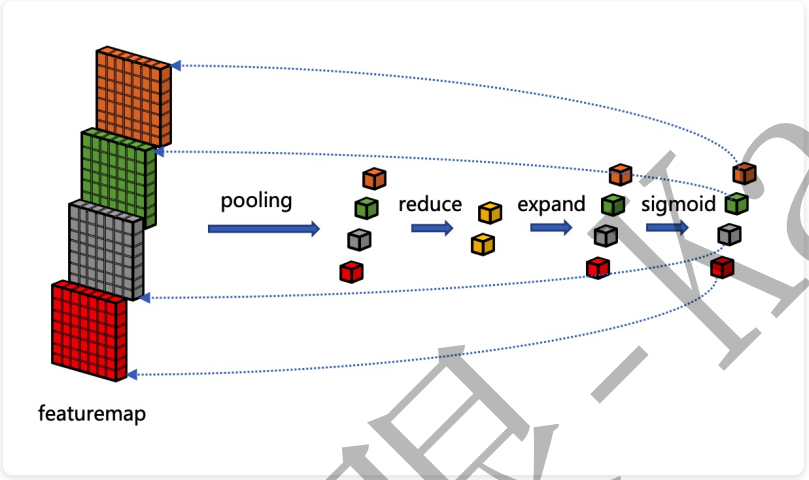


参数数量为 $3 \times 3 \times 10 + 1 \times 1 \times 10 \times 6$，需要 150 个参数。引入 1×1 卷积的目的是为了使得通道间进行交互，并能返回任意的通道数。

(mobilenet)

Channel attention:

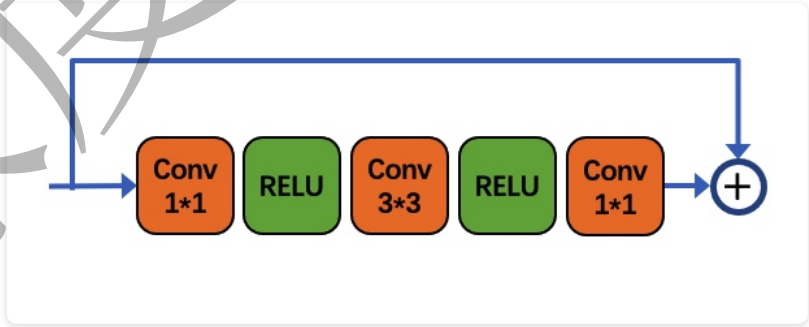
通过卷积输出后的 `featuremap`，每一个 `channel` 都是一视同仁的。很自然的想法是，为什么要一视同仁？可能不同的 `channel` 所蕴含的信息的重要性程度是不一样的。



`pooling` 是为了得到该 `channel` 的「信息」，`reduce` 是为了不同 `channel` 间做交互，知道谁重要，谁不重要，`expand` 是还原回原始的 `channel` 个数。`sigmoid` 是为了归一化。(senet)

Skip connection:

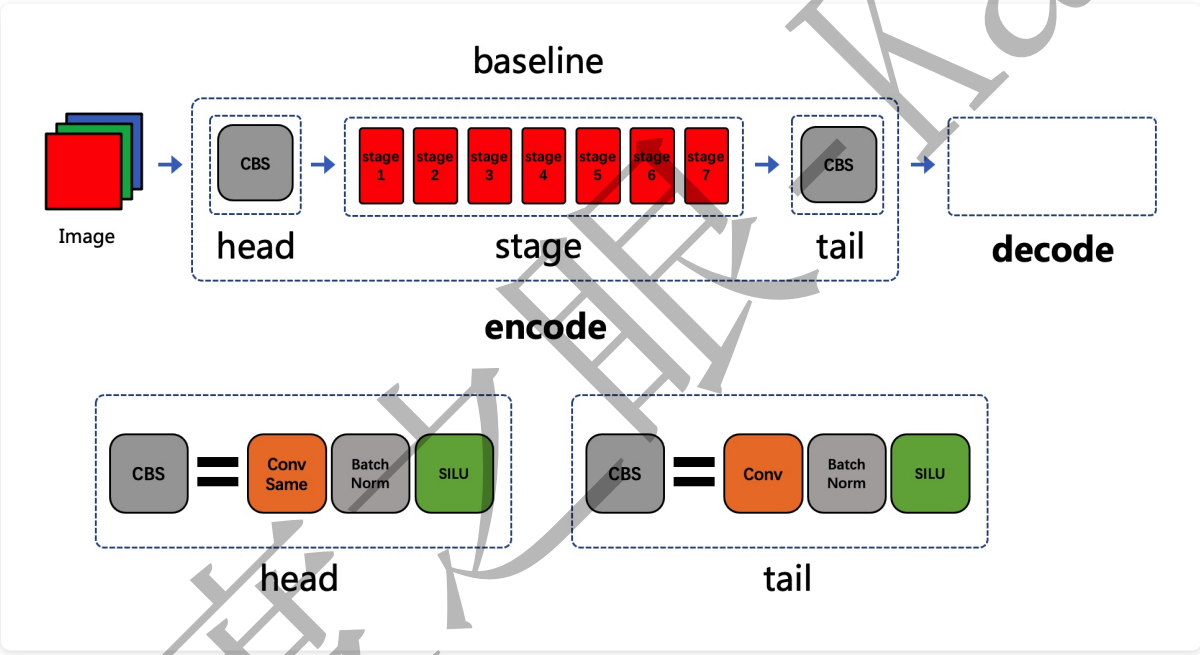
跳层连接，神经网络中最常见的一种操作。



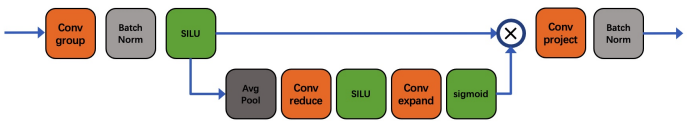
引入两个 1×1 卷积的目的是为了保证输入张量和输出张量的 `channel` 维度一致，这样才能保障一定能做 **加法** 操作。

在 *ResNet* 系列网络中，是先通过 1×1 卷积降维（`channel` 维度），然后再通过 1×1 卷积升维（`channel` 维度）。在 *MobileNet* 网络中，则是先升维，再降维，所以这种残差结构（`Skip connection`）称为 `Inverted Residual`。

准备工作已经就绪，接下来看 `EfficientNet-B0` 的具体结构。

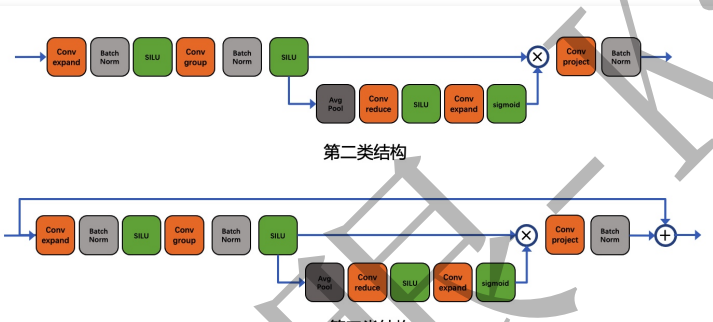


`head` 和 `tail` 如上图所示，`EfficientNet` 有 7 个 *stage*。每个 stage 都是由相同的结构形式根据不同「超参数」组合而成。`EfficientNet` 大概有三类（不考虑 `kernelsize`）。



第一类结构

第一类结构只属于 $stage1$，换句话说，$stage1$ 由若干的第一类结构组成。



第二类结构

第三类结构

其余的 $stagei$ 由第二类结构和第三类结构组成。第二类结构的特点是输入和输出的维度不一样，第二类结构会增加 `channel` 。

第三类结构的特点是输入和输出维度一样，可以做 `Skip connection` 。

第二类和第三类结构都会使用 `Conv expand` 对 `channel` 维度做增加操作。如果有残差连接，那么就是典型的 `Inverted Residual` 结构。

参数配置如下所示：

```
# ksize, input, output, expand, stride, image_size
# layers 1
# MBConv1, k3*3, inputChannels, outputChannels, stride, Resolution
[1, 3, 32, 16, 1, [112, 112]],
```

```
# layers 2
# MBConv6, k3*3, inputChannels, outputChannels, stride, Resolution
[6, 3, 16, 24, 2, [112, 112]],
[6, 3, 24, 24, 1, [56, 56]],

# layers 2
[6, 5, 24, 40, 2, [56, 56]],
[6, 5, 40, 40, 1, [28, 28]],

# layers 3
[6, 3, 40, 80, 2, [28, 28]],
[6, 3, 80, 80, 1, [14, 14]],
[6, 3, 80, 80, 1, [14, 14]],

# layers 3
[6, 5, 80,  112, 1, [14, 14]],
[6, 5, 112, 112, 1, [14, 14]],
[6, 5, 112, 112, 1, [14, 14]],

# layers 4
[6, 5, 112, 192, 2, [14, 14]],
[6, 5, 192, 192, 1, [7, 7]],
[6, 5, 192, 192, 1, [7, 7]],
[6, 5, 192, 192, 1, [7, 7]],

# layers 1
[6, 3, 192, 320, 1, [7, 7]],
```

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

有了 `BaseLine` 之后，通过 d, w, r 系数，就可以拓展到其他的 `EfficientNet` 网络。

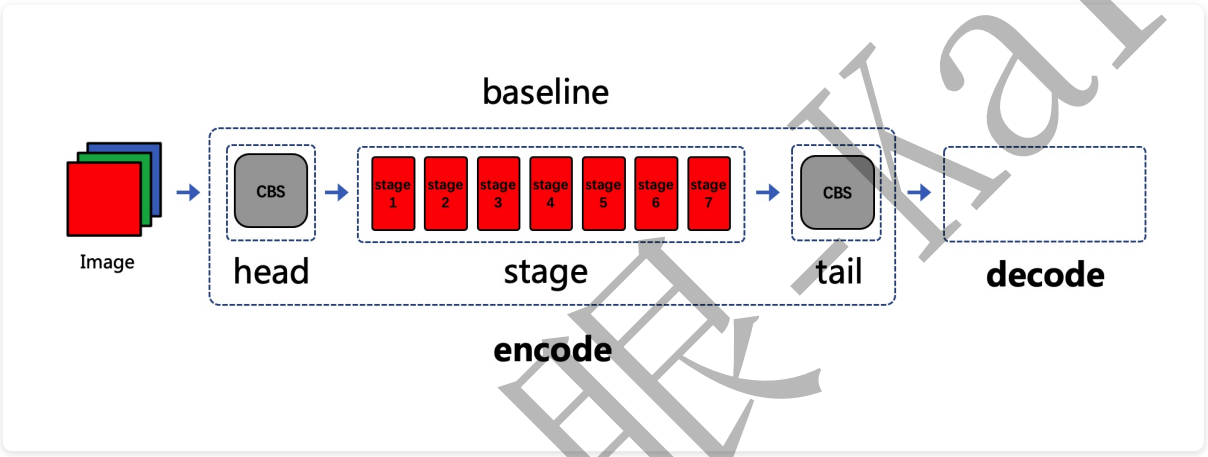
```
# 参数分别表示 width, depth, resolution
'efficientnet-b0': (1.0, 1.0, 224),
'efficientnet-b1': (1.0, 1.1, 240),
'efficientnet-b2': (1.1, 1.2, 260),
'efficientnet-b3': (1.2, 1.4, 300),
'efficientnet-b4': (1.4, 1.8, 380),
'efficientnet-b5': (1.6, 2.2, 456),
'efficientnet-b6': (1.8, 2.6, 528),
'efficientnet-b7': (2.0, 3.1, 600),
```

注意，对于小数，需要向上取整数，下面就以 `EfficientNet-B7` 举例，看一下 `B7` 的网络结构是否是由 `B0` 的网络结构 `Scale` 得到的。

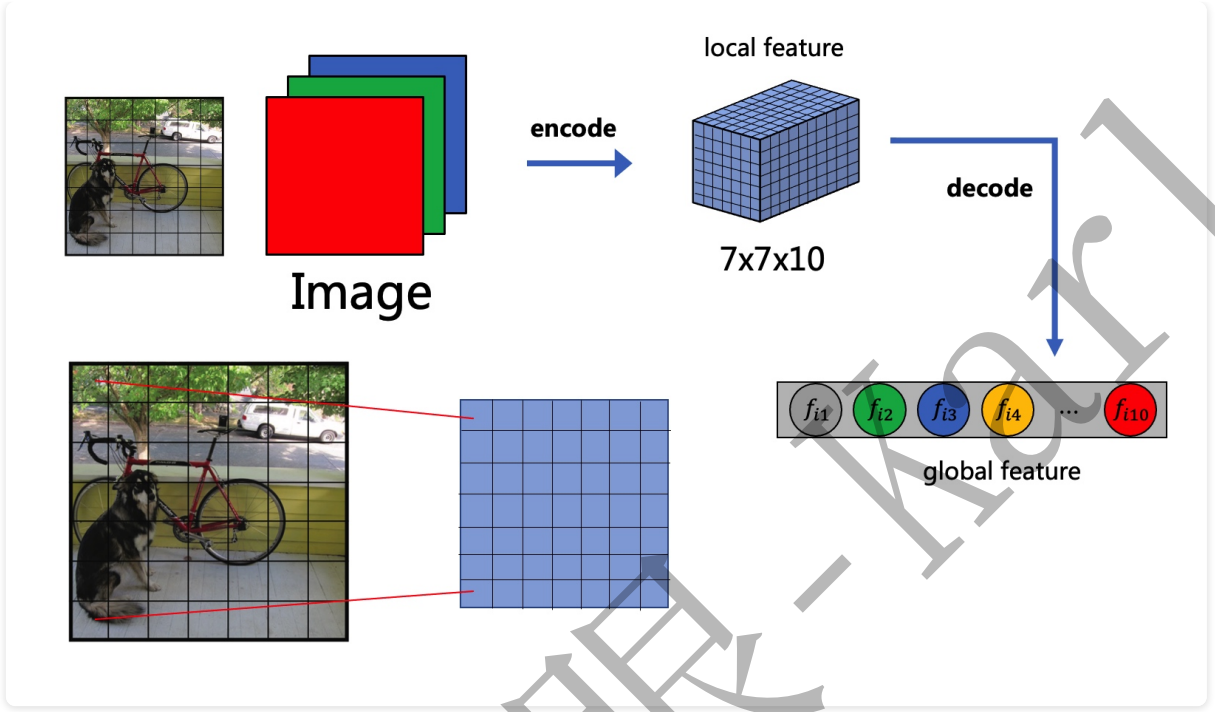
efficientnet-b0代码连接

四、Encode 和 Decode

再回顾一下，卷积模型的一般框架：



现在我们需要来深刻理解一下，`encode`，这样才能知道如何进行`decode`。



每一张图片，经过卷积模型（`Encode`）会输出一张 `featuremap`，这个 `featuremap` 每一个 `channel` 维度的向量都可以理解为对应原图像部分区域的特征向量，即 `local feature`。

最后，我们需要将 `local feature` 「编码」成一个 `feature`，该 `feature` 代表了整张图片。即 `global feature`。

在大部分卷积网络分类模型中，一般使用 `AvgPool` 作为 `Decode` 模块。

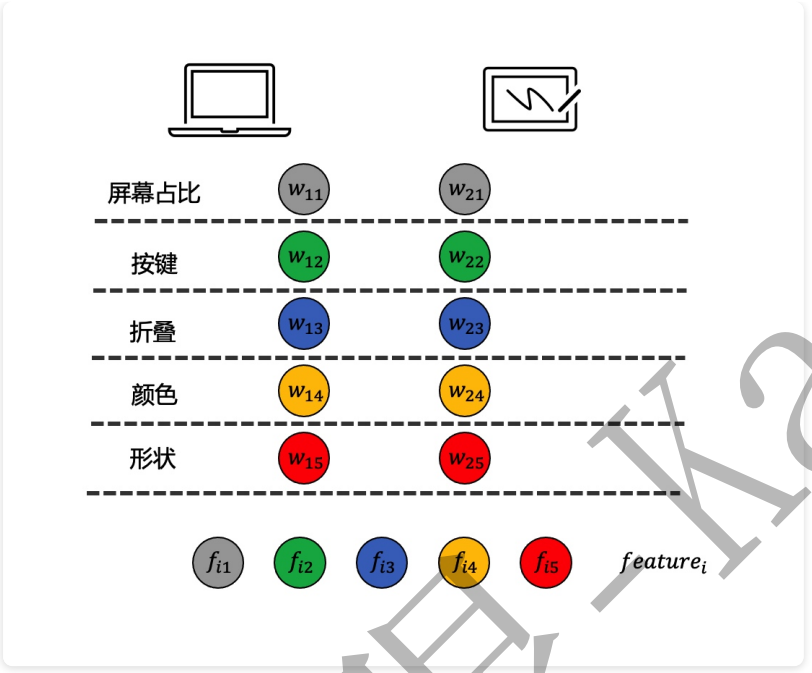
Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

最后 `decode` 的 `feature` 将输入给全连接层进行 `softmax` 分类处理（i.e 逻辑斯特回归模型）。

五、重新审视 softmax

假设有一个二分类图像模型，区分笔记本和平板电脑，最后一层的全连接层参数为 $W = (w^1, w^2), W \in R^{2\times5}$，现在有一张图片，经过模型 `encode` 和 `decode` 后输出

$$\mathbf{f_i} = \begin{pmatrix} f_{i1} \\ f_{i2} \\ \vdots \\ f_{i5} \end{pmatrix}$$



模型预测的时候，即比较 $w_1\mathbf{f_i}$ 和 $w_2\mathbf{f_i}$ 的大小，谁比较大，就预测该图片为哪一个。

所以，$\mathbf{f_i}$ 代表是什么呢？ w_1, w_2 代表的又是什么呢?

w_i 通过硬标签（hard label）端到端的学习出，如果标签，本身蕴含了一些「特征信息」，则可能更利于 w_i 的学习。比如对于笔记本和平板电脑来说，它们本身有一些特征是有交集的，并不是绝对的 1 或 0。笔记本屏幕占比低一些，平板电脑屏幕占比高一些。

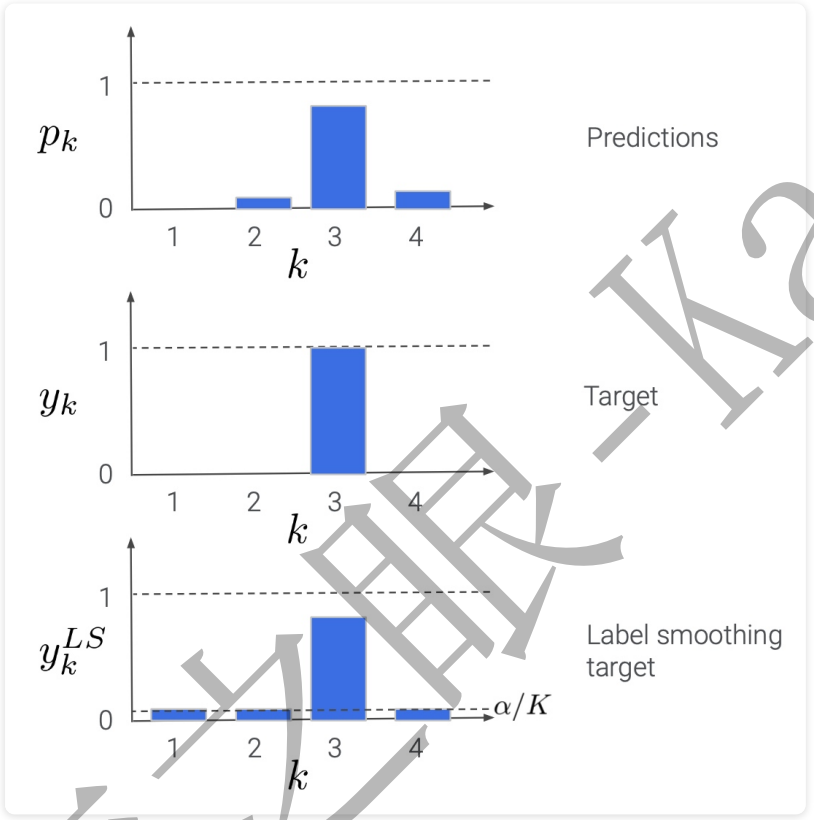
六、软标签

$$y_k^{LS} = y_k(1-\alpha) + \alpha/K \tag{3}$$

其中 K 为类别个数，y_k 为类别标签，α 为平滑系数。

公式 (3) 的构造，能保证如下性质：

$$\sum_{k=1}^{K} y_k^{LS} = 1$$



[该图引用自 google brain]

为什么选择平均呢？也许可以用最大熵原理进行解释。

> *当我们对其余信息一无所知的时候，选择熵最大的一个分布。*

反过来，能不能不选择平均呢？比如让一个 `teacher` 模型告诉我们如何去平滑标签，如何把标签的值部分值分给其他类别。而这可能就是知识蒸馏技术能成功的原因。甚至在后面即将讲到的 `Mixdivdes` 里面也会看到它的影子（refine label）。

七、标签噪声问题

数据的标签会因为各种各样的原因存在一些误差。这称 `label noise` 问题。`label noise` 的问题定义并未完全统一，我们暂时选择如下的定义：

> *标签噪声是指标签被观察到了，但是被标记错误。*

产生标签噪声的原因大致可以归纳如下：

- 信息不充分。
- 专家犯错误。
- 标签偏主观。
- 信息传输误差。

将标签噪声进行统计建模，有三类建模方法，对应三类标签噪声问题。(和缺失值问题类似)

- 完全随机噪声模型。标签噪声和特征向量以及真实标签无关，完全随机。（对称噪声）
- 随机噪声模型。标签噪声和特征向量无关，但和真实标签可能相关。（非对称噪声）
- 非随机噪声模型。标签噪声和特征向量以及真实标签都有关。

举三个例子：

- 登记的地址标签。
- 登记的企业财务情况标签。
- 登记的家庭收入标签。

MixDivides 的核心：

> *用 SSL 的方式来处理 LNL 问题。将高度疑似的噪声样本当做 unlabel 样本，使用 SSL 来处理。*

LNL：learning with noisy label
SSL：semi-supervised learning

LNL：
修正 loss function:

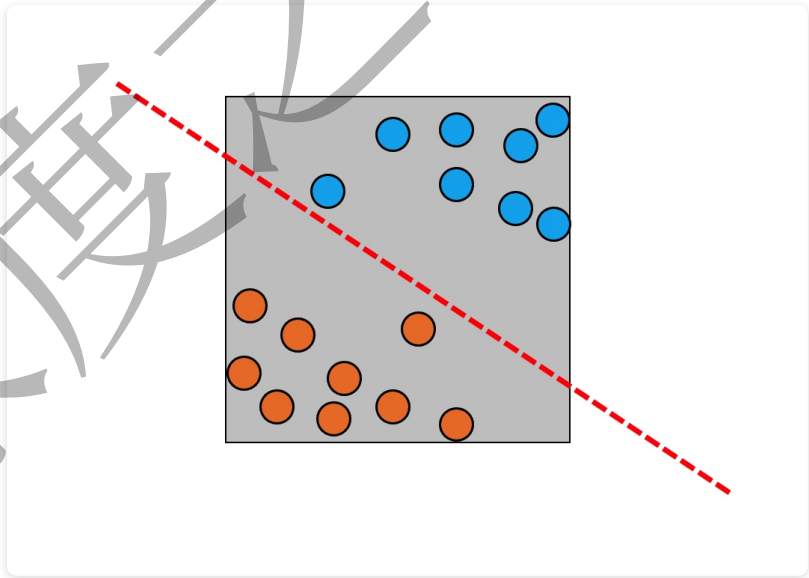
- 样本一视同仁，重新标记噪声样本。
- 样本不再等同，重新赋予样本权重。（极端情况就是分离噪声和干净样本）

SSL：
增加 loss funcion:

- 一致性损失（扰动样本预测一致）
- 熵最小化（减少未标记样本的不稳定性）

核心点：

> *神经网络学习干净样本相比噪声样本会更快。*
> *损失的下降速度：干净样本>非对称噪声>对称噪声。*



如何区分噪声样本和非噪声样本：

在每个 `epoch` ，根据样本的 loss ，利用 `GMM` 模型拟合（2 个高斯分布混合而成）。均值大的是噪声样本，均值小的是干净样本。 `GMM` 模型能返回每一个样本的概率。

热启动：

使用原始数据，训练几个 epoch 得到两个模型。由于可能存在非对称噪声，所以对模型输出的置信度进行抑制。防止过拟合非对称噪声。

$$loss = -y \log p + p \log p$$

训练得到 $\theta^{(1)}, \theta^{(2)}$ 模型。

Algorithm 1: DivideMix. Line 4-8: co-divide; Line 17-18: label co-refinement; Line 20: label co-guessing.

Input: $\theta^{(1)}$ and $\theta^{(2)}$, training dataset $(\mathcal{X}, \mathcal{Y})$, clean probability threshold τ, number of augmentations M, sharpening temperature T, unsupervised loss weight λ_u, Beta distribution parameter α for MixMatch.

$\theta^{(1)}, \theta^{(2)} = \mathrm{WarmUp}(\mathcal{X}, \mathcal{Y}, \theta^{(1)}, \theta^{(2)})$ *// standard training (with confidence penalty)*

while $e < \mathrm{MaxEpoch}$ **do**

  $\mathcal{W}^{(2)} = \mathrm{GMM}(\mathcal{X}, \mathcal{Y}, \theta^{(1)})$ *// model per-sample loss with $\theta^{(1)}$ to obtain clean proabability for $\theta^{(2)}$*

  $\mathcal{W}^{(1)} = \mathrm{GMM}(\mathcal{X}, \mathcal{Y}, \theta^{(2)})$ *// model per-sample loss with $\theta^{(2)}$ to obtain clean proabability for $\theta^{(1)}$*

  for $k = 1, 2$ **do** *// train the two networks one by one*

    $\mathcal{X}_e^{(k)} = \{(x_i, y_i, w_i) | w_i \geq \tau, \forall (x_i, y_i, w_i) \in (\mathcal{X}, \mathcal{Y}, \mathcal{W}^{(k)})\}$ *// labeled training set for $\theta^{(k)}$*

    $\mathcal{U}_e^{(k)} = \{x_i | w_i < \tau, \forall (x_i, w_i) \in (\mathcal{X}, \mathcal{W}^{(k)})\}$ *// unlabeled training set for $\theta^{(k)}$*

    for iter $= 1$ **to** num_iters **do**

      From $\mathcal{X}_e^{(k)}$, draw a mini-batch $\{(x_b, y_b, w_b); b \in (1, ..., B)\}$

      From $\mathcal{U}_e^{(k)}$, draw a mini-batch $\{u_b; b \in (1, ..., B)\}$

      for $b = 1$ **to** B **do**

        for $m = 1$ **to** M **do**

          $\hat{x}_{b,m} = \mathrm{Augment}(x_b)$ *// apply m^{th} round of augmentation to x_b*

          $\hat{u}_{b,m} = \mathrm{Augment}(u_b)$ *// apply m^{th} round of augmentation to u_b*

        end

        $p_b = \frac{1}{M} \sum_m \mathrm{p}_{\mathrm{model}}(\hat{x}_{b,m}; \theta^{(k)})$ *// average the predictions across augmentations of x_b*

        $\bar{y}_b = w_b y_b + (1 - w_b) p_b$

        // refine ground-truth label guided by the clean probability produced by the other network

        $\hat{y}_b = \mathrm{Sharpen}(\bar{y}_b, T)$ *// apply temperature sharpening to the refined label*

        $\bar{q}_b = \frac{1}{2M} \sum_m \left(\mathrm{p}_{\mathrm{model}}(\hat{u}_{b,m}; \theta^{(1)}) + \mathrm{p}_{\mathrm{model}}(\hat{u}_{b,m}; \theta^{(2)})\right)$

        // co-guessing: average the predictions from both networks across augmentations of u_b

        $q_b = \mathrm{Sharpen}(\bar{q}_b, T)$ *// apply temperature sharpening to the guessed label*

      end

      $\hat{\mathcal{X}} = \{(\hat{x}_{b,m}, \hat{y}_b); b \in (1, ..., B), m \in (1, ..., M)\}$ *// augmented labeled mini-batch*

      $\hat{\mathcal{U}} = \{(\hat{u}_{b,m}, q_b); b \in (1, ..., B), m \in (1, ..., M)\}$ *// augmented unlabeled mini-batch*

      $\mathcal{L}_{\mathcal{X}}, \mathcal{L}_{\mathcal{U}} = \mathrm{MixMatch}(\hat{\mathcal{X}}, \hat{\mathcal{U}})$ *// apply MixMatch*

      $\mathcal{L} = \mathcal{L}_{\mathcal{X}} + \lambda_u \mathcal{L}_{\mathcal{U}} + \lambda_r \mathcal{L}_{\mathrm{reg}}$ *// total loss*

      $\theta^{(k)} = \mathrm{SGD}(\mathcal{L}, \theta^{(k)})$ *// update model parameters*

    end

  end

end

$$
\begin{aligned}
&\lambda \sim \mathrm{Beta}(\alpha, \alpha),\\
&\lambda' = \max(\lambda, 1-\lambda),\\
&x' = \lambda' x_1 + (1-\lambda') x_2,\\
&p' = \lambda' p_1 + (1-\lambda') p_2.
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{L}_{\mathcal{X}} &= -\frac{1}{|\mathcal{X}'|} \sum_{x,p \in \mathcal{X}'} \sum_{c} p_c \log(\mathrm{p}^{\mathrm{c}}_{\mathrm{model}}(x;\theta)),\\
\mathcal{L}_{\mathcal{U}} &= \frac{1}{|\mathcal{U}'|} \sum_{x,p \in \mathcal{U}'} \left\| p - \mathrm{p}_{\mathrm{model}}(x;\theta) \right\|_2^2 .
\end{aligned}
$$

$$
\mathcal{L}_{\mathrm{reg}} = \sum_{c} \pi_c \log\left(\pi_c \Big/ \frac{1}{|\mathcal{X}'| + |\mathcal{U}'|} \sum_{x \in \mathcal{X}' + \mathcal{U}'} \mathrm{p}^{\mathrm{c}}_{\mathrm{model}}(x;\theta) \right).
$$