```python
import os
import torch
import torch.nn as nn
import numpy as np
import math
import PIL


def dir_exists(path):
    """
    判断是否存在输入path，如果不存在创建
    """
    if not os.path.exists(path):
        os.makedirs(path)


def initialize_weights(*models):
    """
    初始化模型的weights
    """
    for model in models:
        for m in model.modules():
            if isinstance(m, nn.Conv2d):
                # _calculate_correct_fan(tensor, mode)是算出input和output feature
                # map的元素总数
                # Fills the input Tensor with values according to the method
                # described in Delving deep into rectifiers: Surpassing human-
                # level performance on ImageNet classification - He, K. et al.
                # (2015), using a normal distribution. The resulting tensor will
                # have values sampled from \mathcal{N}(0, \text{std}^2)N(0,std2)
                # where 公式在底部

                nn.init.kaiming_normal_(m.weight.data, nonlinearity='relu')
            elif isinstance(m, nn.BatchNorm2d):
                # Fills self tensor with the specified value.
                # 把weight初始化为1，
                m.weight.data.fill_(1.)
                m.bias.data.fill_(1e-4)
            elif isinstance(m, nn.Linear):
                m.weight.data.normal_(0.0, 0.0001)
                m.bias.data.zero_()


def get_upsampling_weight(in_channels, out_channels, kernel_size):
    """
    得到upsampling 的weight
    """
    factor = (kernel_size + 1) // 2
    if kernel_size % 2 == 1:
        center = factor - 1
    else:
        center = factor - 0.5
```

```
    # ogrid函数作为产生numpy数组与numpy的arange函数功能有点类似，不同的是：
    # 1、arange函数产生的是一维数组，而ogrid函数产生的是二维数组
    # 2、3、ogrid函数产生的数组，第一个数组是以纵向产生的，即数组第二维的大小始终为1。
    # 第二个数组是以横向产生的，即数组第一维的大小始终为1。
    # 例子： kernel_size = 3
    # output = np.ogrid[:kernel_size, :kernel_size]
    # output
    # 输出
    # [array([[0],
    #         [1],
    #         [2]]),
    # array([[0, 1, 2]])]
    og = np.ogrid[:kernel_size, :kernel_size]

    filt = (1 - abs(og[0] - center) / factor) * (
                1 - abs(og[1] - center) / factor)
    weight = np.zeros((in_channels, out_channels, kernel_size, kernel_size),
                      dtype=np.float64)
    # weight[list(range(in_channels)), list(range(out_channels)), :, :]
    # 此处使用列表索引
    # 例子： 取第一行第一列的元素是a[1,1]
    # 取第二行第二列的元素是a[2,2]
    # 那么，a[[1,2],[1,2]]是取什么？
    # 答：先取第一行第一列，再去第二行第二列，一共两个数。
    # 所以此处如果weight是一个（3，3，3，3）的array
    # 取完会变成（3，3，3）的array
    # 如果weight是一个（5，5，3，3）的array
    # 取完会变成（5，3，3）的array
    weight[list(range(in_channels)), list(range(out_channels)), :, :] = filt
    return torch.from_numpy(weight).float()


def colorize_mask(mask, palette):
    """
    对mask进行上色
    input: mask, palette
    # palette 在utils_palette中设置，在dataloder里加载
    output: newmask

    但是实际用的时候新写了这一部分的处理。这个好像不太会用到
    """
    zero_pad = 256 * 3 - len(palette)
    for i in range(zero_pad):
        palette.append(0)
    # P： 8位像素，使用调色板映射到任何其他模式）
    new_mask = PIL.Image.fromarray(mask.astype(np.uint8)).convert('P')
    # 将调色板附加到此图像。
    new_mask.putpalette(palette)
    return new_mask


def set_trainable_attr(m, b):
    """
    设置模型的requires_grad参数
    """
```

```
    m.trainable = b
    for p in m.parameters(): p.requires_grad = b


def apply_leaf(m, f):
    """
    设置叶节点，通过树结构找到网络里的所有元素，并对所有元素使用f method
    """
    c = m if isinstance(m, (list, tuple)) else list(m.children())
    if isinstance(m, nn.Module):
        f(m)
    if len(c) > 0:
        for l in c:
            apply_leaf(l, f)


def set_trainable(l, b):
    """
    设置模型l的所有元素的requires_grad参数
    """
    apply_leaf(l, lambda m: set_trainable_attr(m, b))
```

$$\text{std} = \frac{\text{gain}}{\sqrt{\text{fan\_mode}}}$$