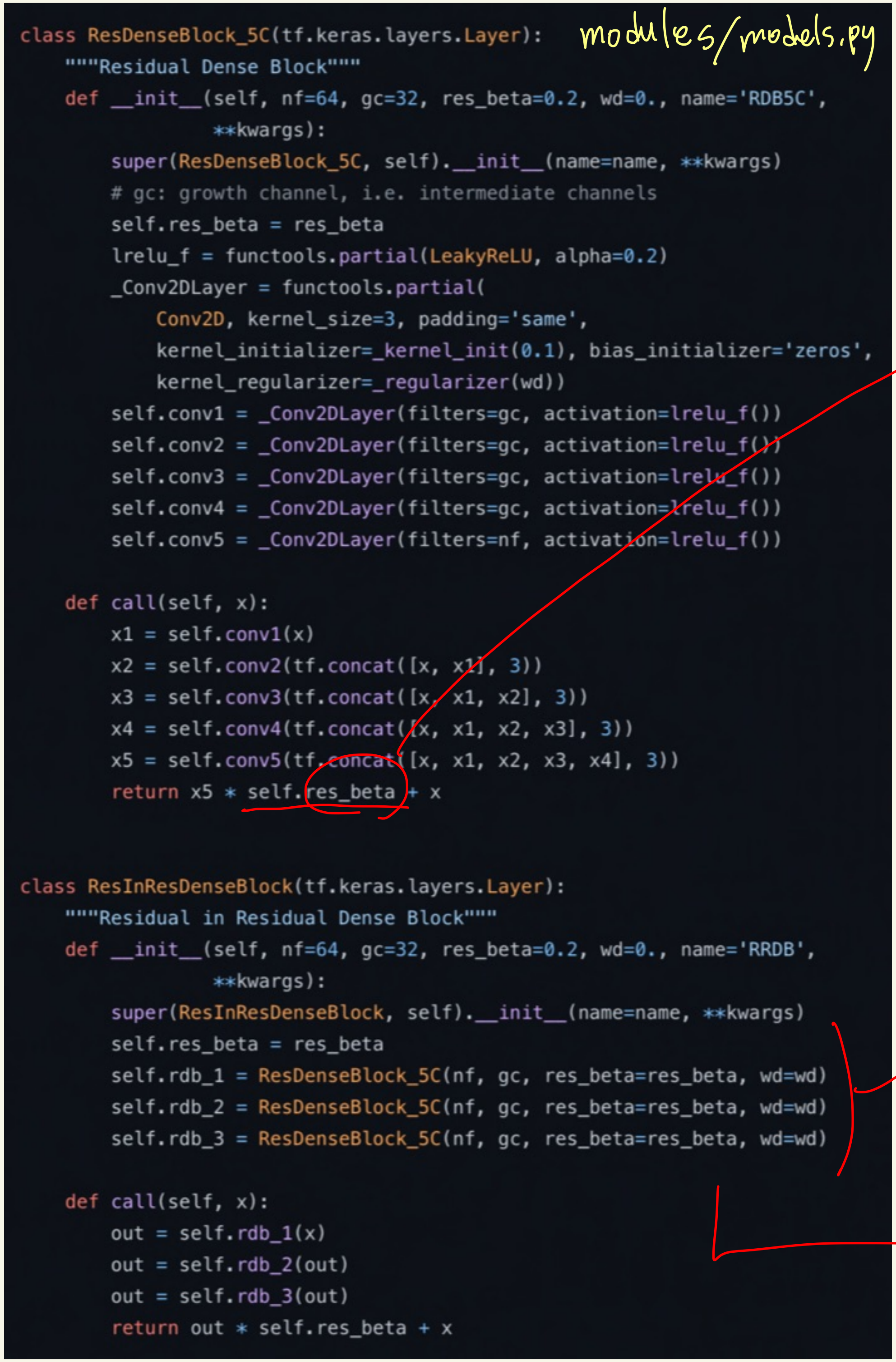
ESRGAN Code 분석

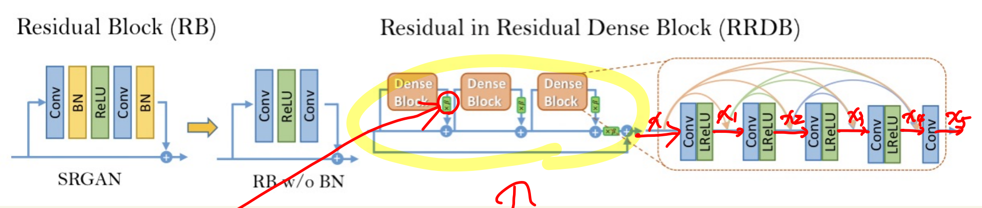
개인적으로 궁금했던 부분 중심으로...

https://github.com/peteryuX/esrgan-tf2



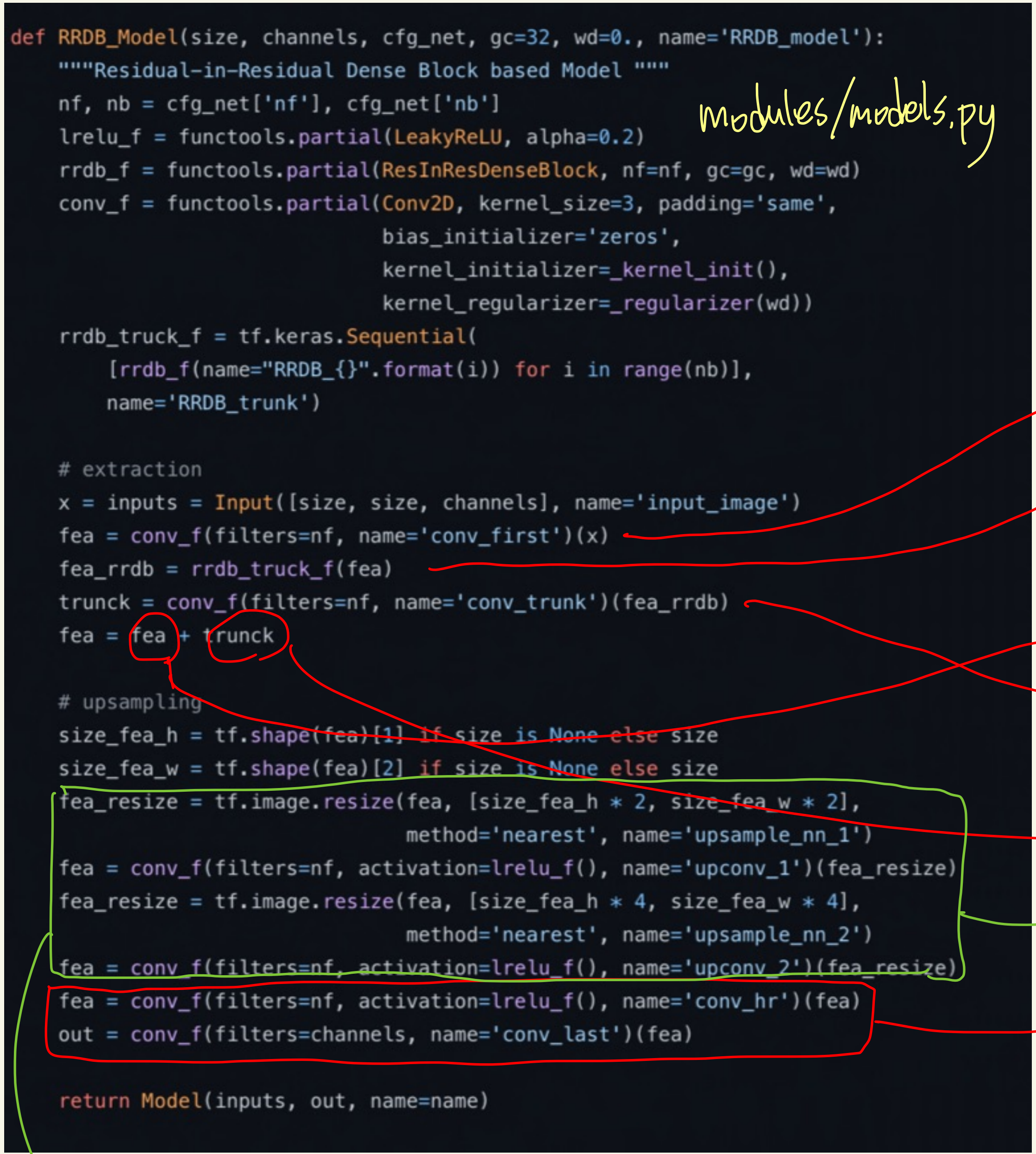
modules/models.py

```python
class ResDenseBlock_5C(tf.keras.layers.Layer):
    """Residual Dense Block"""
    def __init__(self, nf=64, gc=32, res_beta=0.2, wd=0., name='RDB5C',
                 **kwargs):
        super(ResDenseBlock_5C, self).__init__(name=name, **kwargs)
        # gc: growth channel, i.e. intermediate channels
        self.res_beta = res_beta
        lrelu_f = functools.partial(LeakyReLU, alpha=0.2)
        _Conv2DLayer = functools.partial(
            Conv2D, kernel_size=3, padding='same',
            kernel_initializer=_kernel_init(0.1), bias_initializer='zeros',
            kernel_regularizer=_regularizer(wd))
        self.conv1 = _Conv2DLayer(filters=gc, activation=lrelu_f())
        self.conv2 = _Conv2DLayer(filters=gc, activation=lrelu_f())
        self.conv3 = _Conv2DLayer(filters=gc, activation=lrelu_f())
        self.conv4 = _Conv2DLayer(filters=gc, activation=lrelu_f())
        self.conv5 = _Conv2DLayer(filters=nf, activation=lrelu_f())

    def call(self, x):
        x1 = self.conv1(x)
        x2 = self.conv2(tf.concat([x, x1], 3))
        x3 = self.conv3(tf.concat([x, x1, x2], 3))
        x4 = self.conv4(tf.concat([x, x1, x2, x3], 3))
        x5 = self.conv5(tf.concat([x, x1, x2, x3, x4], 3))
        return x5 * self.res_beta + x


class ResInResDenseBlock(tf.keras.layers.Layer):
    """Residual in Residual Dense Block"""
    def __init__(self, nf=64, gc=32, res_beta=0.2, wd=0., name='RRDB',
                 **kwargs):
        super(ResInResDenseBlock, self).__init__(name=name, **kwargs)
        self.res_beta = res_beta
        self.rdb_1 = ResDenseBlock_5C(nf, gc, res_beta=res_beta, wd=wd)
        self.rdb_2 = ResDenseBlock_5C(nf, gc, res_beta=res_beta, wd=wd)
        self.rdb_3 = ResDenseBlock_5C(nf, gc, res_beta=res_beta, wd=wd)

    def call(self, x):
        out = self.rdb_1(x)
        out = self.rdb_2(out)
        out = self.rdb_3(out)
        return out * self.res_beta + x
```



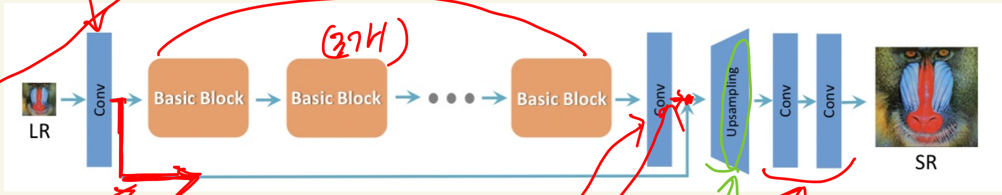
Basic Block 3개.

이코드는 Basic Block 3개 있는 버전.

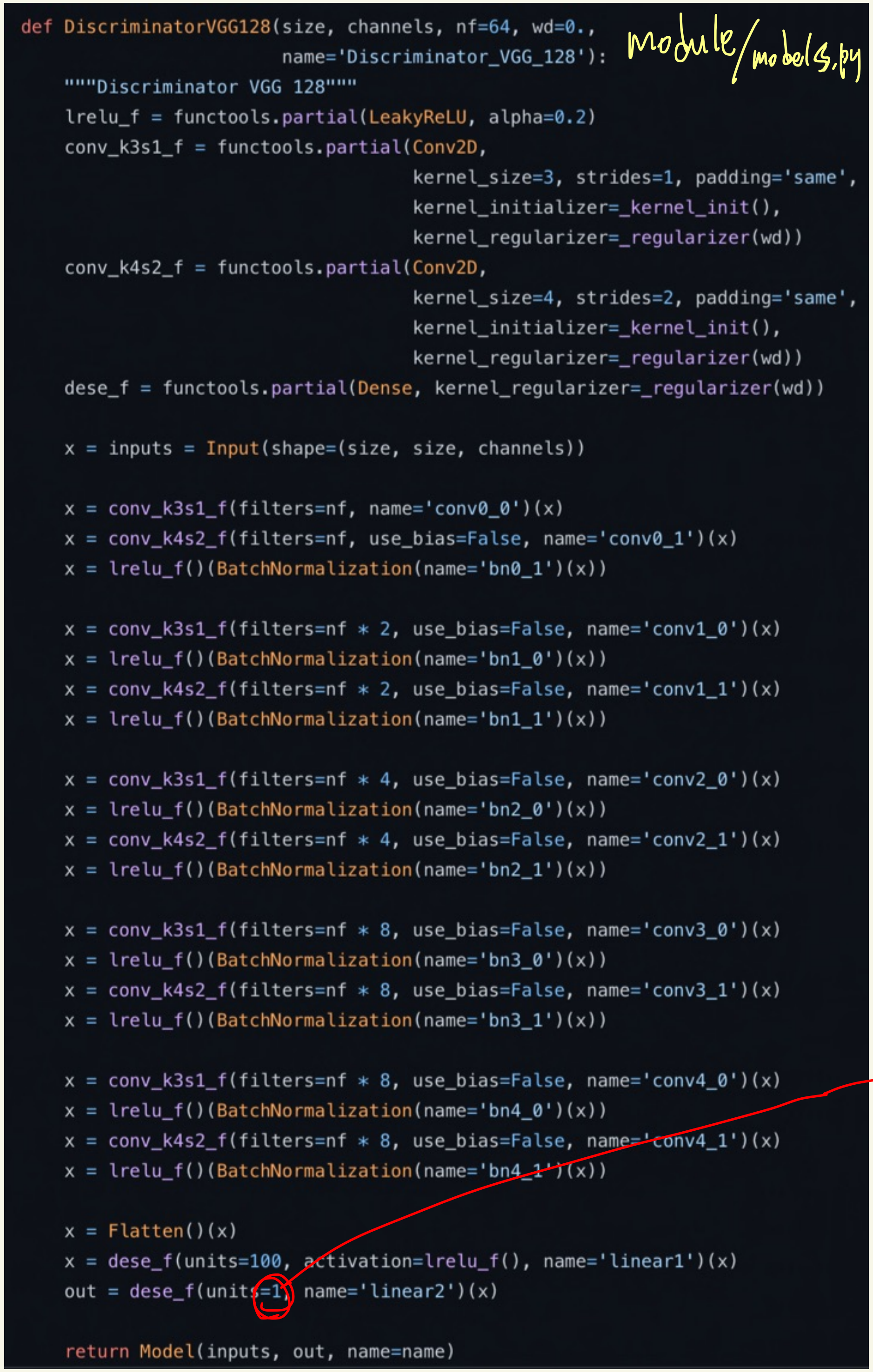


modules/models.py

```python
def RRDB_Model(size, channels, cfg_net, gc=32, wd=0., name='RRDB_model'):
    """Residual-in-Residual Dense Block based Model """
    nf, nb = cfg_net['nf'], cfg_net['nb']
    lrelu_f = functools.partial(LeakyReLU, alpha=0.2)
    rrdb_f = functools.partial(ResInResDenseBlock, nf=nf, gc=gc, wd=wd)
    conv_f = functools.partial(Conv2D, kernel_size=3, padding='same',
                               bias_initializer='zeros',
                               kernel_initializer=_kernel_init(),
                               kernel_regularizer=_regularizer(wd))
    rrdb_truck_f = tf.keras.Sequential(
        [rrdb_f(name="RRDB_{}".format(i)) for i in range(nb)],
        name='RRDB_trunk')

    # extraction
    x = inputs = Input([size, size, channels], name='input_image')
    fea = conv_f(filters=nf, name='conv_first')(x)
    fea_rrdb = rrdb_truck_f(fea)
    trunck = conv_f(filters=nf, name='conv_trunk')(fea_rrdb)
    fea = fea + trunck

    # upsampling
    size_fea_h = tf.shape(fea)[1] if size is None else size
    size_fea_w = tf.shape(fea)[2] if size is None else size
    fea_resize = tf.image.resize(fea, [size_fea_h * 2, size_fea_w * 2],
                                 method='nearest', name='upsample_nn_1')
    fea = conv_f(filters=nf, activation=lrelu_f(), name='upconv_1')(fea_resize)
    fea_resize = tf.image.resize(fea, [size_fea_h * 4, size_fea_w * 4],
                                 method='nearest', name='upsample_nn_2')
    fea = conv_f(filters=nf, activation=lrelu_f(), name='upconv_2')(fea_resize)
    fea = conv_f(filters=nf, activation=lrelu_f(), name='conv_hr')(fea)
    out = conv_f(filters=channels, name='conv_last')(fea)

    return Model(inputs, out, name=name)
```

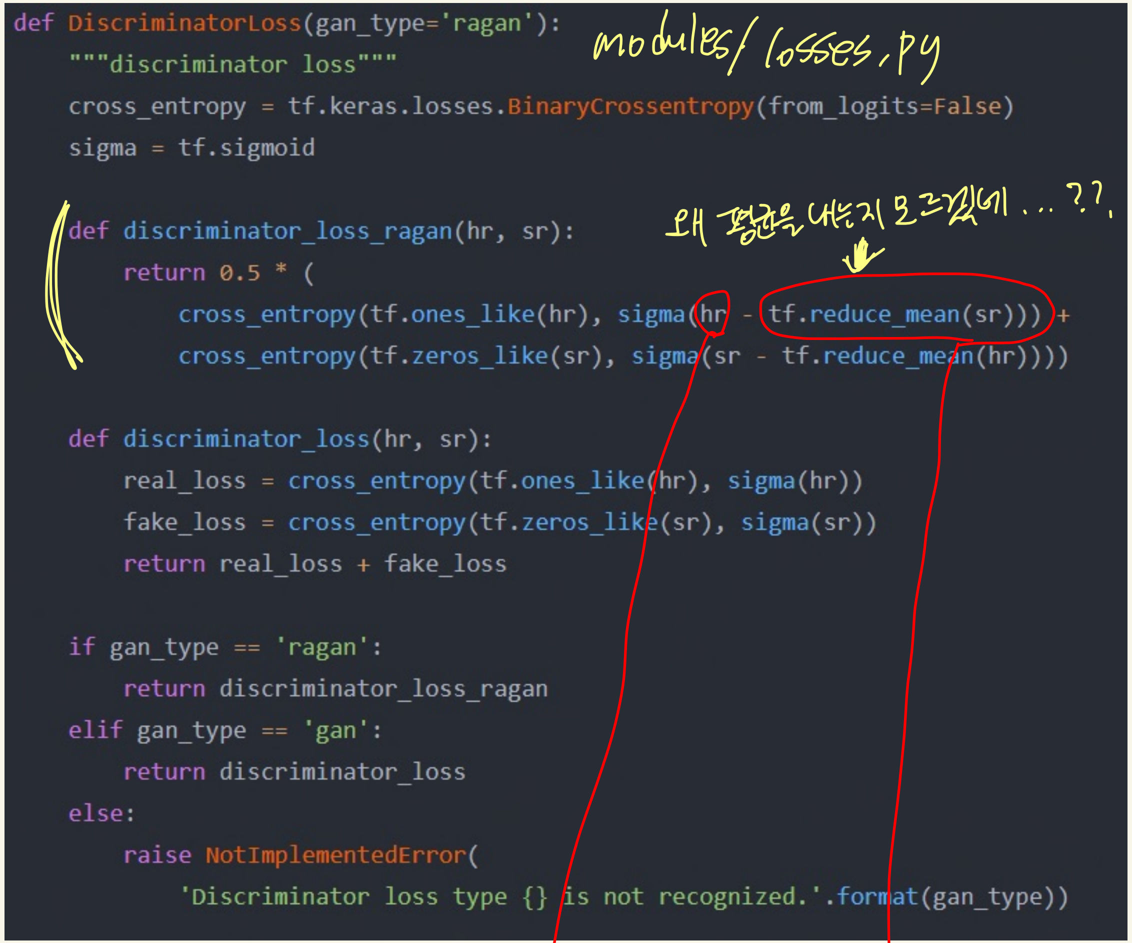


Upsampling은 (resize한 후 Conv)를 두번



module/models.py

```python
def DiscriminatorVGG128(size, channels, nf=64, wd=0.,
                        name='Discriminator_VGG_128'):
    """Discriminator VGG 128"""
    lrelu_f = functools.partial(LeakyReLU, alpha=0.2)
    conv_k3s1_f = functools.partial(Conv2D,
                                    kernel_size=3, strides=1, padding='same',
                                    kernel_initializer=_kernel_init(),
                                    kernel_regularizer=_regularizer(wd))
    conv_k4s2_f = functools.partial(Conv2D,
                                    kernel_size=4, strides=2, padding='same',
                                    kernel_initializer=_kernel_init(),
                                    kernel_regularizer=_regularizer(wd))
    dese_f = functools.partial(Dense, kernel_regularizer=_regularizer(wd))

    x = inputs = Input(shape=(size, size, channels))

    x = conv_k3s1_f(filters=nf, name='conv0_0')(x)
    x = conv_k4s2_f(filters=nf, use_bias=False, name='conv0_1')(x)
    x = lrelu_f()(BatchNormalization(name='bn0_1')(x))

    x = conv_k3s1_f(filters=nf * 2, use_bias=False, name='conv1_0')(x)
    x = lrelu_f()(BatchNormalization(name='bn1_0')(x))
    x = conv_k4s2_f(filters=nf * 2, use_bias=False, name='conv1_1')(x)
    x = lrelu_f()(BatchNormalization(name='bn1_1')(x))

    x = conv_k3s1_f(filters=nf * 4, use_bias=False, name='conv2_0')(x)
    x = lrelu_f()(BatchNormalization(name='bn2_0')(x))
    x = conv_k4s2_f(filters=nf * 4, use_bias=False, name='conv2_1')(x)
    x = lrelu_f()(BatchNormalization(name='bn2_1')(x))

    x = conv_k3s1_f(filters=nf * 8, use_bias=False, name='conv3_0')(x)
    x = lrelu_f()(BatchNormalization(name='bn3_0')(x))
    x = conv_k4s2_f(filters=nf * 8, use_bias=False, name='conv3_1')(x)
    x = lrelu_f()(BatchNormalization(name='bn3_1')(x))

    x = conv_k3s1_f(filters=nf * 8, use_bias=False, name='conv4_0')(x)
    x = lrelu_f()(BatchNormalization(name='bn4_0')(x))
    x = conv_k4s2_f(filters=nf * 8, use_bias=False, name='conv4_1')(x)
    x = lrelu_f()(BatchNormalization(name='bn4_1')(x))

    x = Flatten()(x)
    x = dese_f(units=100, activation=lrelu_f(), name='linear1')(x)
    out = dese_f(units=1, name='linear2')(x)

    return Model(inputs, out, name=name)
```



modules/losses.py

왜 평균을 내는지 모르겠네...??,

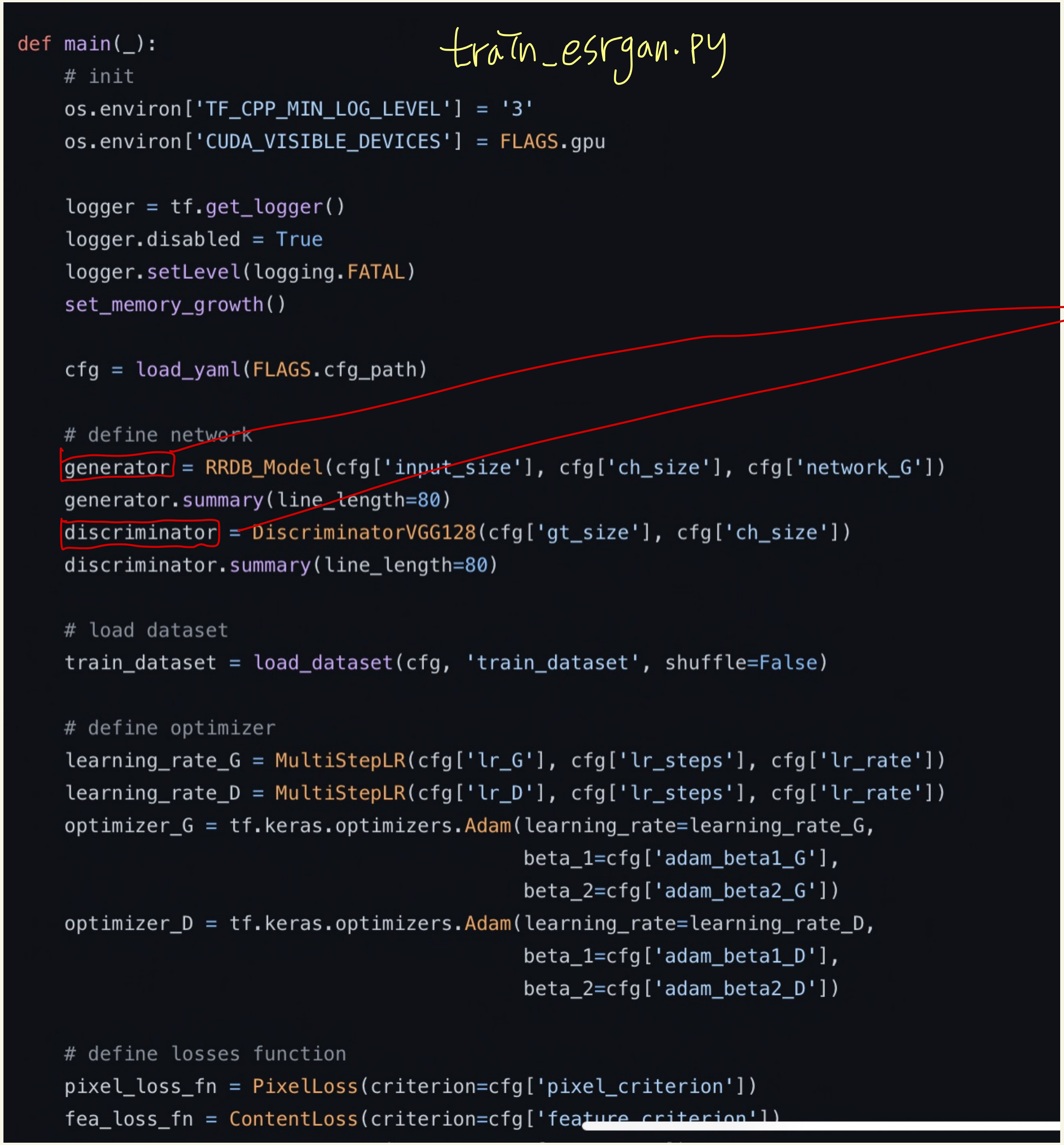
```python
def DiscriminatorLoss(gan_type='ragan'):
    """discriminator loss"""
    cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=False)
    sigma = tf.sigmoid

    def discriminator_loss_ragan(hr, sr):
        return 0.5 * (
            cross_entropy(tf.ones_like(hr), sigma(hr - tf.reduce_mean(sr))) +
            cross_entropy(tf.zeros_like(sr), sigma(sr - tf.reduce_mean(hr))))

    def discriminator_loss(hr, sr):
        real_loss = cross_entropy(tf.ones_like(hr), sigma(hr))
        fake_loss = cross_entropy(tf.zeros_like(sr), sigma(sr))
        return real_loss + fake_loss

    if gan_type == 'ragan':
        return discriminator_loss_ragan
    elif gan_type == 'gan':
        return discriminator_loss
    else:
        raise NotImplementedError(
            'Discriminator loss type {} is not recognized.'.format(gan_type))
```

$$L_D^{Ra} = -\mathbb{E}_{x_r}[\log(D_{Ra}(x_r, x_f))] - \mathbb{E}_{x_f}[\log(1 - D_{Ra}(x_f, x_r))].$$

이진분류.

Discriminator의 output.
배치사이즈가 5라면
hr=[0.8, 0.9, 0.87, 0.5, 0.7] 와 같을 듯.

sr도 같은 형식일듯
sr=[0.1, 0.5, 0.4, ...]
tf.reduce_mean
통과하면 sr의 평균



train_esrgan.py

```python
def main(_):
    # init
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
    os.environ['CUDA_VISIBLE_DEVICES'] = FLAGS.gpu

    logger = tf.get_logger()
    logger.disabled = True
    logger.setLevel(logging.FATAL)
    set_memory_growth()

    cfg = load_yaml(FLAGS.cfg_path)

    # define network
    generator = RRDB_Model(cfg['input_size'], cfg['ch_size'], cfg['network_G'])
    generator.summary(line_length=80)
    discriminator = DiscriminatorVGG128(cfg['gt_size'], cfg['ch_size'])
    discriminator.summary(line_length=80)

    # load dataset
    train_dataset = load_dataset(cfg, 'train_dataset', shuffle=False)

    # define optimizer
    learning_rate_G = MultiStepLR(cfg['lr_G'], cfg['lr_steps'], cfg['lr_rate'])
    learning_rate_D = MultiStepLR(cfg['lr_D'], cfg['lr_steps'], cfg['lr_rate'])
    optimizer_G = tf.keras.optimizers.Adam(learning_rate=learning_rate_G,
                                           beta_1=cfg['adam_beta1_G'],
                                           beta_2=cfg['adam_beta2_G'])
    optimizer_D = tf.keras.optimizers.Adam(learning_rate=learning_rate_D,
                                           beta_1=cfg['adam_beta1_D'],
                                           beta_2=cfg['adam_beta2_D'])

    # define losses function
    pixel_loss_fn = PixelLoss(criterion=cfg['pixel_criterion'])
    fea_loss_fn = ContentLoss(criterion=cfg['feature_criterion'])
```

generator와 discriminator만 기존 모델과 동일한 출력 사이즈로 맞춰주면 다른 모델로 치환가능할 듯,