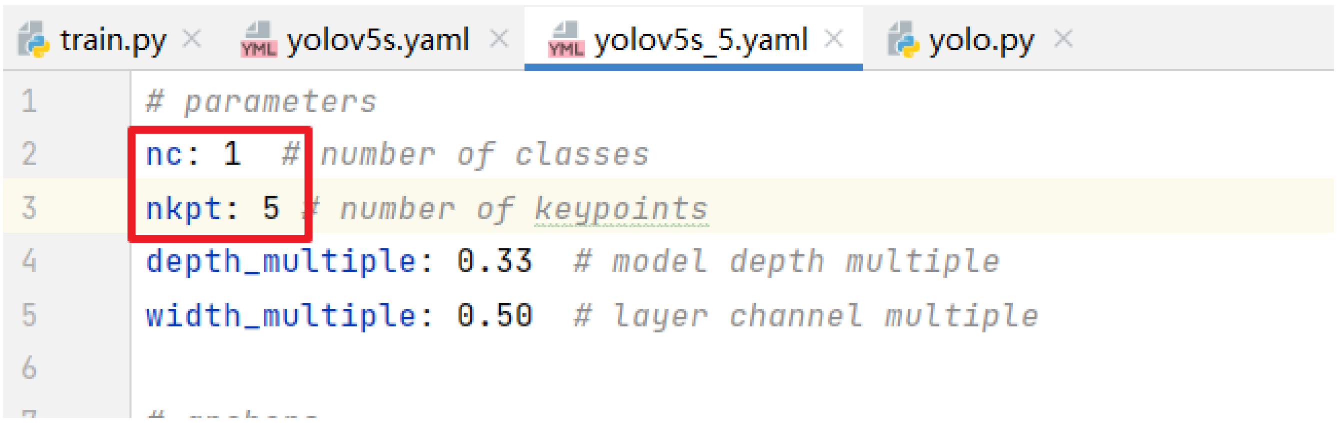
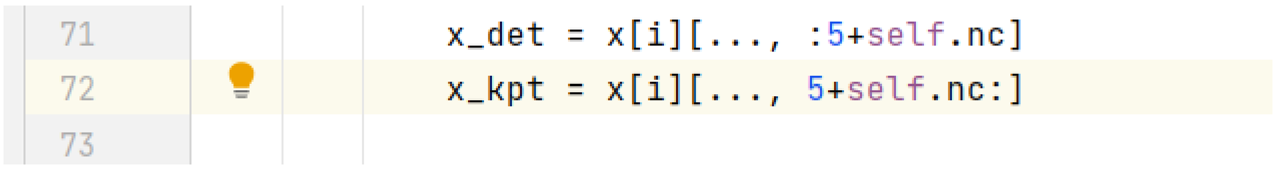
yoloposev5，任意关键点+多类别修改记录

修改模型文件的nc和nkpt参数

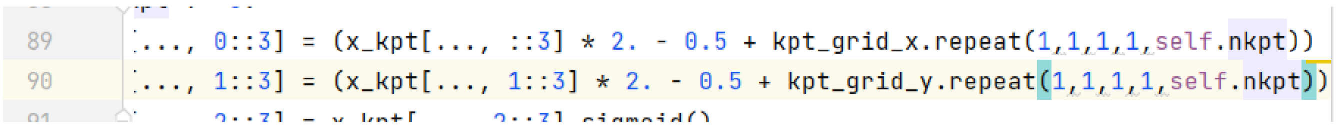


```
# parameters
nc: 1  # number of classes
nkpt: 5 # number of keypoints
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

```

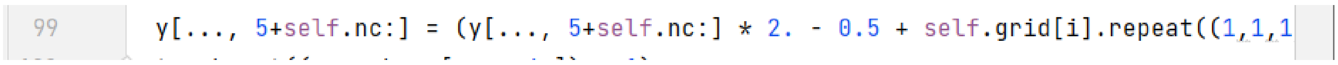
修改models/yolo.py



```
            x_det = x[i][..., :5+self.nc]
            x_kpt = x[i][..., 5+self.nc:]
```

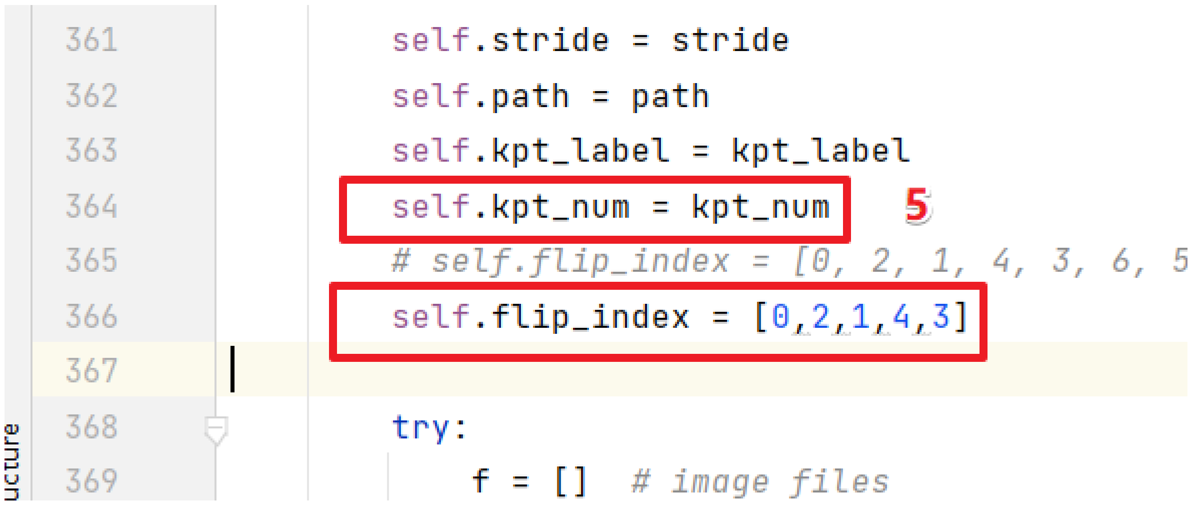


```
..., 0::3] = (x_kpt[..., ::3] * 2. - 0.5 + kpt_grid_x.repeat(1,1,1,1,self.nkpt))
..., 1::3] = (x_kpt[..., 1::3] * 2. - 0.5 + kpt_grid_y.repeat(1,1,1,1,self.nkpt))
```

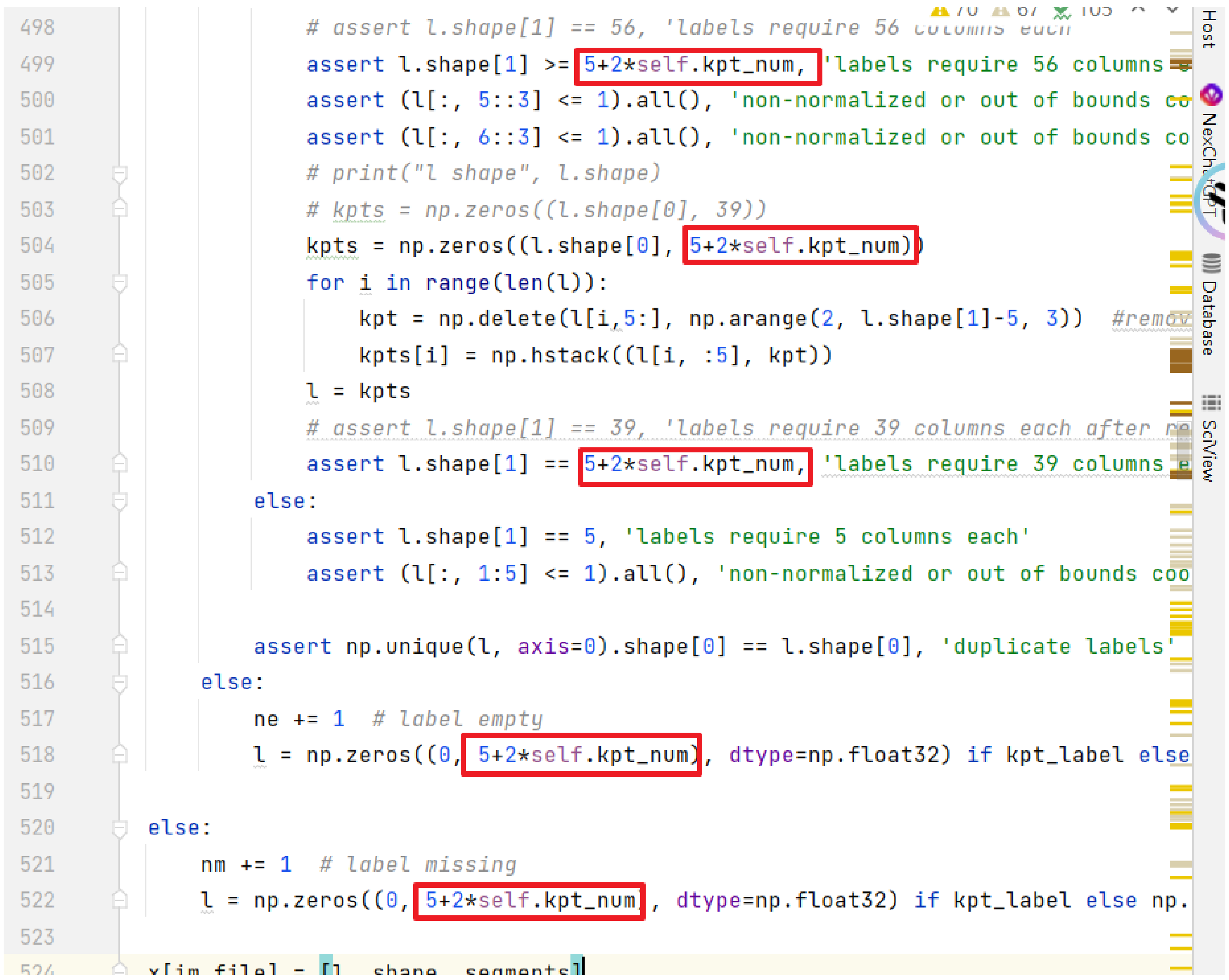


```
 y[..., 5+self.nc:] = (y[..., 5+self.nc:] * 2. - 0.5 + self.grid[i].repeat((1,1,1
```

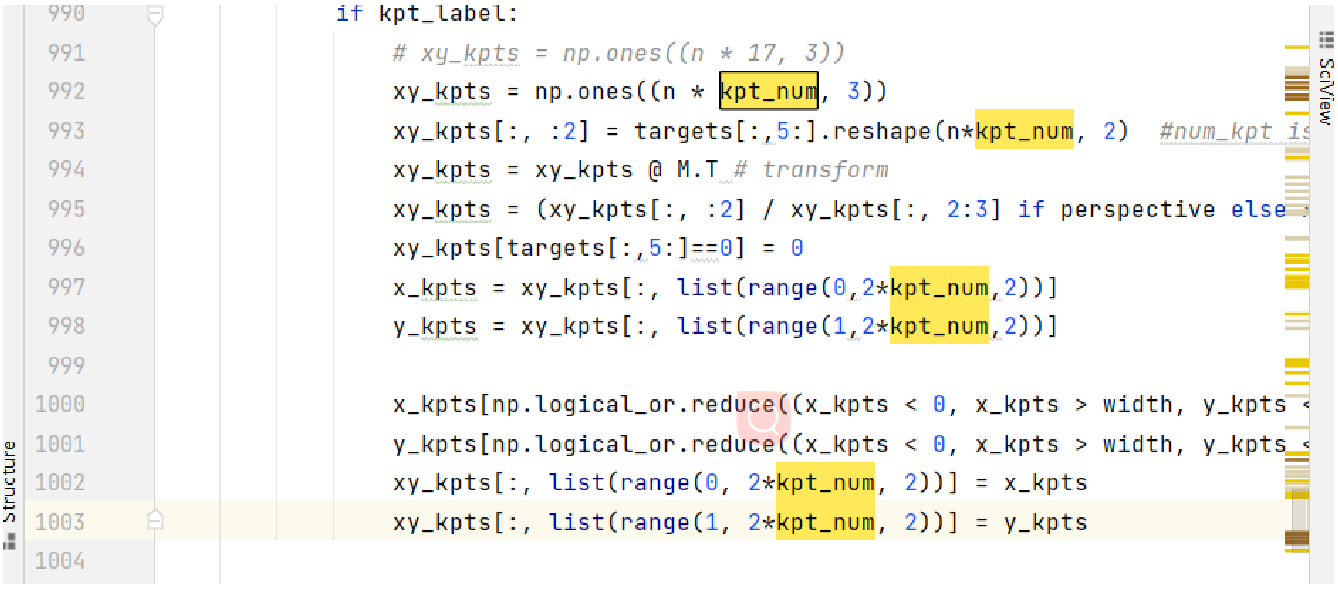
修改utils/dataset.py



```
    self.stride = stride
    self.path = path
    self.kpt_label = kpt_label
    self.kpt_num = kpt_num
    # self.flip_index = [0, 2, 1, 4, 3, 6, 5
    self.flip_index = [0,2,1,4,3]

    try:
        f = []  # image files
```

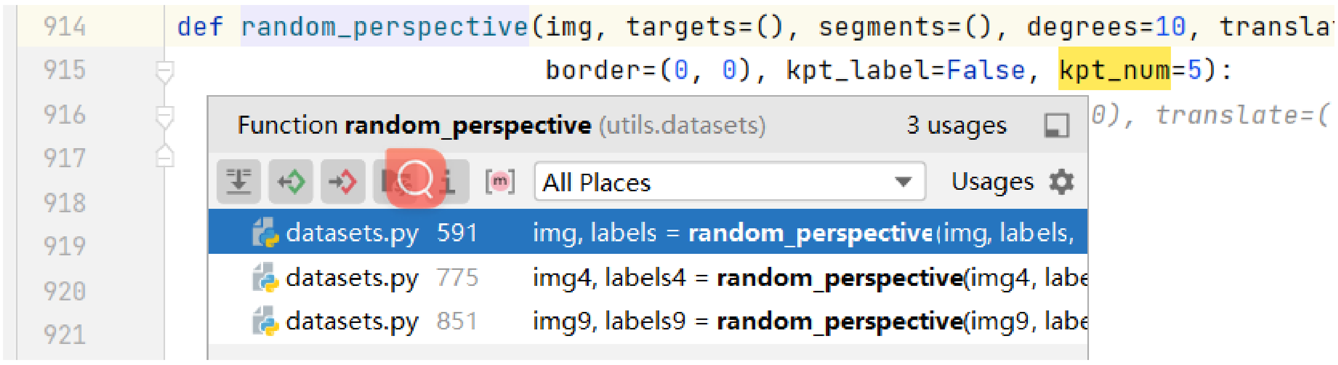


```
            # assert l.shape[1] == 56, 'labels require 56 columns each
            assert l.shape[1] >= 5+2*self.kpt_num, 'labels require 56 columns e
            assert (l[:, 5::3] <= 1).all(), 'non-normalized or out of bounds co
            assert (l[:, 6::3] <= 1).all(), 'non-normalized or out of bounds co
            # print("l shape", l.shape)
            # kpts = np.zeros((l.shape[0], 39))
            kpts = np.zeros((l.shape[0], 5+2*self.kpt_num))
            for i in range(len(l)):
                kpt = np.delete(l[i,5:], np.arange(2, l.shape[1]-5, 3))  #remov
                kpts[i] = np.hstack((l[i, :5], kpt))
            l = kpts
            # assert l.shape[1] == 39, 'labels require 39 columns each after re
            assert l.shape[1] == 5+2*self.kpt_num, 'labels require 39 columns e
        else:
            assert l.shape[1] == 5, 'labels require 5 columns each'
            assert (l[:, 1:5] <= 1).all(), 'non-normalized or out of bounds coo

        assert np.unique(l, axis=0).shape[0] == l.shape[0], 'duplicate labels'
    else:
        ne += 1  # label empty
        l = np.zeros((0, 5+2*self.kpt_num), dtype=np.float32) if kpt_label else

else:
    nm += 1  # label missing
    l = np.zeros((0, 5+2*self.kpt_num), dtype=np.float32) if kpt_label else np.

x[im_file] = [l, shape, segments]
```



```
            if kpt_label:
                # xy_kpts = np.ones((n * 17, 3))
                xy_kpts = np.ones((n * kpt_num, 3))
                xy_kpts[:, :2] = targets[:,5:].reshape(n*kpt_num, 2)  #num_kpt_is
                xy_kpts = xy_kpts @ M.T # transform
                xy_kpts = (xy_kpts[:, :2] / xy_kpts[:, 2:3] if perspective else
                xy_kpts[targets[:,5:]==0] = 0
                x_kpts = xy_kpts[:, list(range(0,2*kpt_num,2))]
                y_kpts = xy_kpts[:, list(range(1,2*kpt_num,2))]

                x_kpts[np.logical_or.reduce((x_kpts < 0, x_kpts > width, y_kpts <
                y_kpts[np.logical_or.reduce((x_kpts < 0, x_kpts > width, y_kpts <
                xy_kpts[:, list(range(0, 2*kpt_num, 2))] = x_kpts
                xy_kpts[:, list(range(1, 2*kpt_num, 2))] = y_kpts
```

random_perspective添加kpt_num,以及调用random_perspective的地方



```
def random_perspective(img, targets=(), segments=(), degrees=10, transla
                       border=(0, 0), kpt_label=False, kpt_num=5):
                                                      0), translate=(
```

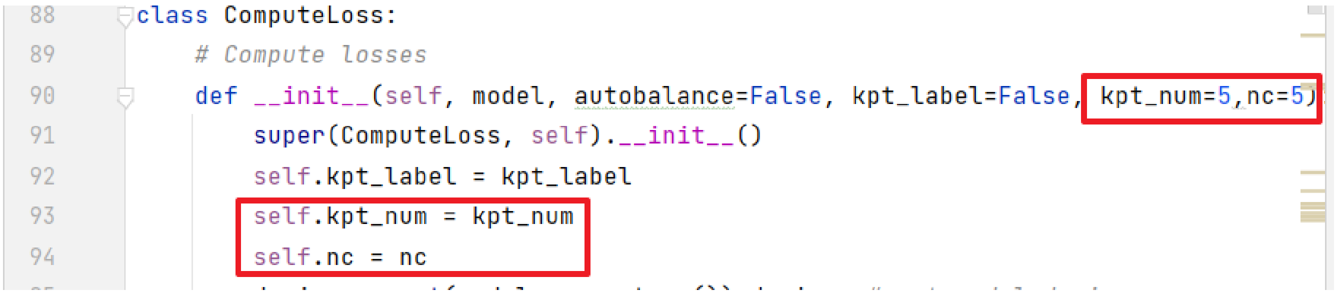
Function **random_perspective** (utils.datasets) 3 usages

All Places   Usages

datasets.py	591	img, labels = **random_perspective**(img, labels,
datasets.py	775	img4, labels4 = **random_perspective**(img4, labe
datasets.py	851	img9, labels9 = **random_perspective**(img9, labe

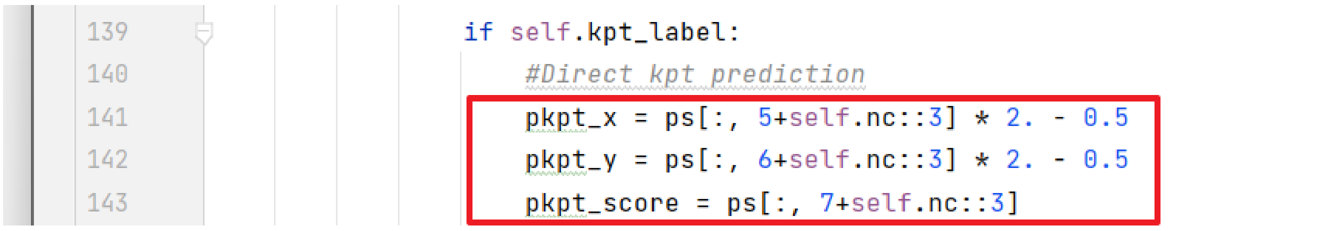
loss.py修改

loss初始化类别数量和关机键数量

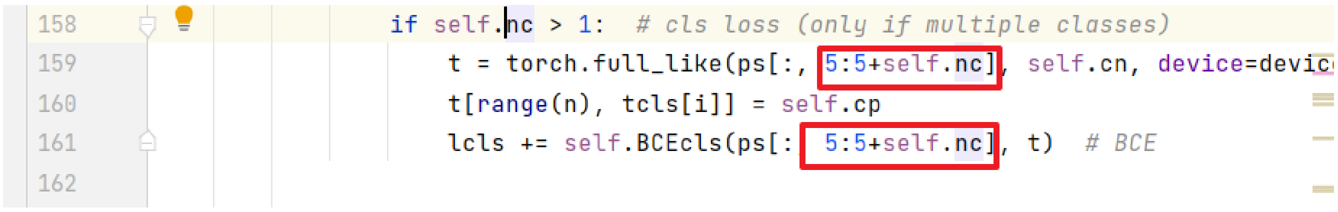


```
class ComputeLoss:
    # Compute losses
    def __init__(self, model, autobalance=False, kpt_label=False, kpt_num=5,nc=5)
        super(ComputeLoss, self).__init__()
        self.kpt_label = kpt_label
        self.kpt_num = kpt_num
        self.nc = nc
```

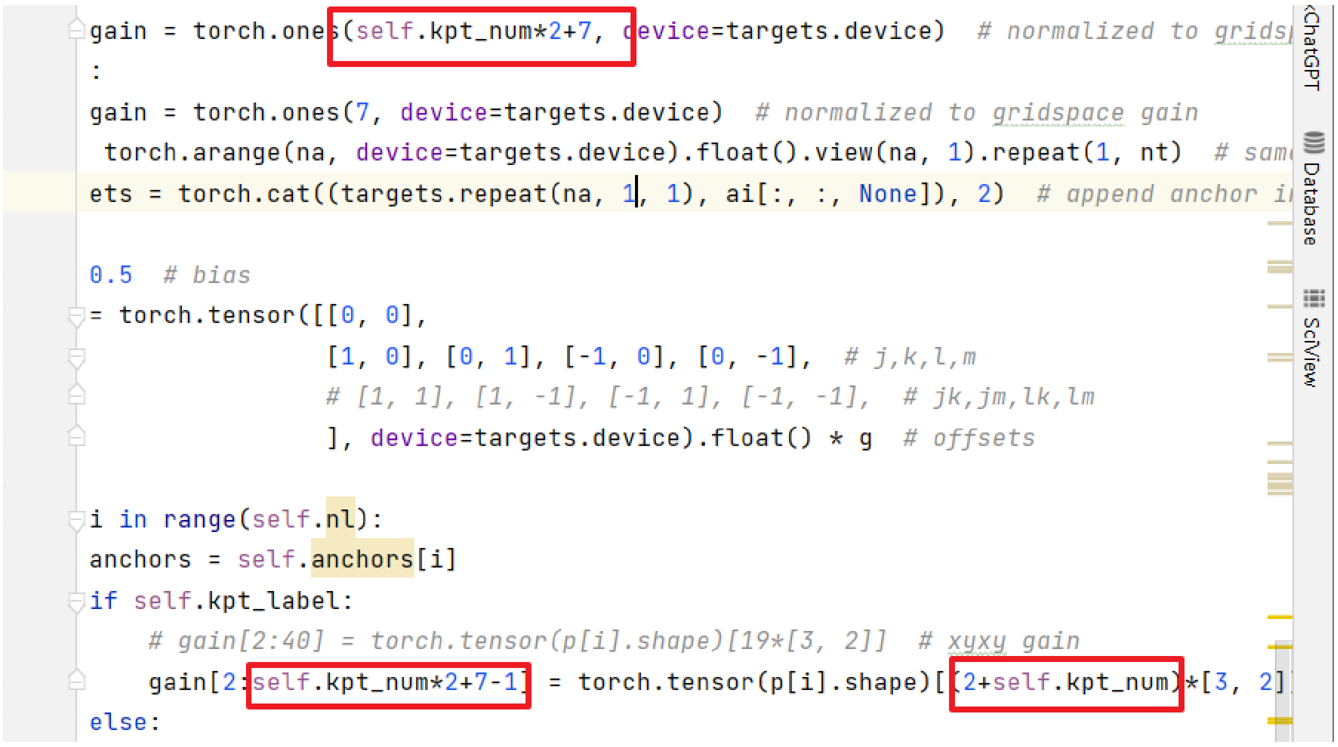
loss 计算中加入类别和关机键点计算



```
            if self.kpt_label:
                #Direct kpt prediction
                pkpt_x = ps[:, 5+self.nc::3] * 2. - 0.5
                pkpt_y = ps[:, 6+self.nc::3] * 2. - 0.5
                pkpt_score = ps[:, 7+self.nc::3]
```

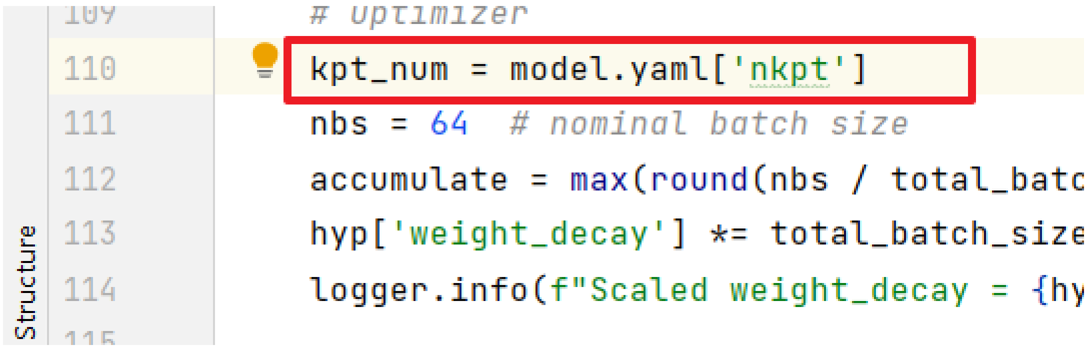


```
        if self.nc > 1:  # cls loss (only if multiple classes)
            t = torch.full_like(ps[:, 5:5+self.nc], self.cn, device=devic
            t[range(n), tcls[i]] = self.cp
            lcls += self.BCEcls(ps[: 5:5+self.nc], t)  # BCE
```

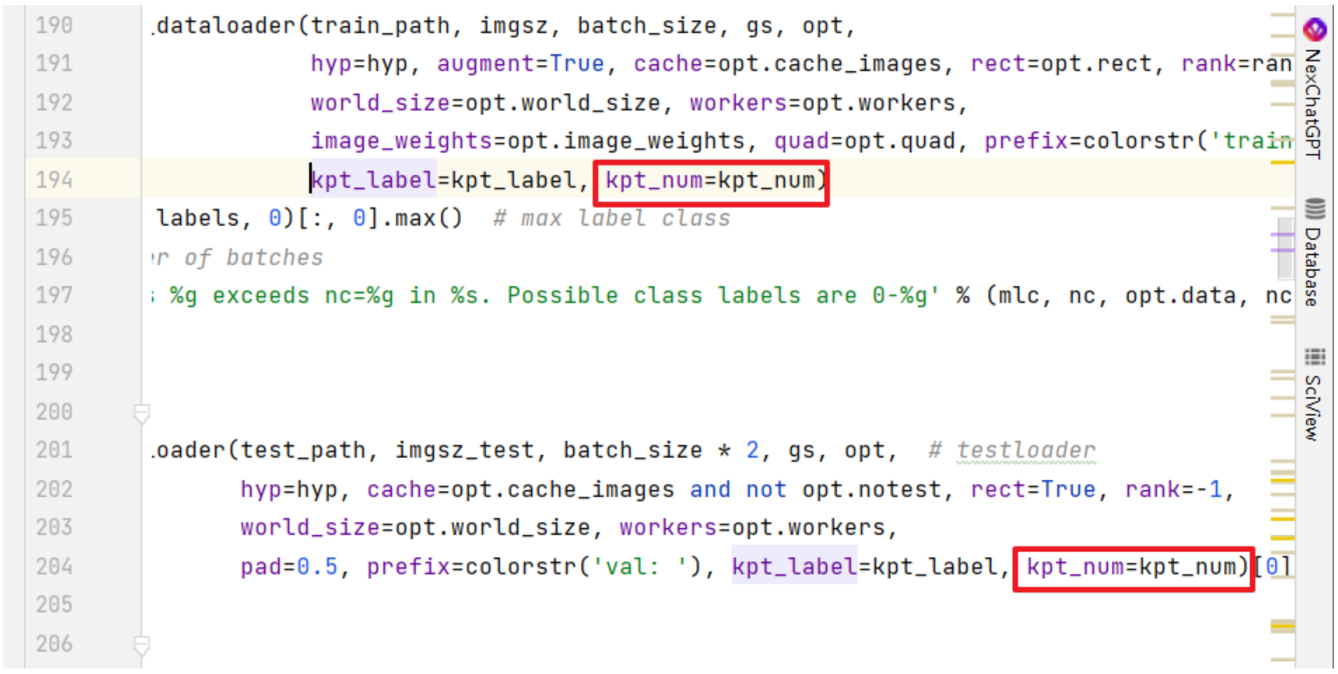


```
gain = torch.ones(self.kpt_num*2+7, device=targets.device)  # normalized to grids
:
gain = torch.ones(7, device=targets.device)  # normalized to gridspace gain
 torch.arange(na, device=targets.device).float().view(na, 1).repeat(1, nt)  # sam
ets = torch.cat((targets.repeat(na, 1, 1), ai[:, :, None]), 2)  # append anchor i

0.5  # bias
= torch.tensor([[0, 0],
                [1, 0], [0, 1], [-1, 0], [0, -1],  # j,k,l,m
                # [1, 1], [1, -1], [-1, 1], [-1, -1],  # jk,jm,lk,lm
                ], device=targets.device).float() * g  # offsets

i in range(self.nl):
anchors = self.anchors[i]
if self.kpt_label:
    # gain[2:40] = torch.tensor(p[i].shape)[19*[3, 2]]  # xyxy gain
    gain[2:self.kpt_num*2+7-1] = torch.tensor(p[i].shape)[(2+self.kpt_num)*[3, 2]]
else:
```

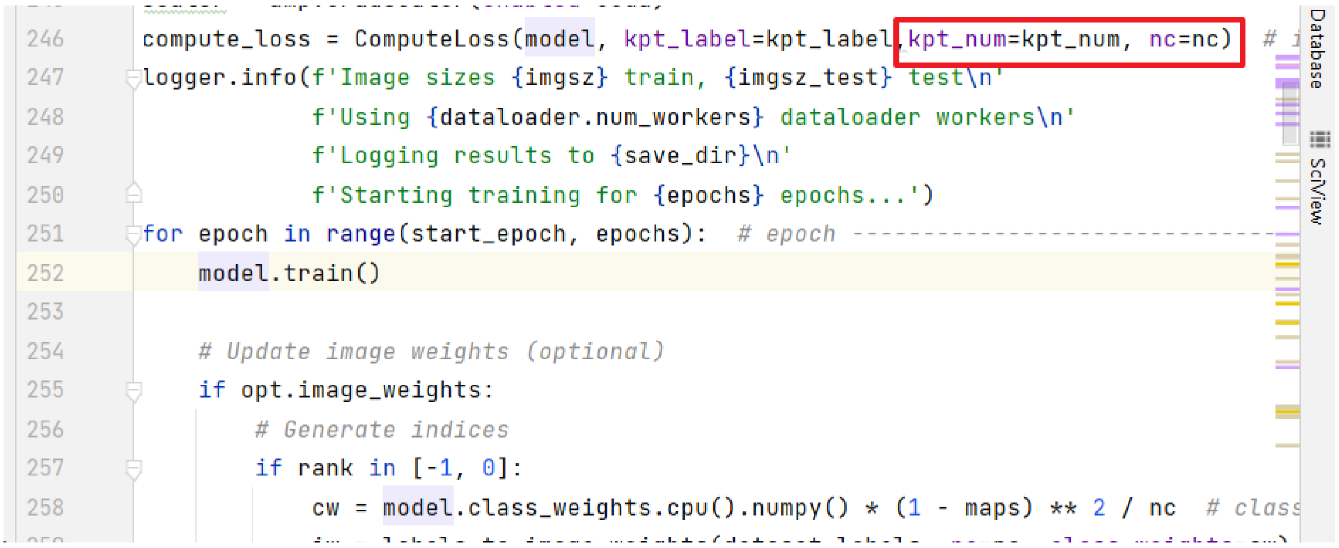
train.py修改



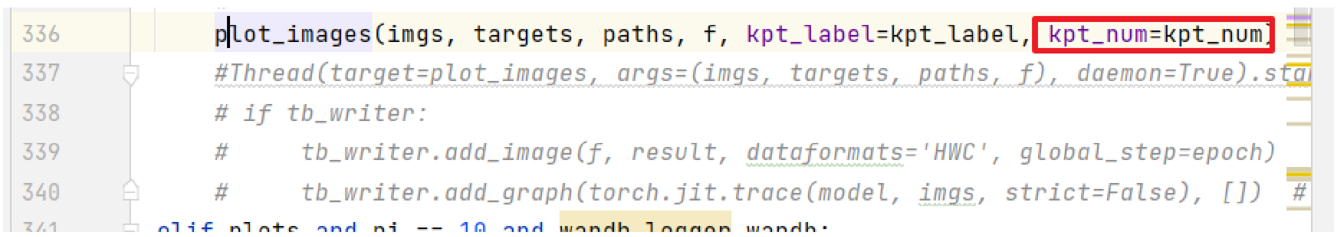
```
    # Optimizer
    kpt_num = model.yaml['nkpt']
    nbs = 64  # nominal batch size
    accumulate = max(round(nbs / total_batc
    hyp['weight_decay'] *= total_batch_size
    logger.info(f"Scaled weight_decay = {hy
```



```
.dataloader(train_path, imgsz, batch_size, gs, opt,
            hyp=hyp, augment=True, cache=opt.cache_images, rect=opt.rect, rank=ran
            world_size=opt.world_size, workers=opt.workers,
            image_weights=opt.image_weights, quad=opt.quad, prefix=colorstr('train
            kpt_label=kpt_label, kpt_num=kpt_num)
 labels, 0)[:, 0].max()  # max label class
r of batches
 %g exceeds nc=%g in %s. Possible class labels are 0-%g' % (mlc, nc, opt.data, nc



.oader(test_path, imgsz_test, batch_size * 2, gs, opt,  # testloader
       hyp=hyp, cache=opt.cache_images and not opt.notest, rect=True, rank=-1,
       world_size=opt.world_size, workers=opt.workers,
       pad=0.5, prefix=colorstr('val: '), kpt_label=kpt_label, kpt_num=kpt_num)[0]

```

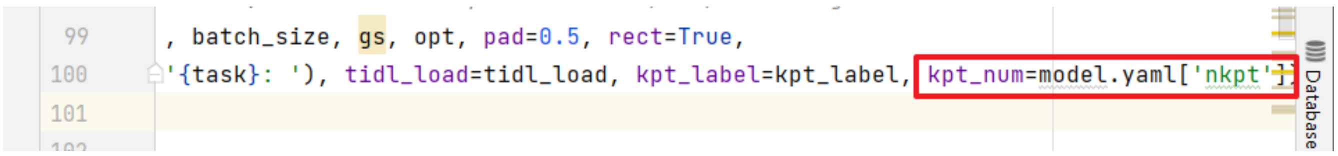


```
compute_loss = ComputeLoss(model, kpt_label=kpt_label, kpt_num=kpt_num, nc=nc)  # i
logger.info(f'Image sizes {imgsz} train, {imgsz_test} test\n'
            f'Using {dataloader.num_workers} dataloader workers\n'
            f'Logging results to {save_dir}\n'
            f'Starting training for {epochs} epochs...')
for epoch in range(start_epoch, epochs):  # epoch ------------------------------
    model.train()

    # Update image weights (optional)
    if opt.image_weights:
        # Generate indices
        if rank in [-1, 0]:
            cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc  # class
```

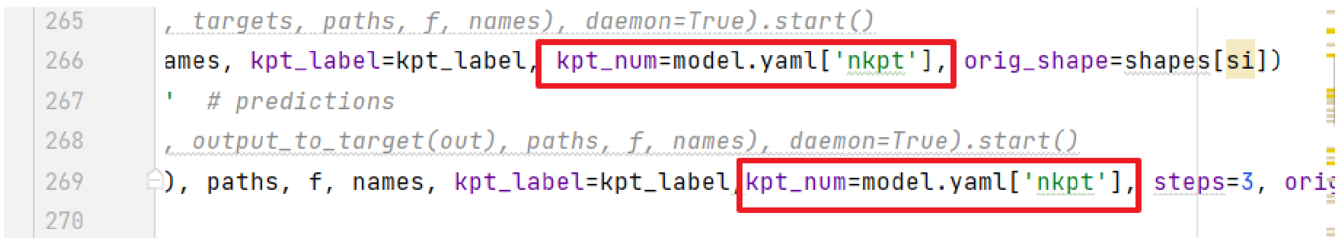


```
    plot_images(imgs, targets, paths, f, kpt_label=kpt_label, kpt_num=kpt_num)
    #Thread(target=plot_images, args=(imgs, targets, paths, f), daemon=True).sta
    # if tb_writer:
    #     tb_writer.add_image(f, result, dataformats='HWC', global_step=epoch)
    #     tb_writer.add_graph(torch.jit.trace(model, imgs, strict=False), [])  #
```

test.py修改

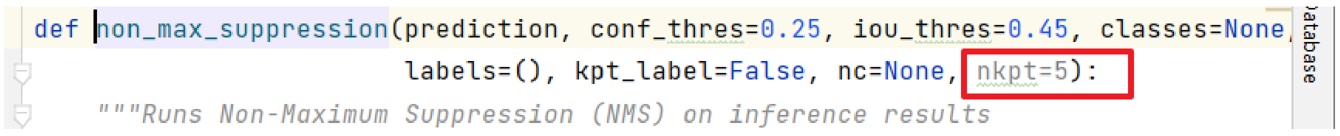


```
, batch_size, gs, opt, pad=0.5, rect=True,
'{task}: '), tidl_load=tidl_load, kpt_label=kpt_label, kpt_num=model.yaml['nkpt']

```

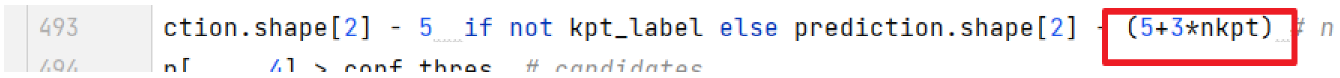


```
, targets, paths, f, names), daemon=True).start()
ames, kpt_label=kpt_label, kpt_num=model.yaml['nkpt'], orig_shape=shapes[si])
'  # predictions
, output_to_target(out), paths, f, names), daemon=True).start()
), paths, f, names, kpt_label=kpt_label, kpt_num=model.yaml['nkpt'], steps=3, orig
```

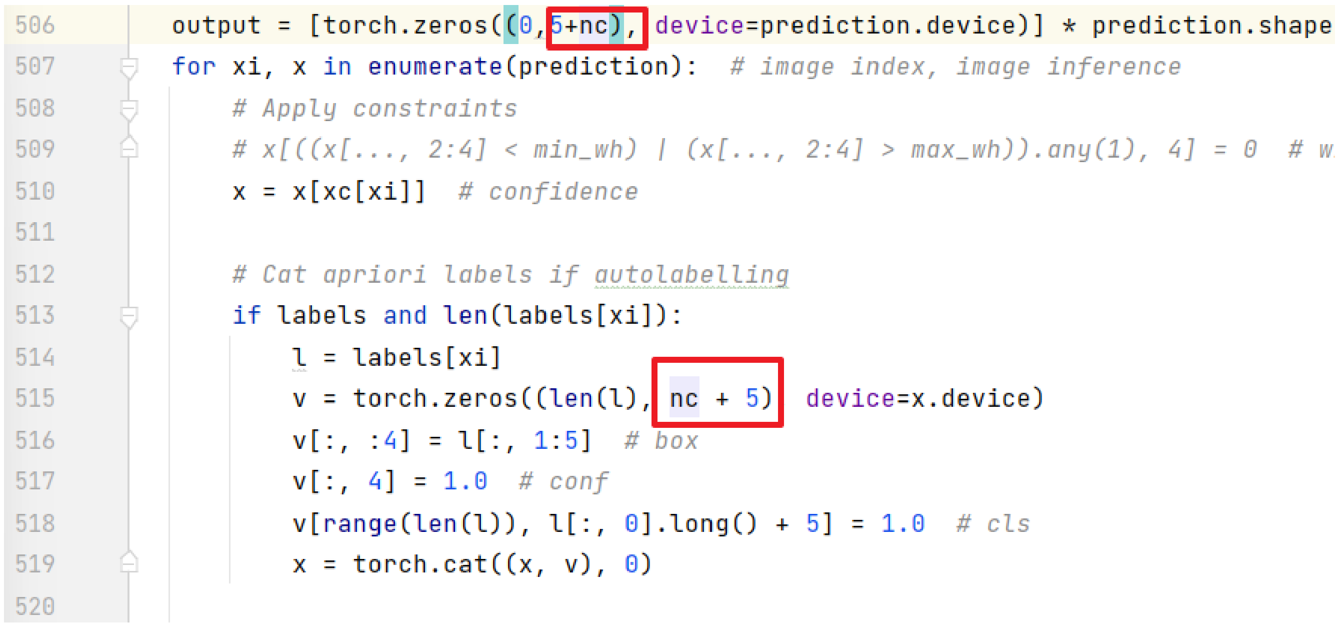
general.py 中non_max_supperssion()方法修改



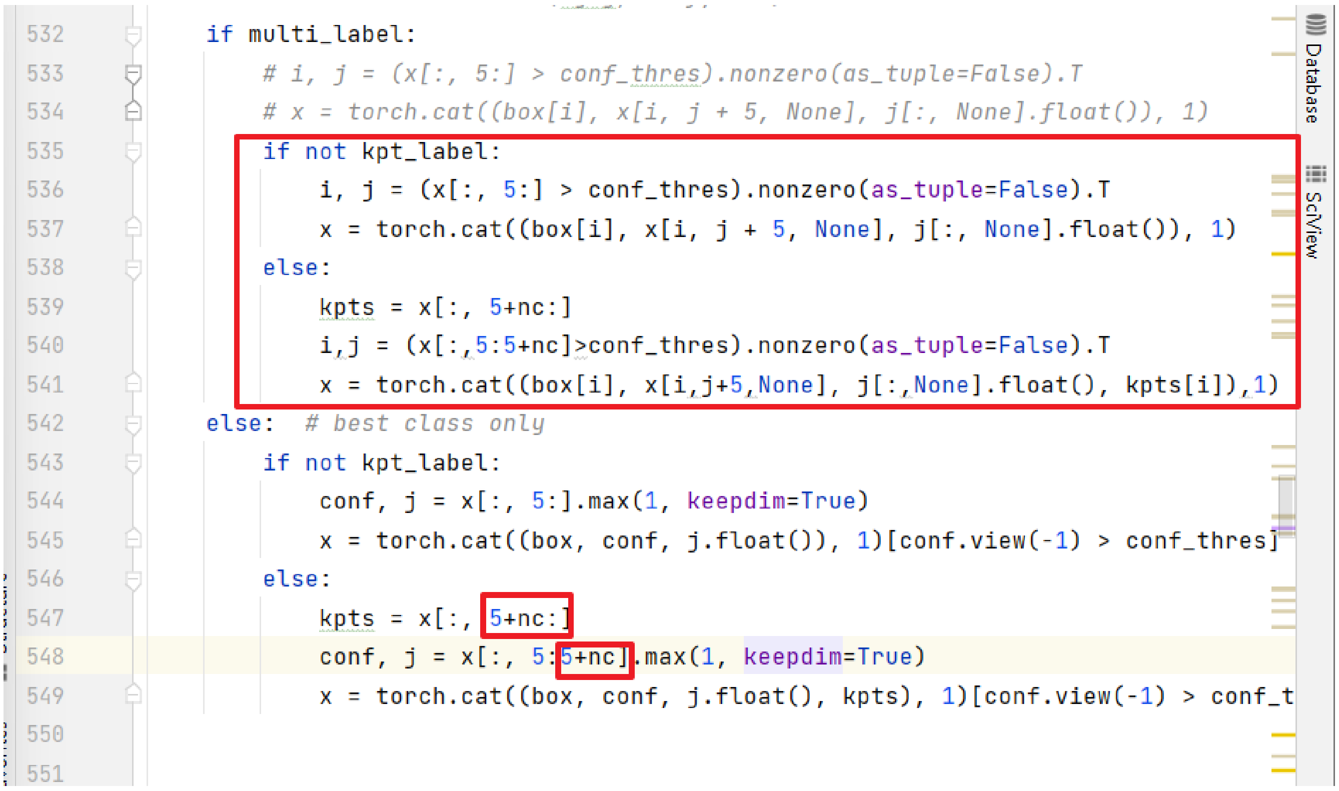
```
def non_max_suppression(prediction, conf_thres=0.25, iou_thres=0.45, classes=None,
                        labels=(), kpt_label=False, nc=None, nkpt=5):
    """Runs Non-Maximum Suppression (NMS) on inference results
```



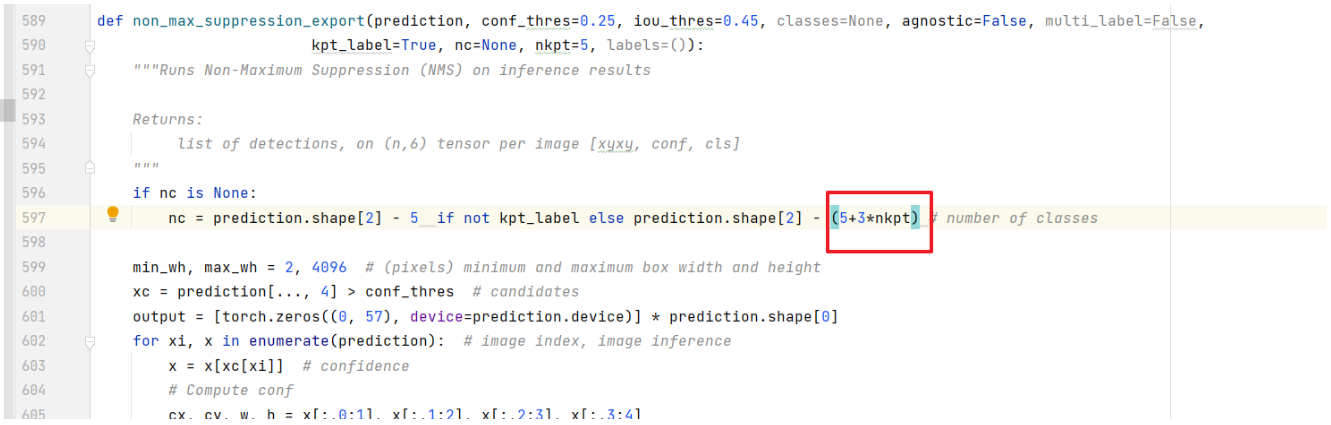
```
ction.shape[2] - 5  if not kpt_label else prediction.shape[2] - (5+3*nkpt)  # n
n[      4] > conf_thres  # candidates
```



```
    output = [torch.zeros((0,5+nc), device=prediction.device)] * prediction.shape
    for xi, x in enumerate(prediction):  # image index, image inference
        # Apply constraints
        # x[((x[..., 2:4] < min_wh) | (x[..., 2:4] > max_wh)).any(1), 4] = 0  # w
        x = x[xc[xi]]  # confidence

        # Cat apriori labels if autolabelling
        if labels and len(labels[xi]):
            l = labels[xi]
            v = torch.zeros((len(l), nc + 5), device=x.device)
            v[:, :4] = l[:, 1:5]  # box
            v[:, 4] = 1.0  # conf
            v[range(len(l)), l[:, 0].long() + 5] = 1.0  # cls
            x = torch.cat((x, v), 0)
```

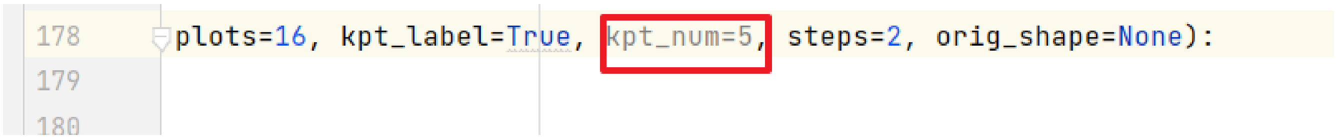


```python
            if multi_label:
                # i, j = (x[:, 5:] > conf_thres).nonzero(as_tuple=False).T
                # x = torch.cat((box[i], x[i, j + 5, None], j[:, None].float()), 1)
                if not kpt_label:
                    i, j = (x[:, 5:] > conf_thres).nonzero(as_tuple=False).T
                    x = torch.cat((box[i], x[i, j + 5, None], j[:, None].float()), 1)
                else:
                    kpts = x[:, 5+nc:]
                    i,j = (x[:,5:5+nc]>conf_thres).nonzero(as_tuple=False).T
                    x = torch.cat((box[i], x[i,j+5,None], j[:,None].float(), kpts[i]),1)
            else:  # best class only
                if not kpt_label:
                    conf, j = x[:, 5:].max(1, keepdim=True)
                    x = torch.cat((box, conf, j.float()), 1)[conf.view(-1) > conf_thres]
                else:
                    kpts = x[:, 5+nc:]
                    conf, j = x[:, 5:5+nc].max(1, keepdim=True)
                    x = torch.cat((box, conf, j.float(), kpts), 1)[conf.view(-1) > conf_t
```

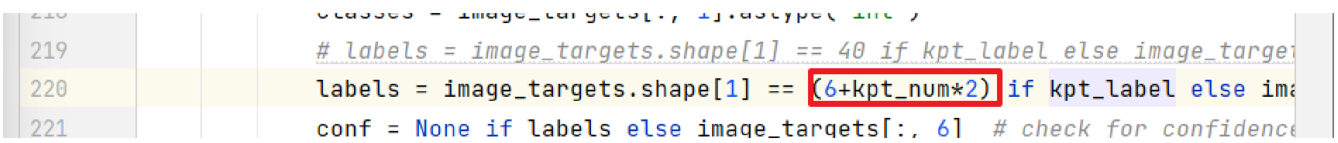


```python
def non_max_suppression_export(prediction, conf_thres=0.25, iou_thres=0.45, classes=None, agnostic=False, multi_label=False,
                               kpt_label=True, nc=None, nkpt=5, labels=()):
    """Runs Non-Maximum Suppression (NMS) on inference results

    Returns:
         list of detections, on (n,6) tensor per image [xyxy, conf, cls]
    """
    if nc is None:
        nc = prediction.shape[2] - 5  if not kpt_label else prediction.shape[2] - (5+3*nkpt) # number of classes

    min_wh, max_wh = 2, 4096  # (pixels) minimum and maximum box width and height
    xc = prediction[..., 4] > conf_thres  # candidates
    output = [torch.zeros((0, 57), device=prediction.device)] * prediction.shape[0]
    for xi, x in enumerate(prediction):  # image index, image inference
        x = x[xc[xi]]  # confidence
        # Compute conf
        cx, cy, w, h = x[:,0:1], x[:,1:2], x[:,2:3], x[:,3:4]
```

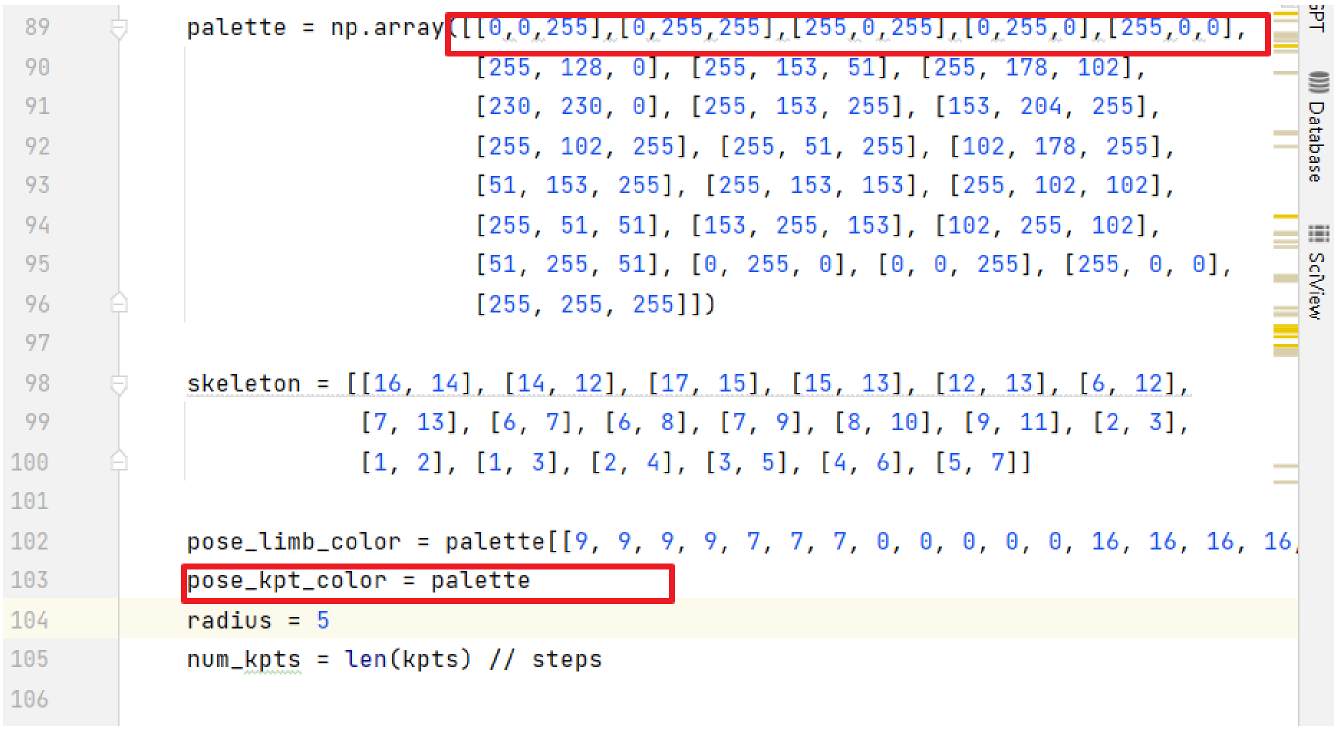
plot.py修改



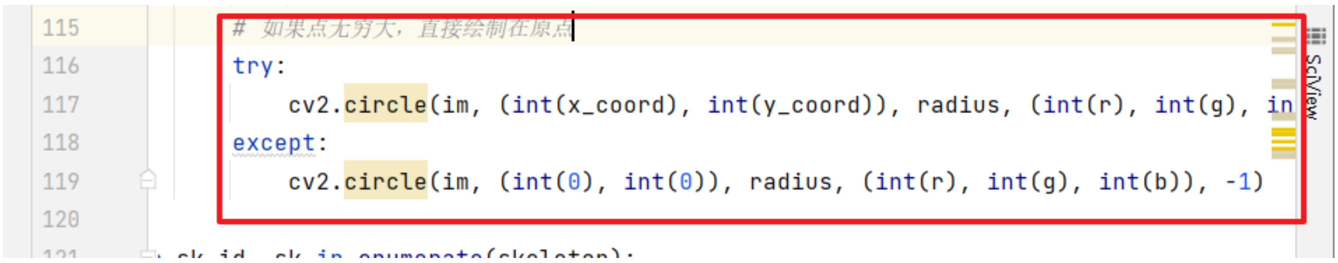
```python
plots=16, kpt_label=True, kpt_num=5, steps=2, orig_shape=None):
```



```python
        # labels = image_targets.shape[1] == 40 if kpt_label else image_target
        labels = image_targets.shape[1] == (6+kpt_num*2) if kpt_label else ima
        conf = None if labels else image_targets[:, 6]  # check for confidence
```



```python
    palette = np.array([[0,0,255],[0,255,255],[255,0,255],[0,255,0],[255,0,0],
                        [255, 128, 0], [255, 153, 51], [255, 178, 102],
                        [230, 230, 0], [255, 153, 255], [153, 204, 255],
                        [255, 102, 255], [255, 51, 255], [102, 178, 255],
                        [51, 153, 255], [255, 153, 153], [255, 102, 102],
                        [255, 51, 51], [153, 255, 153], [102, 255, 102],
                        [51, 255, 51], [0, 255, 0], [0, 0, 255], [255, 0, 0],
                        [255, 255, 255]])

    skeleton = [[16, 14], [14, 12], [17, 15], [15, 13], [12, 13], [6, 12],
                [7, 13], [6, 7], [6, 8], [7, 9], [8, 10], [9, 11], [2, 3],
                [1, 2], [1, 3], [2, 4], [3, 5], [4, 6], [5, 7]]

    pose_limb_color = palette[[9, 9, 9, 9, 7, 7, 7, 0, 0, 0, 0, 0, 16, 16, 16, 16,
    pose_kpt_color = palette
    radius = 5
    num_kpts = len(kpts) // steps

```



```python
        # 如果点无穷大，直接绘制在原点
        try:
            cv2.circle(im, (int(x_coord), int(y_coord)), radius, (int(r), int(g), in
        except:
            cv2.circle(im, (int(0), int(0)), radius, (int(r), int(g), int(b)), -1)

    for sk_id, sk in enumerate(skeleton):
```