1、训练环境

除了按照开源代码的环境要求外，还需要安装其他修改需要的库

训练前需要安装 mmcv-full

有四个模型放在 models 下面，大模型请参考 raftstereo-eth3d.pth，小模型参考 raftstereo-realtime.pth

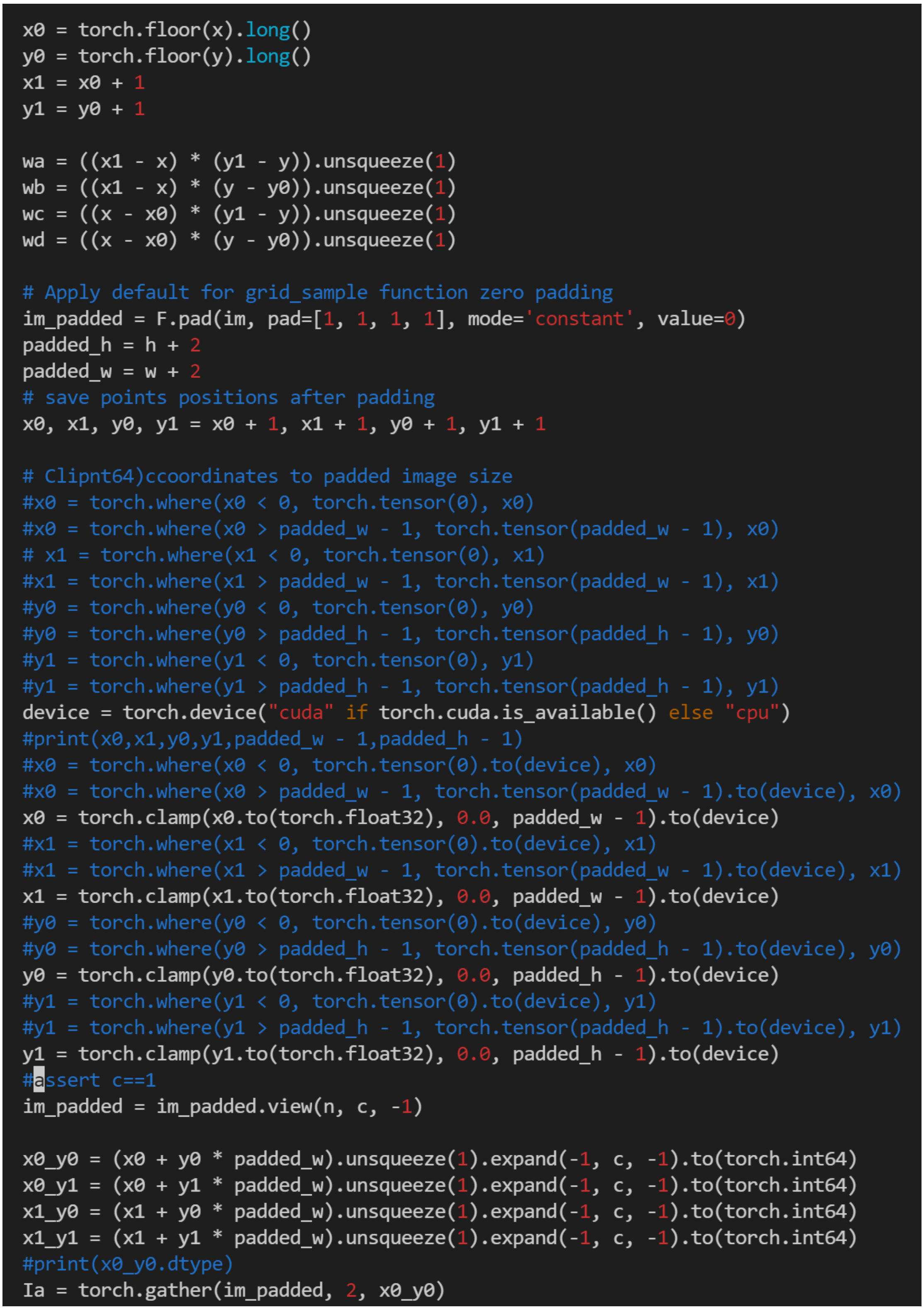
2、OP 修改点

主要是 Correlation volume 部分，还有其他少量的 GRU 部分

Correlation volume 部分：非 cuda 的实现主要由 PytorchAlternateCorrBlock1D 和 CorrBlock1D 实现，其他两个都是都是 cuda 接口实现，高通平台部署的模型必须使用非 cuda 的 op，其中非 cuda 的两个接口修改点主要如下：

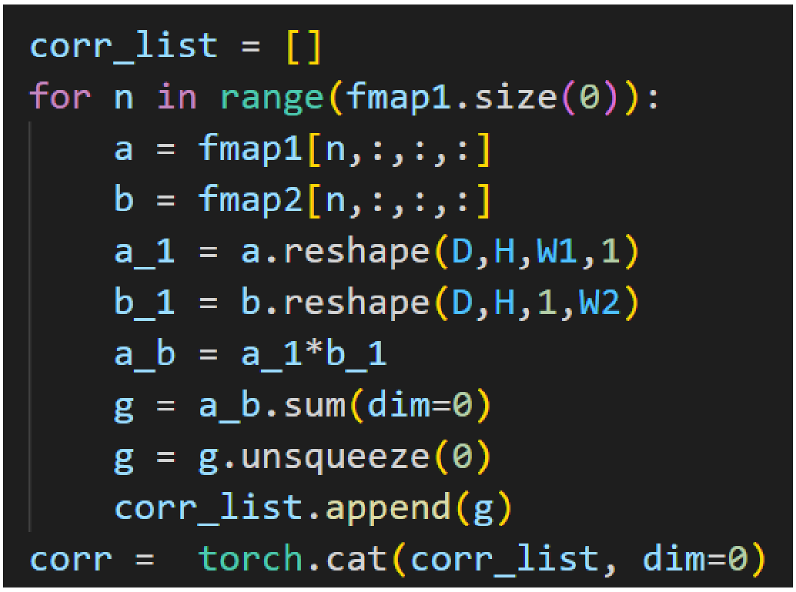
（1）F.grid_sample 用 mmcv.ops.point_sample 里面的 bilinear_grid_sample 替换

说明：打开安装 mmcv 的目录，在 mmcv/ops/point_sample.py 50 行代码开始按照下图修改



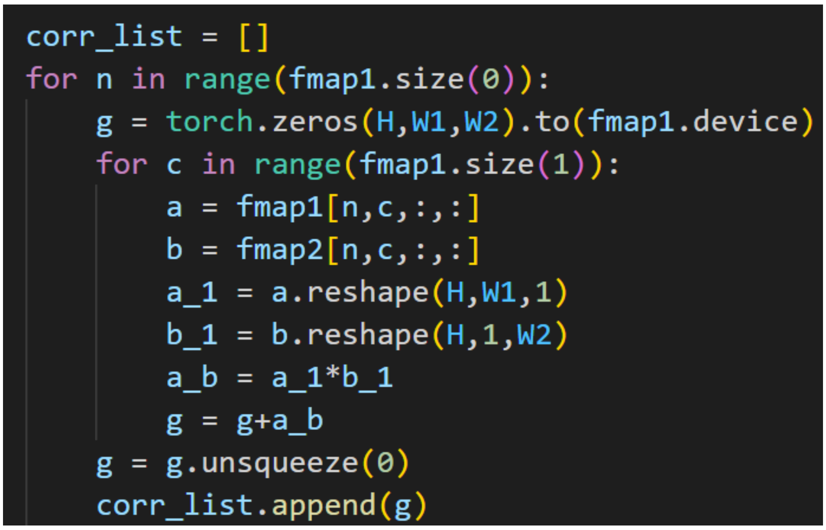
```
x0 = torch.floor(x).long()
y0 = torch.floor(y).long()
x1 = x0 + 1
y1 = y0 + 1

wa = ((x1 - x) * (y1 - y)).unsqueeze(1)
wb = ((x1 - x) * (y - y0)).unsqueeze(1)
wc = ((x - x0) * (y1 - y)).unsqueeze(1)
wd = ((x - x0) * (y - y0)).unsqueeze(1)

# Apply default for grid_sample function zero padding
im_padded = F.pad(im, pad=[1, 1, 1, 1], mode='constant', value=0)
padded_h = h + 2
padded_w = w + 2
# save points positions after padding
x0, x1, y0, y1 = x0 + 1, x1 + 1, y0 + 1, y1 + 1

# Clipnt64)ccoordinates to padded image size
#x0 = torch.where(x0 < 0, torch.tensor(0), x0)
#x0 = torch.where(x0 > padded_w - 1, torch.tensor(padded_w - 1), x0)
# x1 = torch.where(x1 < 0, torch.tensor(0), x1)
#x1 = torch.where(x1 > padded_w - 1, torch.tensor(padded_w - 1), x1)
#y0 = torch.where(y0 < 0, torch.tensor(0), y0)
#y0 = torch.where(y0 > padded_h - 1, torch.tensor(padded_h - 1), y0)
#y1 = torch.where(y1 < 0, torch.tensor(0), y1)
#y1 = torch.where(y1 > padded_h - 1, torch.tensor(padded_h - 1), y1)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#print(x0,x1,y0,y1,padded_w - 1,padded_h - 1)
#x0 = torch.where(x0 < 0, torch.tensor(0).to(device), x0)
#x0 = torch.where(x0 > padded_w - 1, torch.tensor(padded_w - 1).to(device), x0)
x0 = torch.clamp(x0.to(torch.float32), 0.0, padded_w - 1).to(device)
#x1 = torch.where(x1 < 0, torch.tensor(0).to(device), x1)
#x1 = torch.where(x1 > padded_w - 1, torch.tensor(padded_w - 1).to(device), x1)
x1 = torch.clamp(x1.to(torch.float32), 0.0, padded_w - 1).to(device)
#y0 = torch.where(y0 < 0, torch.tensor(0).to(device), y0)
#y0 = torch.where(y0 > padded_h - 1, torch.tensor(padded_h - 1).to(device), y0)
y0 = torch.clamp(y0.to(torch.float32), 0.0, padded_h - 1).to(device)
#y1 = torch.where(y1 < 0, torch.tensor(0).to(device), y1)
#y1 = torch.where(y1 > padded_h - 1, torch.tensor(padded_h - 1).to(device), y1)
y1 = torch.clamp(y1.to(torch.float32), 0.0, padded_h - 1).to(device)
#assert c==1
im_padded = im_padded.view(n, c, -1)

x0_y0 = (x0 + y0 * padded_w).unsqueeze(1).expand(-1, c, -1).to(torch.int64)
x0_y1 = (x0 + y1 * padded_w).unsqueeze(1).expand(-1, c, -1).to(torch.int64)
x1_y0 = (x1 + y0 * padded_w).unsqueeze(1).expand(-1, c, -1).to(torch.int64)
x1_y1 = (x1 + y1 * padded_w).unsqueeze(1).expand(-1, c, -1).to(torch.int64)
#print(x0_y0.dtype)
Ia = torch.gather(im_padded, 2, x0_y0)
```

（2）torch.einsum 由以下代码替代，当 tensor 内存占用低的时候可以用图 1，否则用图 2



```python
corr_list = []
for n in range(fmap1.size(0)):
    a = fmap1[n,:,:,:]
    b = fmap2[n,:,:,:]
    a_1 = a.reshape(D,H,W1,1)
    b_1 = b.reshape(D,H,1,W2)
    a_b = a_1*b_1
    g = a_b.sum(dim=0)
    g = g.unsqueeze(0)
    corr_list.append(g)
corr =  torch.cat(corr_list, dim=0)
```

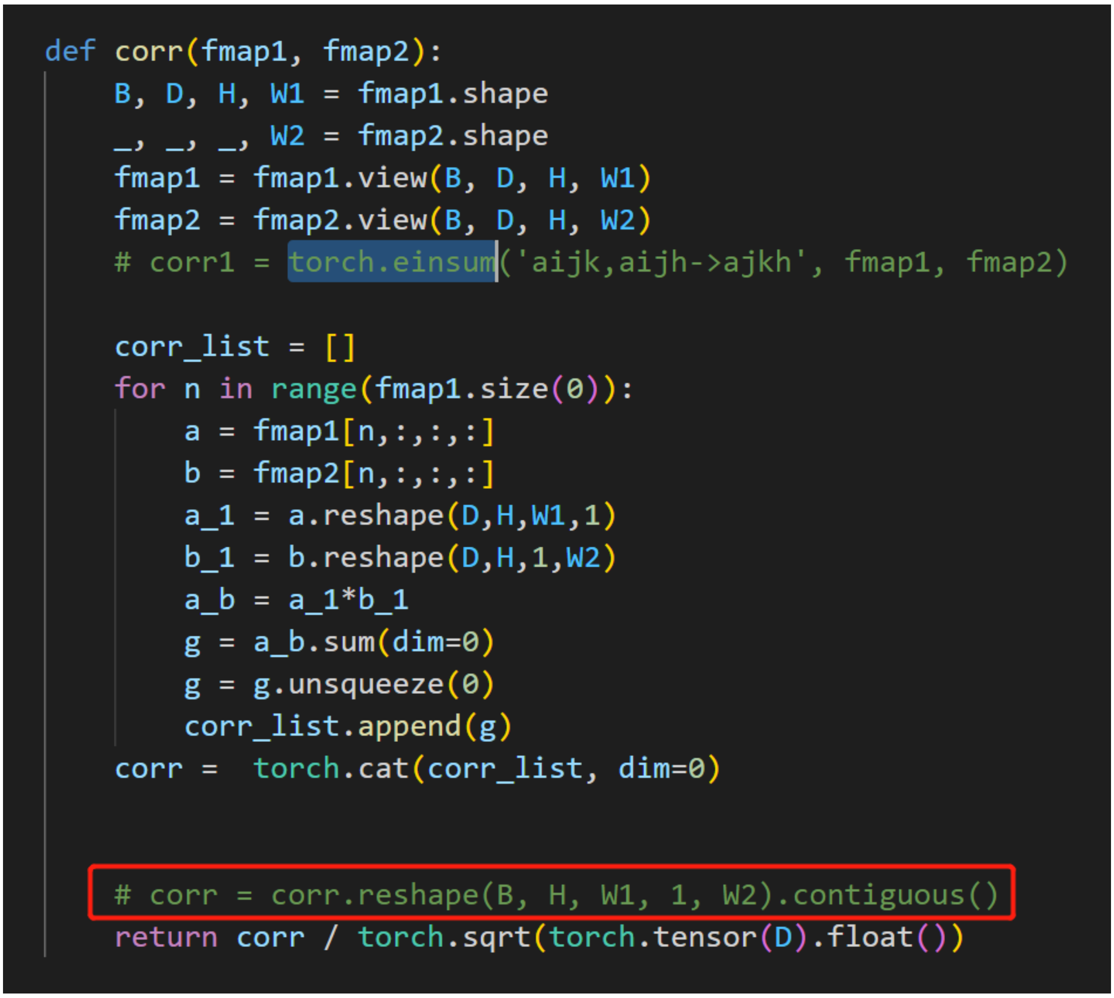
图 1



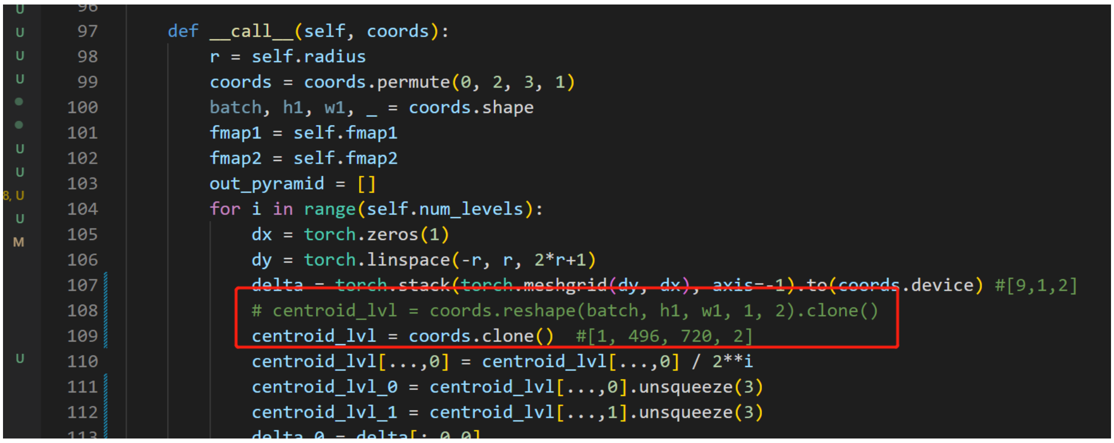
```python
corr_list = []
for n in range(fmap1.size(0)):
    g = torch.zeros(H,W1,W2).to(fmap1.device)
    for c in range(fmap1.size(1)):
        a = fmap1[n,c,:,:]
        b = fmap2[n,c,:,:]
        a_1 = a.reshape(H,W1,1)
        b_1 = b.reshape(H,1,W2)
        a_b = a_1*b_1
        g = g+a_b
    g = g.unsqueeze(0)
    corr_list.append(g)
```

图 2

（3）所有的 tensor 必须是 4D 以下（包括 4D）



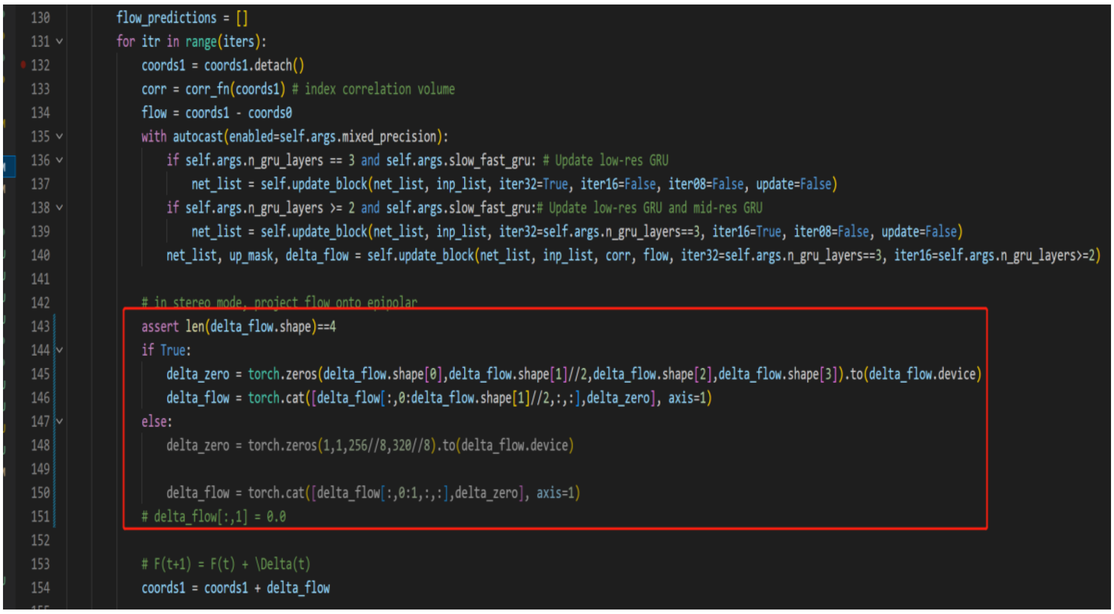
```python
def corr(fmap1, fmap2):
    B, D, H, W1 = fmap1.shape
    _, _, _, W2 = fmap2.shape
    fmap1 = fmap1.view(B, D, H, W1)
    fmap2 = fmap2.view(B, D, H, W2)
    # corr1 = torch.einsum('aijk,aijh->ajkh', fmap1, fmap2)

    corr_list = []
    for n in range(fmap1.size(0)):
        a = fmap1[n,:,:,:]
        b = fmap2[n,:,:,:]
        a_1 = a.reshape(D,H,W1,1)
        b_1 = b.reshape(D,H,1,W2)
        a_b = a_1*b_1
        g = a_b.sum(dim=0)
        g = g.unsqueeze(0)
        corr_list.append(g)
    corr =  torch.cat(corr_list, dim=0)


    # corr = corr.reshape(B, H, W1, 1, W2).contiguous()
    return corr / torch.sqrt(torch.tensor(D).float())
```



```python
    def __call__(self, coords):
        r = self.radius
        coords = coords.permute(0, 2, 3, 1)
        batch, h1, w1, _ = coords.shape
        fmap1 = self.fmap1
        fmap2 = self.fmap2
        out_pyramid = []
        for i in range(self.num_levels):
            dx = torch.zeros(1)
            dy = torch.linspace(-r, r, 2*r+1)
            delta = torch.stack(torch.meshgrid(dy, dx), axis=-1).to(coords.device) #[9,1,2]
            # centroid_lvl = coords.reshape(batch, h1, w1, 1, 2).clone()
            centroid_lvl = coords.clone()  #[1, 496, 720, 2]
            centroid_lvl[...,0] = centroid_lvl[...,0] / 2**i
            centroid_lvl_0 = centroid_lvl[...,0].unsqueeze(3)
            centroid_lvl_1 = centroid_lvl[...,1].unsqueeze(3)
            delta_0 = delta[:,0,0]
```

GRU 部分：主要是 scatterND 的移除和 upsample_flow 的修改，

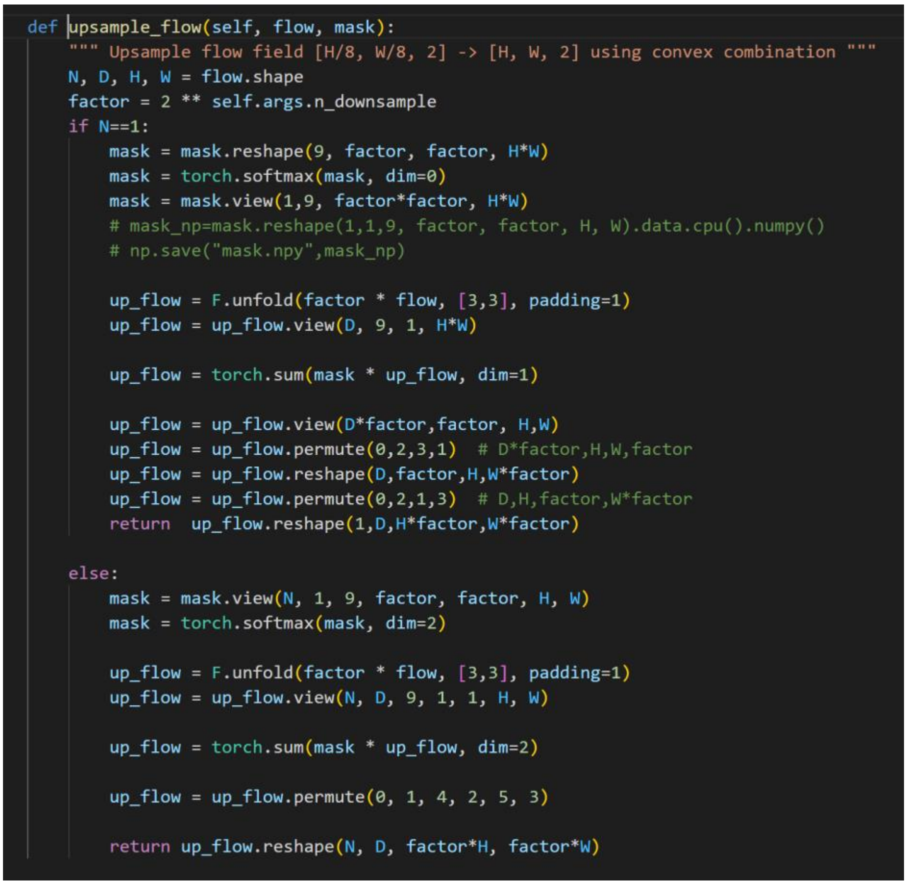
（1）scatterND 详细修改见下图



```
        flow_predictions = []
        for itr in range(iters):
            coords1 = coords1.detach()
            corr = corr_fn(coords1) # index correlation volume
            flow = coords1 - coords0
            with autocast(enabled=self.args.mixed_precision):
                if self.args.n_gru_layers == 3 and self.args.slow_fast_gru: # Update low-res GRU
                    net_list = self.update_block(net_list, inp_list, iter32=True, iter16=False, iter08=False, update=False)
                if self.args.n_gru_layers >= 2 and self.args.slow_fast_gru:# Update low-res GRU and mid-res GRU
                    net_list = self.update_block(net_list, inp_list, iter32=self.args.n_gru_layers==3, iter16=True, iter08=False, update=False)
                net_list, up_mask, delta_flow = self.update_block(net_list, inp_list, corr, flow, iter32=self.args.n_gru_layers==3, iter16=self.args.n_gru_layers>=2)

            # in stereo mode, project flow onto epipolar
            assert len(delta_flow.shape)==4
            if True:
                delta_zero = torch.zeros(delta_flow.shape[0],delta_flow.shape[1]//2,delta_flow.shape[2],delta_flow.shape[3]).to(delta_flow.device)
                delta_flow = torch.cat([delta_flow[:,0:delta_flow.shape[1]//2,:,:],delta_zero], axis=1)
            else:
                delta_zero = torch.zeros(1,1,256//8,320//8).to(delta_flow.device)

                delta_flow = torch.cat([delta_flow[:,0:1,:,:],delta_zero], axis=1)
            # delta_flow[:,1] = 0.0

            # F(t+1) = F(t) + \Delta(t)
            coords1 = coords1 + delta_flow
```

训练和测试的时候 if 后面请用 True，生成模型的时候请用 False，分辨率要手动修改设置，主要原因在于如果用自动生成，会有很多多余的 shape 操作，模型转换就会报错

（2）upsample_flow 的修改见下图，主要是去掉 4D 以上的操作，当输入样本数为 1 的时候，采用 4D 以下的 op，训练时候可以采用原始代码



```
def upsample_flow(self, flow, mask):
    """ Upsample flow field [H/8, W/8, 2] -> [H, W, 2] using convex combination """
    N, D, H, W = flow.shape
    factor = 2 ** self.args.n_downsample
    if N==1:
        mask = mask.reshape(9, factor, factor, H*W)
        mask = torch.softmax(mask, dim=0)
        mask = mask.view(1,9, factor*factor, H*W)
        # mask_np=mask.reshape(1,1,9, factor, factor, H, W).data.cpu().numpy()
        # np.save("mask.npy",mask_np)

        up_flow = F.unfold(factor * flow, [3,3], padding=1)
        up_flow = up_flow.view(D, 9, 1, H*W)

        up_flow = torch.sum(mask * up_flow, dim=1)

        up_flow = up_flow.view(D*factor,factor, H,W)
        up_flow = up_flow.permute(0,2,3,1)  # D*factor,H,W,factor
        up_flow = up_flow.reshape(D,factor,H,W*factor)
        up_flow = up_flow.permute(0,2,1,3)  # D,H,factor,W*factor
        return  up_flow.reshape(1,D,H*factor,W*factor)

    else:
        mask = mask.view(N, 1, 9, factor, factor, H, W)
        mask = torch.softmax(mask, dim=2)

        up_flow = F.unfold(factor * flow, [3,3], padding=1)
        up_flow = up_flow.view(N, D, 9, 1, 1, H, W)

        up_flow = torch.sum(mask * up_flow, dim=2)

        up_flow = up_flow.permute(0, 1, 4, 2, 5, 3)

        return up_flow.reshape(N, D, factor*H, factor*W)
```

3、脚本说明

Run 里面的 start.sh 主要用于单帧测试，toOnnx_realtime.sh 主要用于 pth 模型转 onnx，infer_piliang.sh 主要用于数据批量测试且生成效果视频