**Appendix A**

For the panchromatic image, the input data was a 240 × 240 image patch, and the input data was a 20×20 image patch for the hyperspectral image. Then we had total number of experimental image batches as 78400. We spat the experimental image batches with 50176 as the train data, and spat 12544 as validation data, and spat 15680 as the test data. For model training, when the model was initializing, we set the weight of each convolutional layer with xavier normalization, and the bias of each convolutional layer was set as zero, and the batch size was set as 32 in our experiments. We trained our model with the Pytorch platform. The loss function for our network was the Root Mean Square Error (RMSE). The optimizer for our network was the Adam optimizer. It’s important to emphasize that the model should be trained with a “NVIDIA GeForce RTX 3080Ti graphics card”. Otherwise, you can’t run the code, and the result will be different.

We organize the data as the following code:

**import** os  
os.environ[**"KMP\_DUPLICATE\_LIB\_OK"**] = **"TRUE"  
import** scipy.io **as** scio  
**import** h5py  
**import** torch  
**import** torch.nn **as** nn  
**import** time  
**import** numpy **as** np  
**import** random  
**import** torch.optim **as** optim  
**from** torch.nn **import** init  
**from** skimage.transform **import** resize  
**from** PIL **import** Image

**def** dataset\_preprocessing(pan\_filename,  
 pan\_key,  
 his\_filename,  
 his\_key,  
 w):  
  
 pan = scio.loadmat(pan\_filename)[pan\_key]  
 pan = pan.reshape(pan.shape[0], pan.shape[1], 1)  
 his = scio.loadmat(his\_filename)[his\_key]  
 target = np.zeros((pan.shape[0],pan.shape[1],his.shape[2]),dtype=np.float)  
 **for** i **in** range(his.shape[2]):  
 his\_img = Image.fromarray(np.uint8(his[:, :, i]))  
 target\_img = his\_img.resize((his\_img.width \* 12, his\_img.height \* 12),Image.ANTIALIAS)  
 target[:,:,i] = np.asarray(target\_img)  
 pan\_train\_data = []  
 his\_train\_data = []  
 target\_train\_data = []  
  
 **for** i **in** range(0, his.shape[0] - 1, w):  
 **for** j **in** range(0, his.shape[1] - 1, w):  
 tmp = his[i:i + w, j:j + w]  
 tmp = tmp.transpose(2, 0, 1)  
 his\_train\_data.append(tmp)  
 tmp = pan[12 \* i:12 \* (i + w), 12 \* j:12 \* (j + w)]  
 tmp = tmp.transpose(2, 0, 1)  
 pan\_train\_data.append(tmp)  
 tmp = target[i \* 12:(i + w) \* 12, j \* 12:(j + w) \* 12]  
 target\_train\_data.append(tmp)  
  
 pan\_train\_data = np.array(pan\_train\_data).astype(float)  
 his\_train\_data = np.array(his\_train\_data).astype(float)  
 target\_train\_data = np.array(target\_train\_data).astype(float)  
  
 dict\_train = {**'pan'**:pan\_train\_data,  
 **'his'**:his\_train\_data,  
 **'target'**:target\_train\_data}  
  
 pan\_test\_data = []  
 his\_test\_data = []  
 target\_test\_data = []  
  
 **for** i **in** range(0, his.shape[0] - 1, w):  
 **for** j **in** range(int(his.shape[1] \* 0.8), his.shape[1] - 1, w):  
 tmp = his[i:i + w, j:j + w]  
 tmp = tmp.transpose(2, 0, 1)  
 his\_test\_data.append(tmp)  
 tmp = pan[12 \* i:12 \* (i + w), 12 \* j:12 \* (j + w)]  
 tmp = tmp.transpose(2, 0, 1)  
 pan\_test\_data.append(tmp)  
 tmp = target[i \* 12:(i + w) \* 12, j \* 12:(j + w) \* 12]  
 target\_test\_data.append(tmp)  
  
 pan\_test\_data = np.array(pan\_test\_data).astype(float)  
 his\_test\_data = np.array(his\_test\_data).astype(float)  
 target\_test\_data = np.array(target\_test\_data).astype(float)  
  
 dict\_test = {**'pan'**:pan\_test\_data,  
 **'his'**:his\_test\_data,  
 **'target'**:target\_test\_data}  
  
 dict\_Dataset = {**'train'**:dict\_train,  
 **'test'**:dict\_test}  
  
 **return** dict\_Dataset

Taking the Dianchi dataset as an example, we train the data as the following code:

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
  
 dict\_Dataset = dataset\_preprocessing(pan\_filename=**'dc\_pan.mat'**,  
 pan\_key=**'dc\_pan'**,  
 his\_filename=**'dc\_hyper.mat'**,  
 his\_key=**'dc\_hyper'**,  
 w=20)  
  
 net = My\_NN()  
 LR = 0.0005  
  
 **for** m **in** net.modules():  
 **if** isinstance(m, nn.Conv2d):  
 init.xavier\_normal(m.weight.data)  
  
 num\_epochs = 1500  
 batch\_size = 16  
 criterion = nn.MSELoss(reduce=**True**, size\_average=**True**)  
 optimizer = optim.Adam(net.parameters(), lr=LR)  
 scheduler = optim.lr\_scheduler.CosineAnnealingLR(optimizer, T\_max=10, eta\_min=5e-6)  
  
 losses = []  
 time\_start = time.time()  
 train\_loss\_min = 99999  
 test\_loss\_min = 99999  
 **for** epoch **in** range(num\_epochs):  
  
 batch\_loss = []  
  
 **for** start **in** range(0, len(dict\_Dataset[**'train'**][**'pan'**]), batch\_size):  
 end = start + batch\_size **if** start + batch\_size < len(dict\_Dataset[**'train'**][**'pan'**]) **else** len(  
 dict\_Dataset[**'train'**][**'pan'**])  
 pan\_batch = torch.tensor(dict\_Dataset[**'train'**][**'pan'**][start:end], dtype=torch.float)  
 his\_batch = torch.tensor(dict\_Dataset[**'train'**][**'his'**][start:end], dtype=torch.float)  
 target\_batch = torch.tensor(dict\_Dataset[**'train'**][**'target'**][start:end], dtype=torch.float)  
 prediction = net(his\_batch, pan\_batch)  
 loss = criterion(prediction, target\_batch)  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
 batch\_loss.append(loss.data.numpy())  
  
 scheduler.step()  
  
 **if** train\_loss\_min > np.mean(np.sqrt(batch\_loss)):  
 train\_loss\_min = np.mean(np.sqrt(batch\_loss))  
 torch.save(net, **'my\_nn\_ende\_dc\_1.pkl'**)  
  
 time\_end = time.time()  
 print(**'epoch: '**, epoch, **', loss: '**, np.mean(np.sqrt(batch\_loss)), **', time cost: '**,  
 time\_end - time\_start, **', loss min: '**, train\_loss\_min)  
 time\_start = time.time()

For generate the result, please follow the following code:

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

net = torch.load(**'my\_nn\_ende\_dc\_1.pkl'**)

w = 20  
  
 pan = scio.loadmat(**'dianchi\_pan.mat'**)[**'dc\_pan'**]  
 pan = pan.reshape(1, pan.shape[0], pan.shape[1], 1)  
 pan = pan.astype(float)  
 his = scio.loadmat(**'dianchi\_hyper.mat'**)[**'dc\_hyper'**]  
 his = his.reshape(1, his.shape[0], his.shape[1], his.shape[2])  
 his = his.astype(float)  
  
 output = np.zeros((his.shape[3], pan.shape[1], pan.shape[2]), dtype=np.float32)  
 output\_div = np.zeros((his.shape[3], pan.shape[1], pan.shape[2]), dtype=np.float32)  
 print(output.shape)  
  
 **for** i **in** range(his.shape[1] - w + 1):  
 **for** j **in** range(his.shape[2] - w + 1):  
 time\_start = time.time()  
 his\_w = his[:, i:i + w, j:j + w, :]  
 his\_w = his\_w.swapaxes(1, 3)  
 his\_w = his\_w.swapaxes(2, 3)  
 his\_w = torch.from\_numpy(his\_w).float()  
 pan\_w = pan[:, 12 \* i:12 \* (i + w), 12 \* j:12 \* (j + w), :]  
 pan\_w = pan\_w.swapaxes(1, 3)  
 pan\_w = pan\_w.swapaxes(2, 3)  
 pan\_w = torch.from\_numpy(pan\_w).float()  
 output[:, 12 \* i:12 \* (i + w), 12 \* j:12 \* (j + w)] = output[:, 12 \* i:12 \* (i + w),  
 12 \* j:12 \* (j + w)] + net(his\_w, pan\_w).data.numpy()  
 output\_div[:, 12 \* i:12 \* (i + w), 12 \* j:12 \* (j + w)] = output\_div[:, 12 \* i:12 \* (i + w),  
 12 \* j:12 \* (j + w)] + 1  
 time\_end = time.time()  
 print(i, **' '**, j, **' '**, time\_end - time\_start)  
  
 output = output.swapaxes(0, 2)  
 output = output.swapaxes(0, 1)  
 output\_div = output\_div.swapaxes(0, 2)  
 output\_div = output\_div.swapaxes(0, 1)  
  
 **for** i **in** range(output.shape[0]):  
 **for** j **in** range(output.shape[1]):  
 **for** k **in** range(output.shape[2]):  
 output[i, j, k] = output[i, j, k] / output\_div[i, j, k]  
  
 scio.savemat(**'dc\_fusion\_ende\_1\_1.mat'**, {**'dc\_fusion'**: output})