MLMAPPER USING NEURAL NETWORK

In [1]:

```python
#import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch.utils.data import random_split
from torch.utils.data import DataLoader,Dataset
import torch.nn.functional as F
from torch.utils.tensorboard import SummaryWriter
from sklearn.metrics import r2_score
import cv2
import os
```

In [2]:

```python
#getting the path to the folder
dir=os.getcwd()
```

DATA PATH

In [3]:

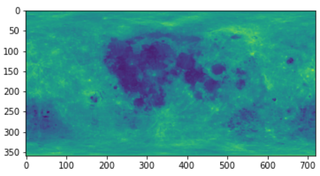
```python
#paths to data
albedo=dir+"/DATA/Data_Albedo/Albedo_Map.csv"
LPFe_Map=dir+"/DATA/Data_Albedo/LPFe_Map.csv"
LPK_Map=dir+"/DATA/Data_Albedo/LPK_Map.csv"
LPTh_Map=dir+"/DATA/Data_Albedo/LPTh_Map.csv"
LPTi_Map=dir+"/DATA/Data_Albedo/LPTi_Map.csv"
```

DATA VISUALIZATION:

In [14]:

```python
#lunar albedo map
albedo_img=np.array(pd.read_csv(albedo))
print(np.shape(albedo_img))
a=albedo_img[:,360:720]
print(np.shape(a))
plt.imshow(albedo_img)
plt.show()
```

```
(359, 720)
(359, 360)
```



In [6]:

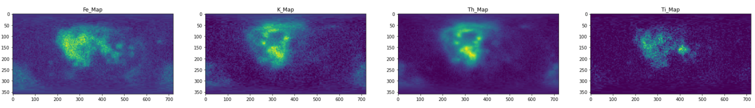
```python
#chemical composition map of the moon surface
fe_map_img=np.array(pd.read_csv(LPFe_Map))
K_map_img=np.array(pd.read_csv(LPK_Map))
Th_map_img=np.array(pd.read_csv(LPTh_Map))
Ti_map_img=np.array(pd.read_csv(LPTi_Map))

ffig, faxes = plt.subplots(1,4 , figsize=(30,30))
faxes[0].imshow(fe_map_img)
faxes[0].set_title("Fe_Map")
faxes[0].grid(False)

faxes[1].imshow(K_map_img)
faxes[1].set_title("K_Map")
faxes[1].grid(False)

faxes[2].imshow(Th_map_img)
faxes[2].set_title("Th_Map")
faxes[2].grid(False)

faxes[3].imshow(Ti_map_img)
faxes[3].set_title("Ti_Map")
faxes[3].grid(False)
```



CUSTOM DATASET TO GET INPUT,GT

In [4]:

```python
#custom dataset for nearby pixels
class dataset_nearby_pixel(Dataset):
    def __init__(self,path_1,path_2,path_3,path_4,path_5,mode):
        self.path_1=path_1
        self.path_2=path_2
        self.path_3=path_3
        self.path_4=path_4
        self.path_5=path_5
        self.mode=mode
        #converting the data to ndarray
        self.X_1=np.array(pd.read_csv(path_1))
        self.X_2=np.array(pd.read_csv(path_2))
        self.X_3=np.array(pd.read_csv(path_3))
        self.X_4=np.array(pd.read_csv(path_4))
        self.X_5=cv2.GaussianBlur(np.array(pd.read_csv(path_5)),ksize=(0,0),sigmaX=9)
        self.X=[]
        self.Y=[]
        n,m=np.shape(self.X_5)[0],np.shape(self.X_5)[1]
        for i in range(1,n-1):
            for j in range(1,m-1):
                nearby=[]
                for chem in [self.X_1,self.X_2,self.X_3,self.X_4]:
                    for a in [-1,0,1]:
                        for b in [-1,0,1]:
                            nearby.append(chem[i+a][j+b])
                self.X.append(nearby)
                self.Y.append([self.X_5[i][j]])
        l=len(self.Y)//2
        if mode == "Train" or "train":
            self.X=np.array(self.X)[:l]
            self.Y=np.array(self.Y)[:l]
        elif mode== "Test" or "test":
            self.X=np.array(self.X)[l:]
            self.Y=np.array(self.Y)[l:]
    def __len__(self):
        self.filelength=np.shape(self.Y)[0]
        return self.filelength

    def __getitem__(self,idx):
        return torch.from_numpy(self.X[idx]),torch.from_numpy(self.Y[idx])
```

In [8]:

```python
#custom dataset
class dataset(Dataset):
  def __init__(self,path_1,path_2,path_3,path_4,path_5,start_split,end_split):
    self.path_1=path_1
    self.path_2=path_2
    self.path_3=path_3
    self.path_4=path_4
    self.path_5=path_5
    self.start_split=start_split
    self.end_split=end_split
    #converting the data to tensors
    self.X_1=torch.FloatTensor(np.array(pd.read_csv(path_1)))
    self.X_2=torch.FloatTensor(np.array(pd.read_csv(path_2)))
    self.X_3=torch.FloatTensor(np.array(pd.read_csv(path_3)))
    self.X_4=torch.FloatTensor(np.array(pd.read_csv(path_4)))
    self.X_5=torch.FloatTensor(cv2.GaussianBlur(np.array(pd.read_csv(path_5)),ksize=(0,0),sigmaX=9))
    #normalizing the data
    self.X_1=(self.X_1[:,self.start_split:self.end_split].flatten()-torch.mean(self.X_1[:,self.start_split:self
    self.X_2=(self.X_2[:,self.start_split:self.end_split].flatten()-torch.mean(self.X_2[:,self.start_split:self
    self.X_3=(self.X_3[:,self.start_split:self.end_split].flatten()-torch.mean(self.X_3[:,self.start_split:self
    self.X_4=(self.X_4[:,self.start_split:self.end_split].flatten()-torch.mean(self.X_4[:,self.start_split:self
    self.X_5=self.X_5[:,self.start_split:self.end_split].flatten()
    self.X=torch.stack((self.X_1,self.X_2,self.X_3,self.X_4),1)
    self.Y=self.X_5

  def __len__(self):
    self.filelength=len(self.Y)
    return self.filelength

  def __getitem__(self,idx):
    return self.X[idx],self.Y[idx]
```

NEURAL NETWORK MODEL

In [5]:

```python
#neural network model
class model(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(model, self).__init__()
        self.hidden1 = torch.nn.Linear(n_feature, n_hidden)
        self.hidden2 = torch.nn.Linear(n_hidden,n_hidden)
        self.hidden3 = torch.nn.Linear(n_hidden,n_hidden)
        self.predict = torch.nn.Linear(n_hidden, n_output)
        self.dropout = torch.nn.Dropout(p=0.2)

    def forward(self, x):
        x = self.dropout(F.relu(self.hidden1(x)))
        x = self.dropout(F.relu(self.hidden2(x)))
        x = self.dropout(F.relu(self.hidden3(x)))
        x = self.predict(x)
        return x
```

Train Function:

In [10]:

```python
class train():
    def __init__(self,batch_size,epochs,lr,train_val_split,scheduler,near):
        self.batch_size=batch_size
        self.scheduler=scheduler
        self.epochs=epochs
        self.lr=lr
        self.train_val_split=train_val_split
        self.near=near
        if self.near== True:
            self.data=dataset_nearby_pixel(LPFe_Map,LPK_Map,LPTh_Map,LPTi_Map,albedo,mode="train")
            self.train_data,self.val_data=random_split(self.data,[len(self.data)-int(self.train_val_split*len(s
            self.train_loader=DataLoader(self.train_data,batch_size=self.batch_size,shuffle=True)
            self.val_loader=DataLoader(self.val_data,batch_size=self.batch_size,shuffle=True)
            self.net=model(n_feature=36, n_hidden=25, n_output=1)
        else:
            self.data=dataset(LPFe_Map,LPK_Map,LPTh_Map,LPTi_Map,albedo,0,360)
            self.train_data,self.val_data=random_split(self.data,[len(self.data)-int(self.train_val_split*len(s
            self.train_loader=DataLoader(self.train_data,batch_size=self.batch_size,shuffle=True)
            self.val_loader=DataLoader(self.val_data,batch_size=self.batch_size,shuffle=True)
            self.net=model(n_feature=4, n_hidden=4, n_output=1)
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=self.lr)
        if self.scheduler==True:
            self.sched=torch.optim.lr_scheduler.ExponentialLR(self.optimizer,gamma=0.7)
        self.loss_func = torch.nn.MSELoss()
        self.writer = SummaryWriter()

    def trainer(self):
        self.net=self.net.train()
        self.net=self.net.cuda()
        for epoch in range(self.epochs):
            for input,gt in self.train_loader:
                input = input.cuda()
                gt = gt.cuda()
                gt=torch.reshape(gt,(len(gt),1))
                output = self.net(input.float())
                loss = self.loss_func(output, gt.float())
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()
            if self.scheduler == True:
                self.sched.step()
            print('Epoch : {},  train loss : {}'.format(epoch+1,loss.item()))
            with torch.no_grad():
                for input,gt in self.val_loader:
                    input=input.cuda()
                    gt= gt.cuda()
                    gt=torch.reshape(gt,(len(gt),1))
                    val_output = self.net(input.float())
                    val_loss = self.loss_func(val_output,gt.float())
            print('Epoch : {},  val_loss : {}'.format(epoch+1,val_loss.item()))
            self.writer.add_scalar("Loss/train", loss, epoch)
            self.writer.add_scalar("Loss/val", val_loss, epoch)
            if self.scheduler == True:
```

```
                self.writer.add_scalar("lr/epoch",self.lr,epoch)
        torch.save(self.net.state_dict(),f"albedo_{self.epochs}_{self.lr}_{self.batch_size}.pth")
```

Normal Training

In [ ]:

```
train_best=train(batch_size=64,epochs=100,lr=0.0001,train_val_split=0.3,scheduler=False,near=False)
train_best.trainer()
```

Nearby Training

In [11]:

```
train_best_near=train(batch_size=64,epochs=100,lr=0.0001,train_val_split=0.3,scheduler=False,near=True)
train_best_near.trainer()
```

```
Epoch : 1,  train loss : 3.663121461868286
Epoch : 1,  val_loss : 6.235400199890137
Epoch : 2,  train loss : 0.7315381765365601
Epoch : 2,  val_loss : 3.8570151329040527
Epoch : 3,  train loss : 0.07767129689455032
Epoch : 3,  val_loss : 0.2592722773551941
Epoch : 4,  train loss : 0.10861585289239883
Epoch : 4,  val_loss : 0.11481534689664841
Epoch : 5,  train loss : 0.008385946974158287
Epoch : 5,  val_loss : 0.017458129674196243
Epoch : 6,  train loss : 0.005931203253567219
Epoch : 6,  val_loss : 0.003923547919839621
Epoch : 7,  train loss : 0.0025380842853337526
Epoch : 7,  val_loss : 0.002941682701930403 7
Epoch : 8,  train loss : 0.004413996357470751
Epoch : 8,  val_loss : 0.0031808745115995407
Epoch : 9,  train loss : 0.012463897466659546
Epoch : 9,  val_loss : 0.0034480565227568 15
Epoch : 10,  train loss : 0.0019360400037840009
Epoch : 10,  val_loss : 0.0037086177617311478
Epoch : 11,  train loss : 0.0040307193994522095
Epoch : 11,  val_loss : 0.010060365311801434
Epoch : 12,  train loss : 0.002174395602196455
Epoch : 12,  val_loss : 0.00344918854534626
Epoch : 13,  train loss : 0.0023952273186296225
Epoch : 13,  val_loss : 0.0030418103560805 32
Epoch : 14,  train loss : 0.0017142867436632514
Epoch : 14,  val_loss : 0.0021764987614005804
Epoch : 15,  train loss : 0.0014936740044504404
Epoch : 15,  val_loss : 0.0022012456320226192
Epoch : 16,  train loss : 0.003076739376410842
Epoch : 16,  val_loss : 0.0009612948633730412
Epoch : 17,  train loss : 0.0015559596940875053
Epoch : 17,  val_loss : 0.0025409129448235035
Epoch : 18,  train loss : 0.0015360764227807522
Epoch : 18,  val_loss : 0.0018302740063518286
Epoch : 19,  train loss : 0.001237700693309307
Epoch : 19,  val_loss : 0.0019883476197719574
Epoch : 20,  train loss : 0.0021255577448755503
Epoch : 20,  val_loss : 0.0022203782573342323
Epoch : 21,  train loss : 0.0016786059131845832
Epoch : 21,  val_loss : 0.0010781113523989916
Epoch : 22,  train loss : 0.0017608670750632882
Epoch : 22,  val_loss : 0.0012630890123546124
Epoch : 23,  train loss : 0.0010691287461668253
Epoch : 23,  val_loss : 0.0013512464938685298
Epoch : 24,  train loss : 0.0014031454920768738
Epoch : 24,  val_loss : 0.0012054055696353316
Epoch : 25,  train loss : 0.000898982398211956
Epoch : 25,  val_loss : 0.0012353716883808374
Epoch : 26,  train loss : 0.0011798352934420109
Epoch : 26,  val_loss : 0.001084773102775216
Epoch : 27,  train loss : 0.0014434707118198276
Epoch : 27,  val_loss : 0.0014055464416742325
Epoch : 28,  train loss : 0.0011658326257020235
Epoch : 28,  val_loss : 0.0011192505480721593
Epoch : 29,  train loss : 0.000996971968561411
Epoch : 29,  val_loss : 0.0010654942598193884
Epoch : 30,  train loss : 0.0009139752364717424
Epoch : 30,  val_loss : 0.001506188651546836
Epoch : 31,  train loss : 0.0011110090417787433
Epoch : 31,  val_loss : 0.0016047991812229156
Epoch : 32,  train loss : 0.000691440945956856
Epoch : 32,  val_loss : 0.0007233612122945487
Epoch : 33,  train loss : 0.0012839565752074122
Epoch : 33,  val_loss : 0.000796501524746418
Epoch : 34,  train loss : 0.0012452957453206182
Epoch : 34,  val_loss : 0.0008774904999881983
Epoch : 35,  train loss : 0.0011977748945355415
Epoch : 35,  val_loss : 0.0013015763834118843
Epoch : 36,  train loss : 0.0007563638500869274
Epoch : 36,  val_loss : 0.0005425375420600176
Epoch : 37,  train loss : 0.0013771128142252564
Epoch : 37,  val_loss : 0.0011425369884818792
Epoch : 38,  train loss : 0.0005532536306418478
Epoch : 38,  val_loss : 0.0005635066190734506
Epoch : 39,  train loss : 0.0007006324594840407
Epoch : 39,  val_loss : 0.0016416457947343588
Epoch : 40,  train loss : 0.001068621058948338
Epoch : 40,  val_loss : 0.0008183511672541499
Epoch : 41,  train loss : 0.0010209355968981981
Epoch : 41,  val_loss : 0.0005957492394372821
Epoch : 42,  train loss : 0.0007886828389018774
Epoch : 42,  val_loss : 0.0009132841369137168
Epoch : 43,  train loss : 0.0012918113498017192
Epoch : 43,  val_loss : 0.0004883845103904605
Epoch : 44,  train loss : 0.001936109852977097
Epoch : 44,  val_loss : 0.001604806398972869
Epoch : 45,  train loss : 0.0011068286839872599
Epoch : 45,  val_loss : 0.0008437742362730205
Epoch : 46,  train loss : 0.00047956325579434633
Epoch : 46,  val_loss : 0.0008930325857363641
Epoch : 47,  train loss : 0.0005946513847447932
Epoch : 47,  val_loss : 0.0014873286709189415
Epoch : 48,  train loss : 0.0009564015781506896
Epoch : 48,  val_loss : 0.0009345290018245578
Epoch : 49,  train loss : 0.0008203402976505458
Epoch : 49,  val_loss : 0.0006639375351369381
Epoch : 50,  train loss : 0.0006902476889081299
Epoch : 50,  val_loss : 0.0004883912042714655
Epoch : 51,  train loss : 0.0004926125984638929
Epoch : 51,  val_loss : 0.0005961637943983078
Epoch : 52,  train loss : 0.0010780099546536803
Epoch : 52,  val_loss : 0.0004606142174452543
Epoch : 53,  train loss : 0.000989591353572905
Epoch : 53,  val_loss : 0.000580992316827178
Epoch : 54,  train loss : 0.0011824460234493017
Epoch : 54,  val_loss : 0.000974780588876456
Epoch : 55,  train loss : 0.0006822660798206925
Epoch : 55,  val_loss : 0.0004125862615182996
Epoch : 56,  train loss : 0.0008135645766742527
Epoch : 56,  val_loss : 0.0006902160821482539
Epoch : 57,  train loss : 0.0009542423649691045
Epoch : 57,  val_loss : 0.00039271762943826616
Epoch : 58,  train loss : 0.0008922462584450841
Epoch : 58,  val_loss : 0.0007991312886588275
Epoch : 59,  train loss : 0.0006207296391949058
Epoch : 59,  val_loss : 0.0007134519401006401
Epoch : 60,  train loss : 0.0009470575605519116
Epoch : 60,  val_loss : 0.000669115805067122
Epoch : 61,  train loss : 0.0007827691151760519
Epoch : 61,  val_loss : 0.0007869868422858417
Epoch : 62,  train loss : 0.000519180262926966
Epoch : 62,  val_loss : 0.0007944981334730983
Epoch : 63,  train loss : 0.0008018278749659657
Epoch : 63,  val_loss : 0.0008610078948549926
Epoch : 64,  train loss : 0.0009551187395118177
Epoch : 64,  val_loss : 0.0013211076147854328
Epoch : 65,  train loss : 0.0005195135017856956
Epoch : 65,  val_loss : 0.0005760446656495333
Epoch : 66,  train loss : 0.0003230560396332294
Epoch : 66,  val_loss : 0.0009266051347367465
Epoch : 67,  train loss : 0.000610663671977818
Epoch : 67,  val_loss : 0.0005076468805782497
Epoch : 68,  train loss : 0.0008647426147945225
Epoch : 68,  val_loss : 0.001021400559693575
Epoch : 69,  train loss : 0.001387079362757504
Epoch : 69,  val_loss : 0.0010973482858389616
Epoch : 70,  train loss : 0.0004339746665209532
Epoch : 70,  val_loss : 0.0007570530287921429
Epoch : 71,  train loss : 0.00159328687004745
Epoch : 71,  val_loss : 0.0005887422594241798
Epoch : 72,  train loss : 0.0007722012815065682
Epoch : 72,  val_loss : 0.0005130531499162316
Epoch : 73,  train loss : 0.0018175251316279173
Epoch : 73,  val_loss : 0.0004409474495332688
Epoch : 74,  train loss : 0.00042749271960929036
Epoch : 74,  val_loss : 0.001030655112117529
Epoch : 75,  train loss : 0.0007682000286877155
Epoch : 75,  val_loss : 0.0010296714026480913
Epoch : 76,  train loss : 0.00037100250483490527
Epoch : 76,  val_loss : 0.0008103959262371063
Epoch : 77,  train loss : 0.0005352107109501958
Epoch : 77,  val_loss : 0.0006429732893593609
Epoch : 78,  train loss : 0.0005549193010665476
Epoch : 78,  val_loss : 0.0007945024408400059
Epoch : 79,  train loss : 0.0007028317195363343
Epoch : 79,  val_loss : 0.0005052924971096218
Epoch : 80,  train loss : 0.0007530470029450953
Epoch : 80,  val_loss : 0.0005417302018031478
Epoch : 81,  train loss : 0.0008213587570935488
Epoch : 81,  val_loss : 0.0004592018376570195
Epoch : 82,  train loss : 0.0006208617123775184
Epoch : 82,  val_loss : 0.0006394697120413184
Epoch : 83,  train loss : 0.00046105877845548093
Epoch : 83,  val_loss : 0.0005040430114604533
Epoch : 84,  train loss : 0.0012992353877052665
Epoch : 84,  val_loss : 0.0007674590451642871
Epoch : 85,  train loss : 0.0004630036710295826
Epoch : 85,  val_loss : 0.0006507369107566774
Epoch : 86,  train loss : 0.0007244964363053441
Epoch : 86,  val_loss : 0.0006188398692756891
Epoch : 87,  train loss : 0.0006361436098814011
Epoch : 87,  val_loss : 0.0006549252429977059
Epoch : 88,  train loss : 0.0009385882876813412
Epoch : 88,  val_loss : 0.00028558558551594615
Epoch : 89,  train loss : 0.0005726788658648729
Epoch : 89,  val_loss : 0.0007325569167733192
Epoch : 90,  train loss : 0.0005987148033455014
Epoch : 90,  val_loss : 0.0008759270422160625
Epoch : 91,  train loss : 0.0005115828826092184
Epoch : 91,  val_loss : 0.0008082077838480473
Epoch : 92,  train loss : 0.0006906176568008959
Epoch : 92,  val_loss : 0.000645042397081852
Epoch : 93,  train loss : 0.0004759599396493286
Epoch : 93,  val_loss : 0.0006071369862183928
Epoch : 94,  train loss : 0.0005118490080349147
Epoch : 94,  val_loss : 0.0005505573353730142
Epoch : 95,  train loss : 0.0005994646926410496
Epoch : 95,  val_loss : 0.0005493136122822762
Epoch : 96,  train loss : 0.0006827820907346904
Epoch : 96,  val_loss : 0.0006166108651086688
Epoch : 97,  train loss : 0.0005422237445600331
Epoch : 97,  val_loss : 0.0008642153698019683
Epoch : 98,  train loss : 0.0006725455750711262
Epoch : 98,  val_loss : 0.0005417084903456271
Epoch : 99,  train loss : 0.0006881540757603943
Epoch : 99,  val_loss : 0.0003667235723696649
Epoch : 100,  train loss : 0.00047162937698885 8
Epoch : 100,  val_loss : 0.0006030588410794735
```

Test Normal

In [12]:

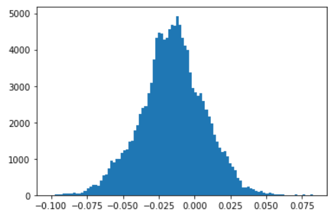
```
test_data=dataset(LPFe_Map,LPK_Map,LPTh_Map,LPTi_Map,albedo,360,720)
test_loader=DataLoader(test_data,batch_size=1)
loss_function=torch.nn.MSELoss()
net_test=model(4,4,1)
net_test=net_test.cuda()
net_test.load_state_dict(torch.load(dir+"/pytorch-models/albedo_blur_best.pth"))
right_predicted=[]
right_truth=[]
total_loss=[]
net_test=net_test.eval()
for i,l in test_loader:
    i=i.cuda()
    l=l.cuda()
    l=torch.reshape(l,(len(l),1))
    output=net_test(i.float())
    loss=loss_function(output,l.float())
    loss=loss.cpu().item()
    total_loss.append(np.sqrt(loss))
    right_predicted.append(output.cpu().item())
    right_truth.append(l.cpu().item())
```

```
print("RMSE Loss on right half :",np.mean(total_loss))
print("R2 score:",r2_score(right_truth,right_predicted))
residual=np.subtract(right_predicted,right_truth)
plt.hist(residual,bins=100)
plt.show()
```

```
RMSE Loss on right half : 0.021777820362415982
R2 score: 0.5114797708710908
```

Out[12]:

```
(array([2.000e+00, 2.000e+00, 1.000e+01, 1.000e+01, 2.900e+01, 4.300e+01,
        3.500e+01, 4.800e+01, 4.600e+01, 5.700e+01, 4.800e+01, 5.400e+01,
        5.700e+01, 1.210e+02, 1.860e+02, 2.440e+02, 2.710e+02, 2.900e+02,
        2.630e+02, 4.110e+02, 5.460e+02, 5.570e+02, 7.260e+02, 9.420e+02,
        8.910e+02, 9.890e+02, 9.980e+02, 1.149e+03, 1.225e+03, 1.250e+03,
        1.473e+03, 1.504e+03, 1.778e+03, 1.920e+03, 2.229e+03, 2.405e+03,
        2.457e+03, 2.796e+03, 3.098e+03, 3.721e+03, 4.325e+03, 4.465e+03,
        4.448e+03, 4.288e+03, 4.335e+03, 4.571e+03, 4.670e+03, 4.658e+03,
        4.926e+03, 4.677e+03, 4.332e+03, 4.120e+03, 4.000e+03, 3.379e+03,
        2.950e+03, 2.824e+03, 2.741e+03, 2.811e+03, 2.588e+03, 2.343e+03,
        2.153e+03, 1.968e+03, 1.658e+03, 1.473e+03, 1.311e+03, 1.185e+03,
        1.220e+03, 1.055e+03, 8.720e+02, 7.700e+02, 6.860e+02, 4.770e+02,
        3.970e+02, 2.180e+02, 2.070e+02, 2.250e+02, 1.820e+02, 1.720e+02,
        1.230e+02, 9.100e+01, 1.050e+02, 7.300e+01, 4.200e+01, 5.800e+01,
        3.900e+01, 2.800e+01, 1.600e+01, 1.300e+01, 1.000e+01, 6.000e+00,
        6.000e+00, 1.000e+00, 2.000e+00, 1.800e+01, 5.000e+00, 7.000e+00,
        9.000e+00, 7.000e+00, 0.000e+00, 2.000e+01]),
 array([-1.01200700e-01, -9.93611804e-02, -9.75216609e-02, -9.56821415e-02,
        -9.38426220e-02, -9.20031026e-02, -9.01635832e-02, -8.83240637e-02,
        -8.64845443e-02, -8.46450248e-02, -8.28055054e-02, -8.09659860e-02,
        -7.91264665e-02, -7.72869471e-02, -7.54474276e-02, -7.36079082e-02,
        -7.17683887e-02, -6.99288693e-02, -6.80893499e-02, -6.62498304e-02,
        -6.44103110e-02, -6.25707915e-02, -6.07312721e-02, -5.88917527e-02,
        -5.70522332e-02, -5.52127138e-02, -5.33731943e-02, -5.15336749e-02,
        -4.96941555e-02, -4.78546360e-02, -4.60151166e-02, -4.41755971e-02,
        -4.23360777e-02, -4.04965582e-02, -3.86570388e-02, -3.68175194e-02,
        -3.49779999e-02, -3.31384805e-02, -3.12989610e-02, -2.94594416e-02,
        -2.76199222e-02, -2.57804027e-02, -2.39408833e-02, -2.21013638e-02,
        -2.02618444e-02, -1.84223250e-02, -1.65828055e-02, -1.47432861e-02,
        -1.29037666e-02, -1.10642472e-02, -9.22472775e-03, -7.38520831e-03,
        -5.54568887e-03, -3.70616943e-03, -1.86664999e-03, -2.71305442e-05,
         1.81238890e-03,  3.65190834e-03,  5.49142778e-03,  7.33094722e-03,
         9.17046666e-03,  1.10099861e-02,  1.28495055e-02,  1.46890250e-02,
         1.65285444e-02,  1.83680639e-02,  2.02075833e-02,  2.20471027e-02,
         2.38866222e-02,  2.57261416e-02,  2.75656611e-02,  2.94051805e-02,
         3.12447000e-02,  3.30842194e-02,  3.49237388e-02,  3.67632583e-02,
         3.86027777e-02,  4.04422972e-02,  4.22818166e-02,  4.41213360e-02,
         4.59608555e-02,  4.78003749e-02,  4.96398944e-02,  5.14794138e-02,
         5.33189332e-02,  5.51584527e-02,  5.69979721e-02,  5.88374916e-02,
         6.06770110e-02,  6.25165305e-02,  6.43560499e-02,  6.61955693e-02,
         6.80350888e-02,  6.98746082e-02,  7.17141277e-02,  7.35536471e-02,
         7.53931665e-02,  7.72326860e-02,  7.90722054e-02,  8.09117249e-02,
         8.27512443e-02]),
 <BarContainer object of 100 artists>)
```



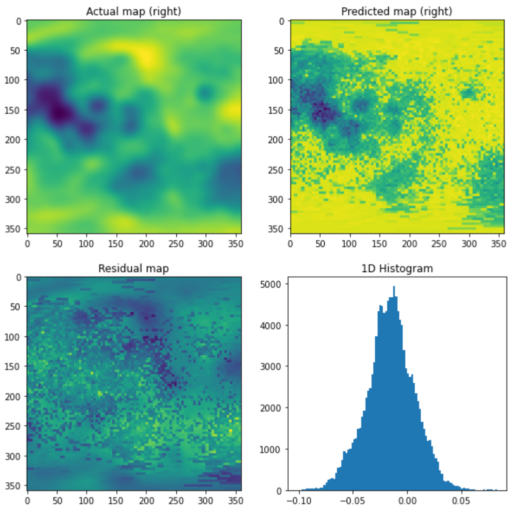
In [17]:

```
ffig, faxes = plt.subplots(2, 2, figsize=(10, 10))
faxes[0, 0].imshow(np.reshape(right_truth,(359,360)))
faxes[0, 0].set_title("Actual map (right)")
faxes[0, 0].grid(False)

faxes[0, 1].imshow(np.reshape(right_predicted,(359,360)))
faxes[0, 1].set_title("Predicted map (right)")
faxes[0, 1].grid(False)

faxes[1, 0].imshow(np.reshape(residual,(359,360)))
faxes[1, 0].set_title("Residual map")
faxes[1, 0].grid(False)

faxes[1, 1].hist(residual, bins=100)
faxes[1, 1].set_title("1D Histogram")
faxes[1, 1].grid(False)
```



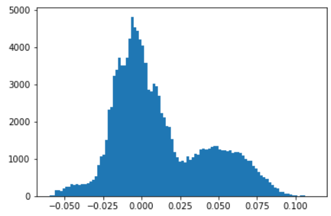
Test Nearby

In [6]:

```
test_data_nearby=dataset_nearby_pixel(LPFe_Map,LPK_Map,LPTh_Map,LPTi_Map,albedo,mode="test")
test_loader_nearby=DataLoader(test_data_nearby,batch_size=1)
loss_function=torch.nn.MSELoss()
net_test=model(36,25,1)
net_test=net_test.cuda()
net_test.load_state_dict(torch.load(dir+"/pytorch-models/albedo_nearby_blur_best.pth"))
right_predicted_near=[]
right_truth_near=[]
total_loss_near=[]
net_test=net_test.eval()
for i,l in test_loader_nearby:
    i=i.cuda()
    l=l.cuda()
    l=torch.reshape(l,(len(l),1))
    output=net_test(i.float())
    loss=loss_function(output,l.float())
    loss=loss.cpu().item()
    total_loss_near.append(np.sqrt(loss))
    right_predicted_near.append(output.cpu().item())
    right_truth_near.append(l.cpu().item())


print("RMSE Loss on right half :",np.mean(total_loss_near))
print("R2 score:",r2_score(right_truth_near,right_predicted_near))
residual_near=np.subtract(right_predicted_near,right_truth_near)
plt.hist(residual_near,bins=100)
plt.show()
```

```
RMSE Loss on right half : 0.02398096107180612
R2 score: 0.6268074896858192
```



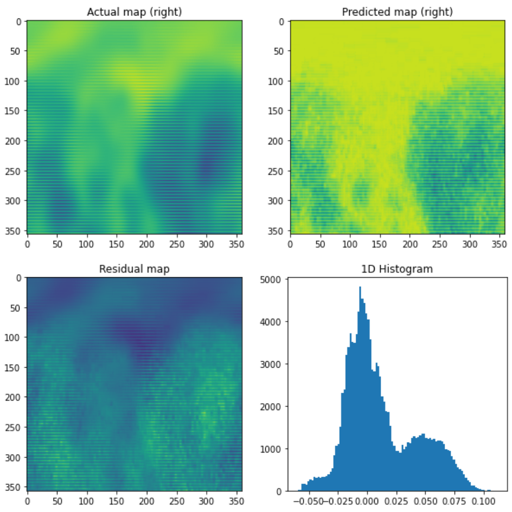
In [7]:

```
ffig, faxes = plt.subplots(2, 2, figsize=(10, 10))
faxes[0, 0].imshow(np.reshape(right_truth_near,(357,359)))
faxes[0, 0].set_title("Actual map (right)")
faxes[0, 0].grid(False)

faxes[0, 1].imshow(np.reshape(right_predicted_near,(357,359)))
faxes[0, 1].set_title("Predicted map (right)")
faxes[0, 1].grid(False)

faxes[1, 0].imshow(np.reshape(residual_near,(357,359)))
faxes[1, 0].set_title("Residual map")
faxes[1, 0].grid(False)

faxes[1, 1].hist(residual_near, bins=100)
faxes[1, 1].set_title("1D Histogram")
faxes[1, 1].grid(False)
```



In [ ]: