Real-time traffic prediction in Chicago using U-Net

CEE 498 Course Project Final Report

Shiyuan Wang, Yudi Wang, Jingzi Chen

Shiyuan8@illinois.edu yudiw3@illinois.edu jingzic2@illinois.edu

University of Urbana-Champaign, Department of Civil and Environmental Engineering

Abstract

Traffic mobility plays a very crucial role in densely-populated cities. It's important to gain the ability to forecast real-time traffic for policymakers in order to relieve the congestion problem. The objective of this project is to predict short-term future traffic volume, directions, and speed on a high-res (750m * 675m) map of Chicago. We developed the traffic prediction model based on the U-Net model, which is a simple structure for pixel-wise outputs. The mean absolute error of predictions of each pixel after training was 6.8681. The spatial distribution of predicted speeds was close to the ground truth; the predicted volumes were close to zero at most pixels, while ground-truth volumes were also close to zero. The report is organized as follows: Section 1 is the introduction and background of the project. Section 2 contains the detailed implementation, including data processing and model development. Section 3 shows the results and discussion, and section 4 concludes the whole project and proposes some further improvements. Training output and codes are attached in the appendix.

1. Introduction

Traffic mobility is crucial in densely-populated cities, and it's influenced by various factors. Thus, traffic forecasting remains challenging. With the increasing traffic volume, it is especially important to predict the traffic mobility (i.e., volume, speed, direction) of the near future. In IARAI (Institute of advanced research in artificial intelligence), traffic data are presented as temporal sequences of cartographic maps, which we are making use of. We are planning to make traffic predictions for the next period from the current period, and hopefully, the neural network will find out the underlying patterns in those data. In all, the task of this project is to predict short-term future traffic volume, directions, and speed on a high-res map of Chicago.

Figure 1 Road map of Chicago

This traffic data has high dimensionality than images: A RGB image has 3 dimensions: the image height, width, and RGB channels; while the data we focus on also have 3 dimensions, but we have 8 channels in the last dimension: traffic volume and speed in NE (directions of 0-90 degrees), SE, SW, and NW respectively. Therefore, the algorithms that are suitable for image prediction can also be deployed in our case. The following graph describes the similarity between the data and an RGB image.

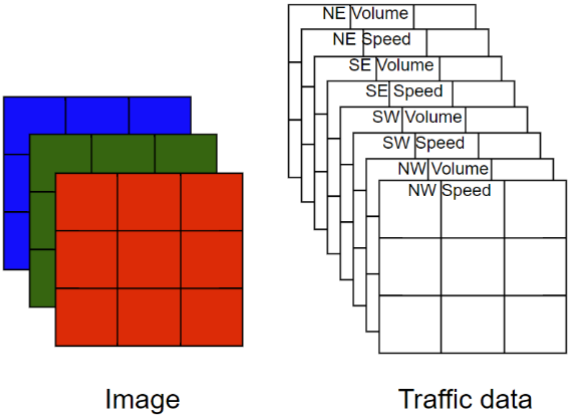


Figure 2 Similarity between the traffic data and an RGB image

As we stated before, we are using data from the previous timestamps to predict data in subsequent time stamps. Therefore, we don't have separate features and labels. Instead, traffic data in each timestamp can be either features for predicting subsequent traffic data, or target labels to supervise learning from previous data. As the graph shows below, traffic data at t_{n-2} is the target for learning from data of t_0 to t_{n-3}, and is also the input for the prediction of traffic data at t_{n-1}.

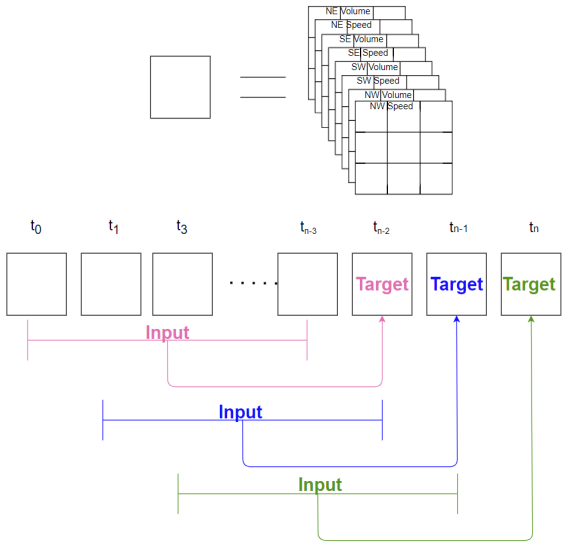


Figure 3 Time frames for training

2. Methodology

2.1 Dataset

The dataset includes the traffic information from January 2019 to June 2019 and from January 2020 to June 2020 in Chicago. The Chicago city road map, which covers 50 km × 50 km of urban area, is divided into cells with an area of about 100m x 100m, and each cell contains traffic data: traffic speed, volume and direction are recorded every five minutes for current road information. The traffic volume and speed in Chicago are mapped in the cells of each road map at each time slot. Each map in the specific time slot contains 8 channels. The first two of the 8 channels record the aggregated volume and average speed whose direction is between 0 and 90 degrees (i.e., NE), the next two channels represent volume and speed for all flows in the direction of SE, the following two for SW and NW, respectively.

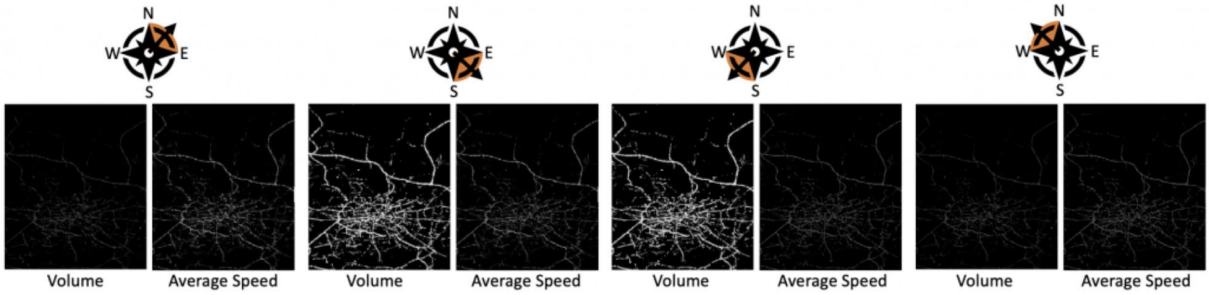


Figure 4 Map of traffic volume and speed in 4 directions

The original dataset is obtained from HERE sources and below shows the description of the variables:

- Volume: The number of cars capped both above and below and normalized and discretized to an integer number between 0 and 255 (uint8).
- Mean speed: The average speed of the collected cars. The values are capped at a maximum level and then discretized to $\{1, 2, \ldots, 255\}$, by linearly scaling the capping speed to 255 and rounding the resulting values to the nearest integer. Value 0 means no cars were collected.

2.2 Data preprocessing

In this project, the initial data we get is a four-dimensional array of 288 * 495 * 436 * 8. This means the size of the researched area is 495 * 436 and the area of each grid is 100m * 100m, and 8 features: speed and volume in four directions are considered, and data is recorded every 5 minutes for a total of one day's worth of data (60/5*24 = 288). Due to the large amount of data, the image resolution should be reduced. The current plan is to reduce the array from 495 * 436 * 8 to 64 * 64 * 8 by using pooling which causes the resolution to change from 100m * 100m to 750m * 675m. Since we are more concerned about the congested traffic condition, we would like to predict the highest volume and the lowest speed in each 750m * 675m area, the channel order should be changed as NEvolume, SEvolume, SWvolume, NWvolume, NEspeed, SEspeed, SWspeed, NWspeed which would benefit for subsequent pooling. For volume prediction, we select max pooling to preprocess data while for speed prediction, we select the non-zero minimum pooling. The schematic diagram is as follows,

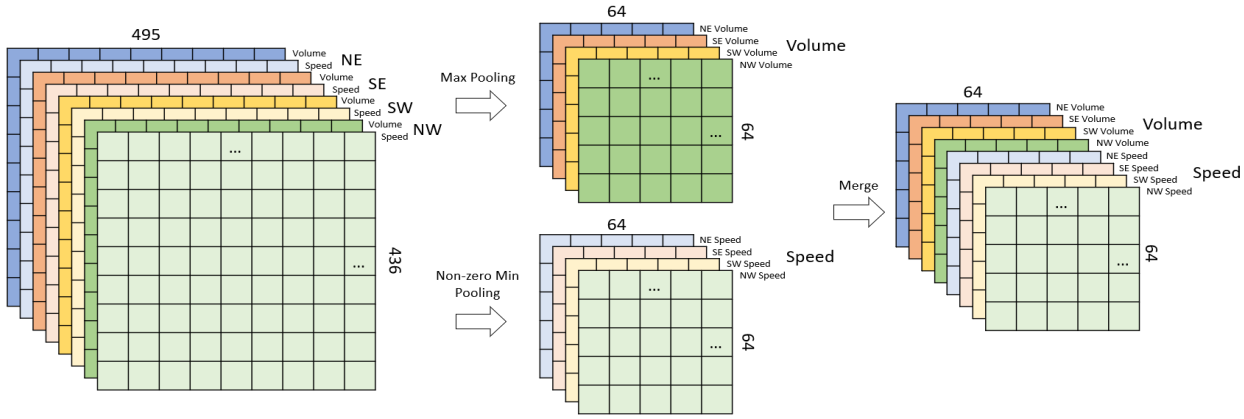


Figure 5 Preprocessing scheme

After the preprocessing, the physical meaning of our dataset would be:

- Volume: The max number of cars per 100m* 100m area within a 750m * 675m area.
- Speed: The lowest average speed within a 750m * 675m area.

Those results would help target the congestion degree, that is, whether the volume exceeds the normal volume and whether the speed is slower than the speed limit in each division area.

Below shows two simple figures describing the volume and speed at 9 am on 06/01/19:

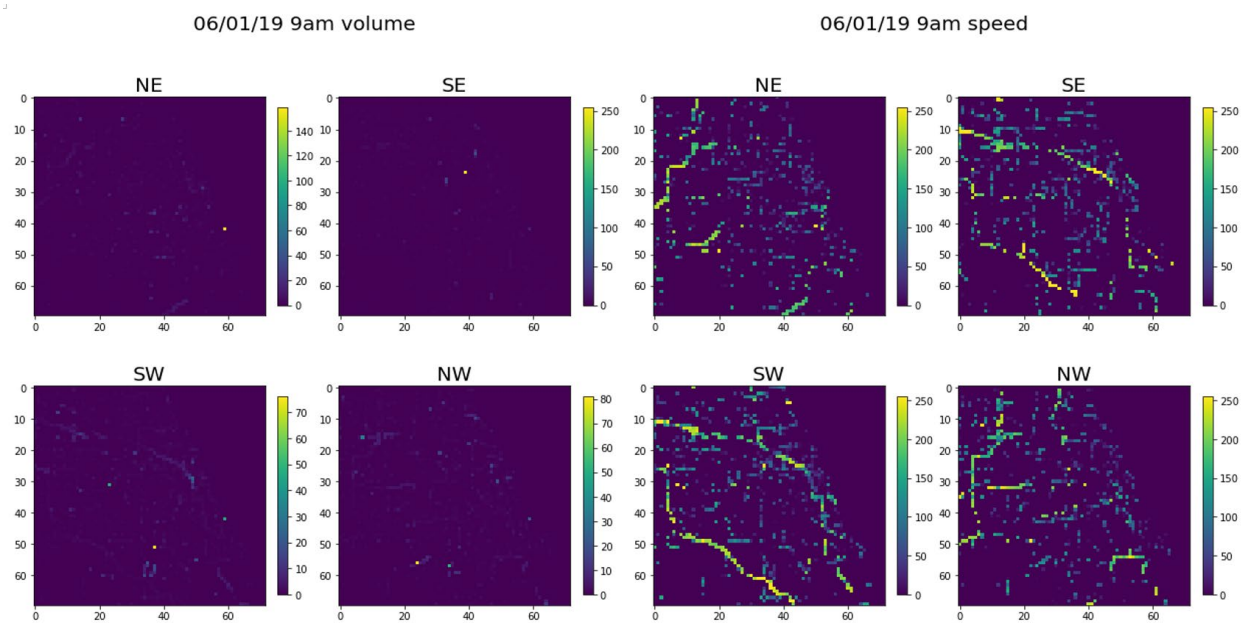


*Figure 6 Volume and speed with 64*64 resolutions at June 1st 2019*

2.3 U-Net based model

The traffic prediction model we developed is based on the U-Net model, which is a simple structure for pixel-wise outputs. The U-Net is a type of convolutional neural network which was developed for image segmentation initially. U-Net architecture consists of an encoder part and a decoder part. The encoder compressed the initial input array to an encoded array with fewer pixels but more channels by repeated convolutions and pooling. Then, the decoder expands the encoded array to output with more pixels and fewer channels by up-convolutions (transposed convolution) and concatenations with features from the previous encoder. The reason for choosing U-Net is that training a U-Net model commonly needs less time compared to recurrent neural networks which may perform with similar accuracy.

As we stated before, U-Net is composed of an encoder part and a decoder part. The encoder part is simply composed of convolution and max pooling, which we've learned in class; while the decoder part contains transposed convolution which we haven't learned before. Generally, we can treat transposed convolution as the reverse of a standard convolutional layer. In other words, transposed convolution converts from a lower resolution image to a higher one. The following graph shows how a simple transposed convolution works.

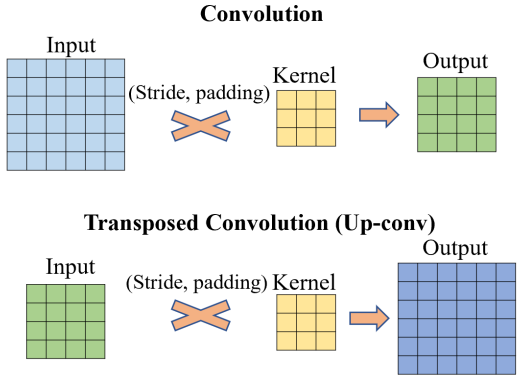


Figure 7 Comparison between convolution and transposed convolution

The input of the model is a 64*64*96 array which is concatenated by twelve 64*64*8 arrays from t_{n-14} to t_{n-3}. The output is a 64*64*8 array corresponding to t_n. Note that each 64*64*8 array represents the 5-min interval of traffic data. That is, suppose we are at 8:00 am, we want to predict traffic data at 8: 15 am, using previous traffic data from 7:00 am to 8:00 am.

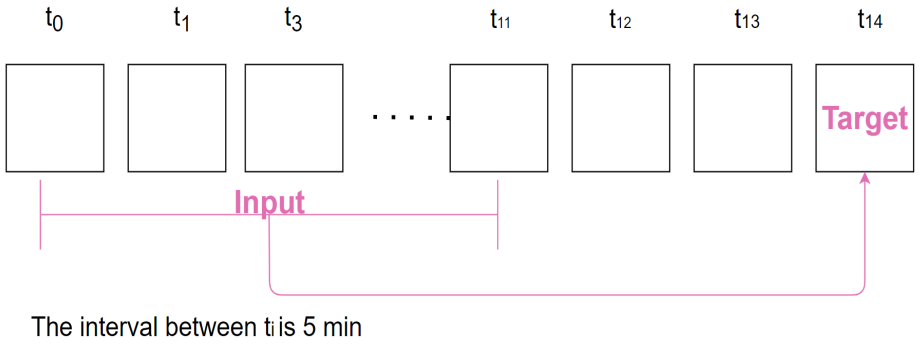


Figure 8 Sliding window on timestamps for training

We assume that the traffic data at a specific timestamp are only influenced by previous short-term traffic data (1 hour in our case). Long-term factors such as seasonal effects are ignored in our setting. The following table shows how we split the dataset into train data and test data.

Table 1 Traning and testing data split

Train-test split	Time	Percentage
Train data	2019/01/2 -- 2019/05/31	75% (150 days)
Test data	2019/06/01 -- 2019/06/30	25% (30 days)

The traffic prediction model's architecture is shown in the following figure. In this model, the 64*64*96 input array is transformed into a 64*64*128 array by 2 convolutions with 3x3 kernels and 1 padding to keep the array's size in the first 2 dimensions. Then, the 64*64*128 array is compressed into a 32*32*128 array by pooling. In the pooling process, the maximum values in the pooling kernel are retained in the post-processed array. Furthermore, the 32*32*128 array is transformed into a 32*32*256 array by convolutions similar to the previous transformation. With the processed 32*32*256 array, a 64*64*128 array is obtained by up-convolution and the 64*64*128 array is further concatenated with the same-size array from the previous convoluting process. With 4 times convolutions, leaky ReLU, and hyperbolic tangent function, the concatenated 64*64*256 array is transformed into a 64*64*8 array. After changing negative elements to zero, the array's elements are between 0 and 255, which is compatible with the ground truth's range.

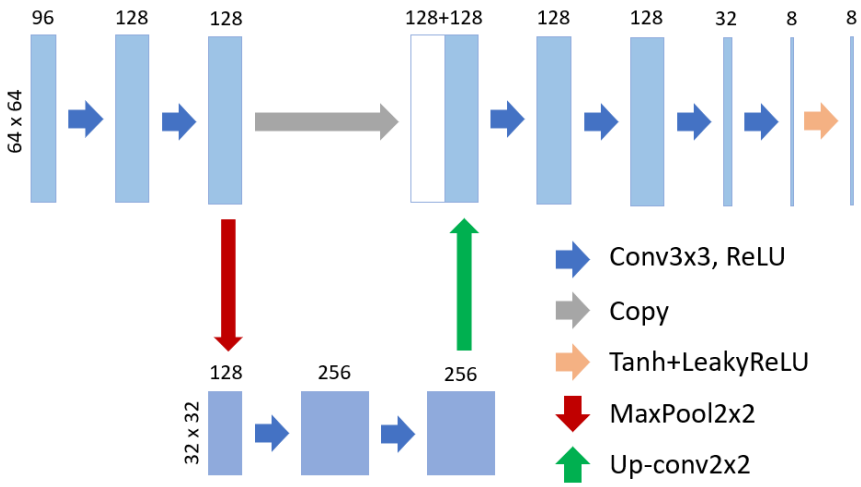


Figure 9 Sliding window on timestamps for training

The model output shape per operation is shown as follows.

Table 2 Model output shape per operation

Block		Output shape (img h, img w, channels)
Input		(64, 64, 96)
Block 1	Conv+ReLu	(64, 64, 128)
	Conv+ReLu	(64, 64, 128)
Max Pooling		(32, 32, 128)
Block 2	Conv+ReLu	(32, 32, 256)
	Conv+ReLu	(32, 32, 256)
Up-Conv		(64, 64, 128)
Concatenate		(64, 64, 256)
Block 3	Conv+ReLu	(64, 64, 128)
	Conv+ReLu	(64, 64, 128)
Conv		(64, 64, 32)
Conv+Tanh+LeakyReLu		(64, 64, 8)

The training parameters are shown in the following table.

Table 3 Training parameters

Batch size	100
Iterations in each epoch	432
Epoch number	10
Loss function	Mean square error
Optimizer	Adaptive Moment Estimation (Adam)
Learning rate	0.001

3. Results and discussion

By implementing previously mentioned data processing and model selecting, the final model building was completed on Jupyter notebook. The complete code is attached at the end. The model is trained on a remote server.

The loss estimated by the training dataset and testing dataset for the models after each training epoch is shown in the following figure. The untrained MSE loss on the test dataset was 2123.12. The MSE loss after 10-epoch training was 470.56. 78 percent of the mean square error was eliminated. The results showed the efficiency of training.

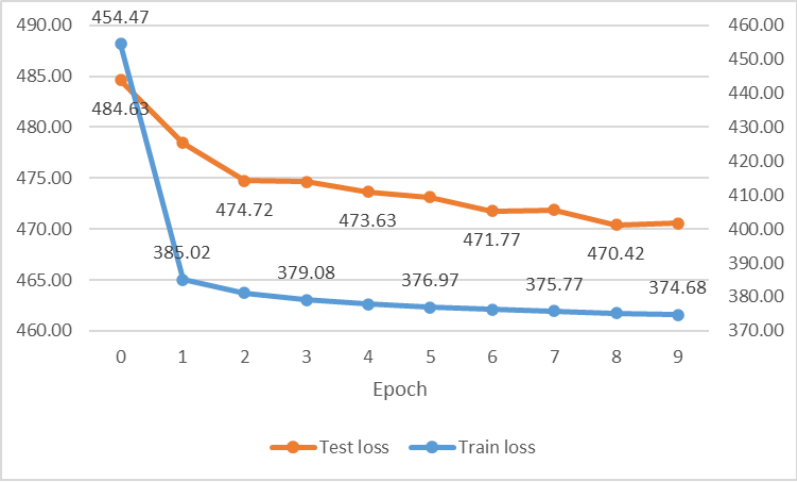


Figure 10 Loss on training and testing dataset of each epoch

After 10 epochs, the absolute errors at each pixel between predictions and ground truths in test data were also estimated. The histogram of pixel-wise absolute errors is shown in the following figure. The red line represents the mean absolute error, which was 6.8681. The distribution of absolute errors was highly right-skewed, which indicated that most of the errors were less than the mean absolute error.

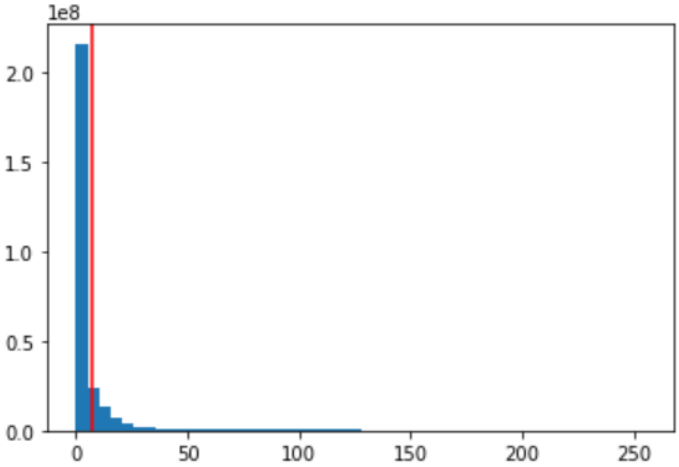


Figure 11 Histogram of pixel absolute errors

The following figure zooms to the histogram of pixel-wise absolute errors less than 50. The red line still represents the mean absolute error of all pixels. The results were also right-skewed. The histogram indicated that most of the pixel-wise predictions were highly accurate.

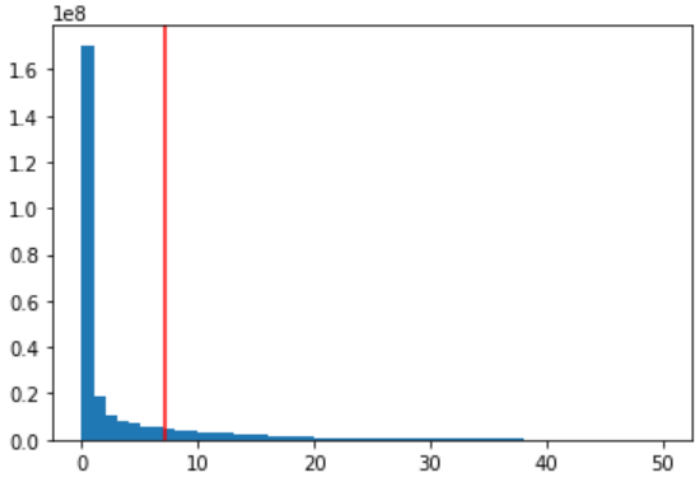


Figure 12 Histogram of absolute errors less than 50

The predictions and ground truths of the speed map are shown in the following figure. The spatial distribution of speed was close to the ground truth. For example, the left-bottom corners of SW and SE, and the right-bottom corners of NE and NW had higher speeds. However, the predicted values were not accurate at this specific timestamp. The possible reason is insufficient training epochs or limited layers in the prediction model due to limited computational resources.

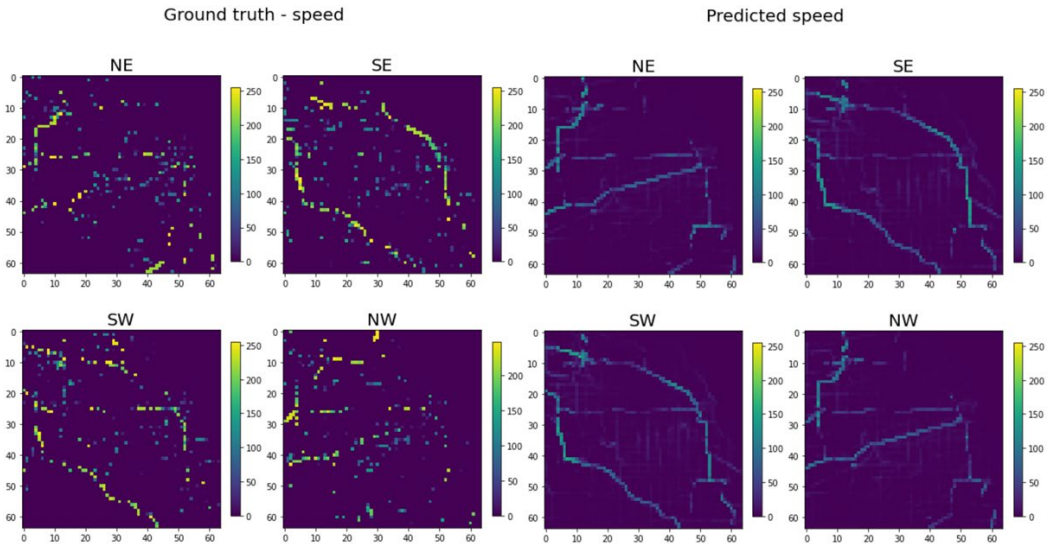


Figure 13 Speed maps of predictions and ground truth

The predictions and ground truths of the volume map are shown in the following figure. The predicted and ground-truth volumes were close to zero at most pixels. The predicted values were inaccurate at this specific timestamp. The possible reason is insufficient training epochs or limited layers in the prediction model due to limited computational resources. The maximum pooling kernels in the model's architecture may also lead to excessive loss of original information.

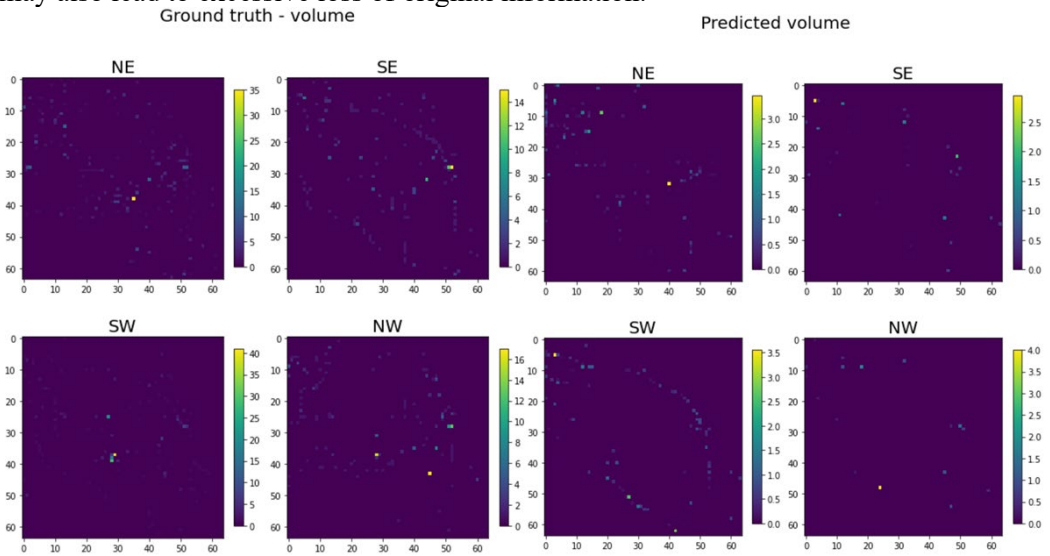


Figure 14 Volume maps of predictions and ground truth

4. Conclusion and further developments

The project proposed a traffic prediction model based on U-Net architecture. After running limited epochs, the mean squared error compared with the test data was 470.56, which decreased by 78%. The mean absolute error of predictions after training was 6.8681. The spatial distribution of speed was close to the ground truth. However, the predicted values of speed were inaccurate at some specific timestamps. The predicted volumes were close to zero at most pixels, while ground-truth volumes were also closed to zero.

For further development, increasing the resolution of data, increasing the depth of U-Net architecture, and more epochs in training may increase the accuracy. Changing maximum pooling to average pooling in the connection of down blocks also decreases the loss of original information and further

improves accuracy. Studies show that transposed convolution often causes a checkerboard pattern of artifacts in the results, and thus trying out a new decoder/upsampling method other than transposed convolution may also improve the performance of the current model.

5. Individual responsibilities contributions

Every group member will contribute to each task equally. We list the tasks as follows:

1. Proposing idea. (√)
2. Group meetings. (5 times)
3. Paper and slides writing. (√)
4. Model development (√)

References

https://www.here.com/
https://amaarora.github.io/2020/09/13/unet.html
https://distill.pub/2016/deconv-checkerboard/
https://machinelearningmastery.com/cnn-long-short-term-memory-networks/
https://www.iarai.ac.at/traffic4cast/2021-competition/challenge/#analyticschallenge

Appendix

Output of training

Untrained test

Test loss: 2123.1247404251976

Epoch 0

Train loss: 454.46982470023386
Test loss: 484.6342233241289

Epoch 1

Train loss: 385.0198075367514
Test loss: 478.4517373271372

Epoch 2

Train loss: 381.0989158966702
Test loss: 474.719195837262

Epoch 3

Train loss: 379.08081337913285
Test loss: 474.64014513739227

Epoch 4

Train loss: 377.8818218470173
Test loss: 473.63264763492276

Epoch 5

Train loss: 376.9697783928183
Test loss: 473.12029432976385

Epoch 6

Train loss: 376.2976159186485
Test loss: 471.7649981619298

Epoch 7

Train loss: 375.7709767635903
Test loss: 471.86696054743624

Epoch 8

Train loss: 375.22483347740086
Test loss: 470.42053503277657

Epoch 9

Train loss: 374.6795085013051
Test loss: 470.55507405598956

Data Preprocessing

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive/')
```

```
Mounted at /content/gdrive/
```

In [ ]:
```
import os
path_of_this_jupyer_notebook_without_filename=r'/content/gdrive/MyDrive/CEE498_Project'
os.chdir(path_of_this_jupyer_notebook_without_filename)
```

In [1]:
```
!unzip "/content/gdrive/MyDrive/CEE498_Project/training.zip" -d "/content";
```

In [ ]:
```
import numpy as np
import h5py
import os
import torch
import torch.nn as nn
from tqdm import tqdm
```

Example of lowering resolution of inputs

In [ ]:
```
# Reduce the number of pixels from 495*436 to 70*72
# resolution from 100m*100m to around 750m * 675m
f = np.array(h5py.File('/content/training/2019-06-06_CHICAGO_8ch.h5','r')['array']) # original shape (288, 495, 436, 8)
f_mov_array = np.moveaxis((f),-1,1) # to shape(288, 8, 495, 436)
print('first move axis',f_mov_array.shape)

f_v = f_mov_array[:,[0,2,4,6],:,:] # index by volume
f_s = f_mov_array[:,[1,3,5,7],:,:] # index by speed

m = nn.MaxPool2d((7,6))

# Max volume
f_v = torch.from_numpy(f_v).float() # numpy to torch
output_v = m(f_v) # to shape(288, 4, 70, 72)
print('after max pooling of volume',output_v.shape)

# Max (0,Min(speed))
f_s = torch.from_numpy(f_s).float() # numpy to torch
f_s = f_s.where(f_s!=0, torch.tensor(256.0)) * (-1)
output_s = m(f_s) * (-1) # to shape(288, 4, 70, 72)
output_s = output_s.where(output_s!=256, torch.tensor(0.0))
print('after min pooling of speed',output_s.shape)

output_v = np.moveaxis(output_v.numpy(),1,-1)
output_s = np.moveaxis(output_s.numpy(),1,-1)
output_final = np.concatenate((output_v,output_s),axis=-1) # to shape(288, 70, 72, 8)
print('final output shape',output_final.shape)
# # output_final = torch.from_numpy(output_final).float() # numpy to torch
```

```
first move axis (288, 8, 495, 436)
after max pooling of volume torch.Size([288, 4, 70, 72])
after min pooling of speed torch.Size([288, 4, 70, 72])
final output shape (288, 70, 72, 8)
```

In [ ]:
```
# write into h5 file
file = h5py.File('test.h5','w')
file.create_dataset('data', data = output_final)
file.close()
```

In [ ]:
```
aa = np.array(h5py.File('/content/gdrive/MyDrive/CEE498_Project/test.h5','r')['data'])
```

Preprocess the data

Train data *2019-01-02 -- 2019-05-31*

In [ ]:
```
# extract file name from 2019-01-02
train_file= [sorted(os.listdir(r'/content/training'))[0][:10]]

# extract matrices from 2019-01-02
f = np.array(h5py.File('/content/training/'+sorted(os.listdir(r'/content/training'))[100],'r')['array']) # 0, 50, 100

f_mov_array = np.moveaxis((f),-1,1) # to shape(288, 8, 495, 436)

f_v = f_mov_array[:,[0,2,4,6],:,:] # index by volume
f_s = f_mov_array[:,[1,3,5,7],:,:] # index by speed

m = nn.MaxPool2d((7, 6))

# Max volume
f_v = torch.from_numpy(f_v).float() # numpy to torch
output_v = m(f_v) # to shape(288, 4, 70, 72)

# Max (0,Min(speed))
f_s = torch.from_numpy(f_s).float() # numpy to torch
f_s = f_s.where(f_s!=0, torch.tensor(256.0)) * (-1)
output_s = m(f_s) * (-1) # to shape(288, 4, 70, 72)
output_s = output_s.where(output_s!=256, torch.tensor(0.0))

output_v = np.moveaxis(output_v.numpy(),1,-1)
output_s = np.moveaxis(output_s.numpy(),1,-1)
train_data = np.concatenate((output_v,output_s),axis=-1) # to shape(288, 70, 72, 8)

type(train_data),train_data.shape
```

Out[ ]:
```
(numpy.ndarray, (288, 70, 72, 8))
```

In [ ]:
```
for filename in tqdm(sorted(os.listdir(r'/content/training'))[1:150]): # extract files from 2019-01-03 -- 2019-05-31 [1:150] #1:50, 51:100 101:150
  # extract the date
  train_file.append(filename[:10])
  # extract matrice and change the resolution
  f = np.array(h5py.File('/content/training/'+filename,'r')['array'])

  f_mov_array = np.moveaxis((f),-1,1) # to shape(288, 8, 495, 436)

  f_v = f_mov_array[:,[0,2,4,6],:,:] # index by volume
  f_s = f_mov_array[:,[1,3,5,7],:,:] # index by speed

  m = nn.MaxPool2d((7, 6))

  # Max volume
  f_v = torch.from_numpy(f_v).float() # numpy to torch
  output_v = m(f_v) # to shape(288, 4, 70, 72)

  # Max (0,Min(speed))
  f_s = torch.from_numpy(f_s).float() # numpy to torch
  f_s = f_s.where(f_s!=0, torch.tensor(256.0)) * (-1) # convert 0 to the smallest number -256
  output_s = m(f_s) * (-1) # to shape(288, 4, 70, 72)
  output_s = output_s.where(output_s!=256, torch.tensor(0.0)) # convert 256 to 0

  output_v = np.moveaxis(output_v.numpy(),1,-1)
  output_s = np.moveaxis(output_s.numpy(),1,-1)
  output_final = np.concatenate((output_v,output_s),axis=-1) # to shape(288, 70, 72, 8)

  train_data = np.concatenate((train_data,output_final))
  # train_data = torch.from_numpy(train_data).float() # we want to numpy to torch when using, so no need to convert it to torch right now.
train_data.shape
```

```
100%|██████████| 49/49 [04:34<00:00,  5.59s/it]
```

Out[ ]:
```
(14400, 70, 72, 8)
```

In [ ]:
```
# write into h5 file
file = h5py.File('Train_7072_3.h5','w')
file.create_dataset('data', data = train_data)
file.close()
```

In [ ]:
```
!mv Train_7072_3.h5 gdrive/MyDrive/CEE498_Project/
```

Test data *2019-06-01 -- 2019-06-30*

In [ ]:
```
# extract file name from 2019-06-01
test_file= [sorted(os.listdir(r'/content/training'))[150][:10]]

# extract matrices from 2019-06-01
f = np.array(h5py.File('/content/training/'+sorted(os.listdir(r'/content/training'))[150],'r')['array'])

f_mov_array = np.moveaxis((f),-1,1) # to shape(288, 8, 495, 436)

f_v = f_mov_array[:,[0,2,4,6],:,:] # index by volume
f_s = f_mov_array[:,[1,3,5,7],:,:] # index by speed

m = nn.MaxPool2d((7,6))

# Max volume
f_v = torch.from_numpy(f_v).float() # numpy to torch
output_v = m(f_v) # to shape(288, 4, 70, 72)

# Max (0,Min(speed))
f_s = torch.from_numpy(f_s).float() # numpy to torch
f_s = f_s.where(f_s!=0, torch.tensor(256.0)) * (-1)
output_s = m(f_s) * (-1) # to shape(288, 4, 70, 72)
output_s = output_s.where(output_s!=256, torch.tensor(0.0))

output_v = np.moveaxis(output_v.numpy(),1,-1)
output_s = np.moveaxis(output_s.numpy(),1,-1)
test_data = np.concatenate((output_v,output_s),axis=-1) # to shape(288, 70, 72, 8)

type(test_data),test_data.shape
```

Out[ ]:
```
(numpy.ndarray, (288, 70, 72, 8))
```

In [ ]:
```
for filename in tqdm(sorted(os.listdir(r'/content/training'))[151:180]): # extract files from 2019-06-02 -- 2019-06-30
  # extract the date
  test_file.append(filename[:10])
  # extract matrice and change the resolution
  f = np.array(h5py.File('/content/training/'+filename,'r')['array'])

  f_mov_array = np.moveaxis((f),-1,1) # to shape(288, 8, 495, 436)

  f_v = f_mov_array[:,[0,2,4,6],:,:] # index by volume
  f_s = f_mov_array[:,[1,3,5,7],:,:] # index by speed

  m = nn.MaxPool2d((7, 6))

  # Max volume
  f_v = torch.from_numpy(f_v).float() # numpy to torch
  output_v = m(f_v) # to shape(288, 4, 70, 72)

  # Max (0,Min(speed))
  f_s = torch.from_numpy(f_s).float() # numpy to torch
  f_s = f_s.where(f_s!=0, torch.tensor(256.0)) * (-1) # convert 0 to the smallest number -256
  output_s = m(f_s) * (-1) # to shape(288, 4, 70, 72)
  output_s = output_s.where(output_s!=256, torch.tensor(0.0)) # convert 256 to 0

  output_v = np.moveaxis(output_v.numpy(),1,-1)
  output_s = np.moveaxis(output_s.numpy(),1,-1)
  output_final = np.concatenate((output_v,output_s),axis=-1) # to shape(288, 70, 72, 8)

  test_data = np.concatenate((test_data,output_final))
  # train_data = torch.from_numpy(test_data).float() # we want to numpy to torch when using, so no need to convert it to torch right now.
test_data.shape
```

```
100%|██████████| 29/29 [02:23<00:00,  4.97s/it]
```

Out[ ]:
```
(8640, 70, 72, 8)
```

In [ ]:
```
# write into h5 file
file = h5py.File('Test_7072.h5','w')
file.create_dataset('data', data = test_data)
file.close()
```

In [ ]:
```
!mv Train2.h5 gdrive/MyDrive/CEE498_Project/
```

Feature Engineering

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

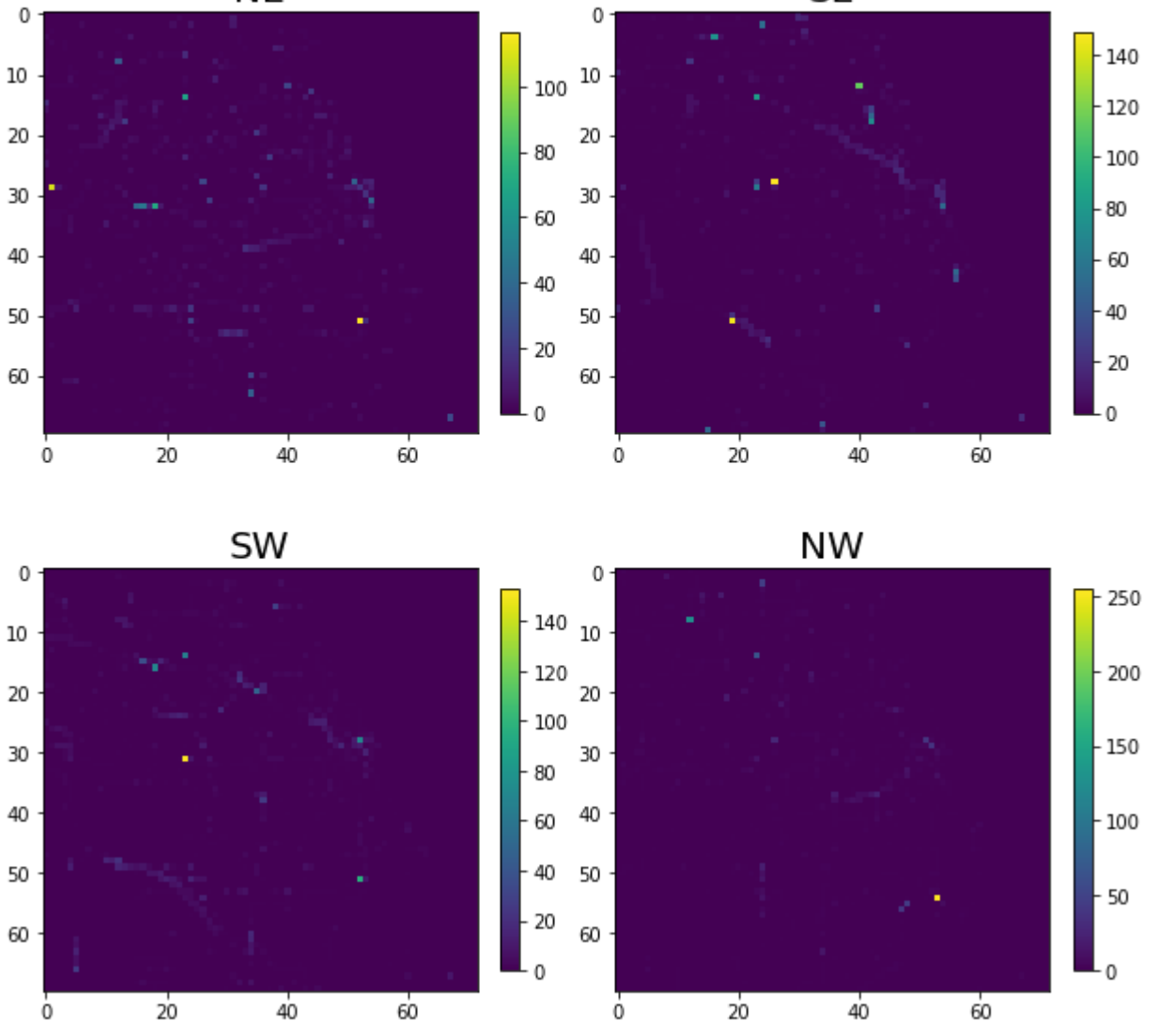
```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/
gdrive", force_remount=True).
```

In [3]:
```
import numpy as np
import h5py
import matplotlib.pyplot as plt
```

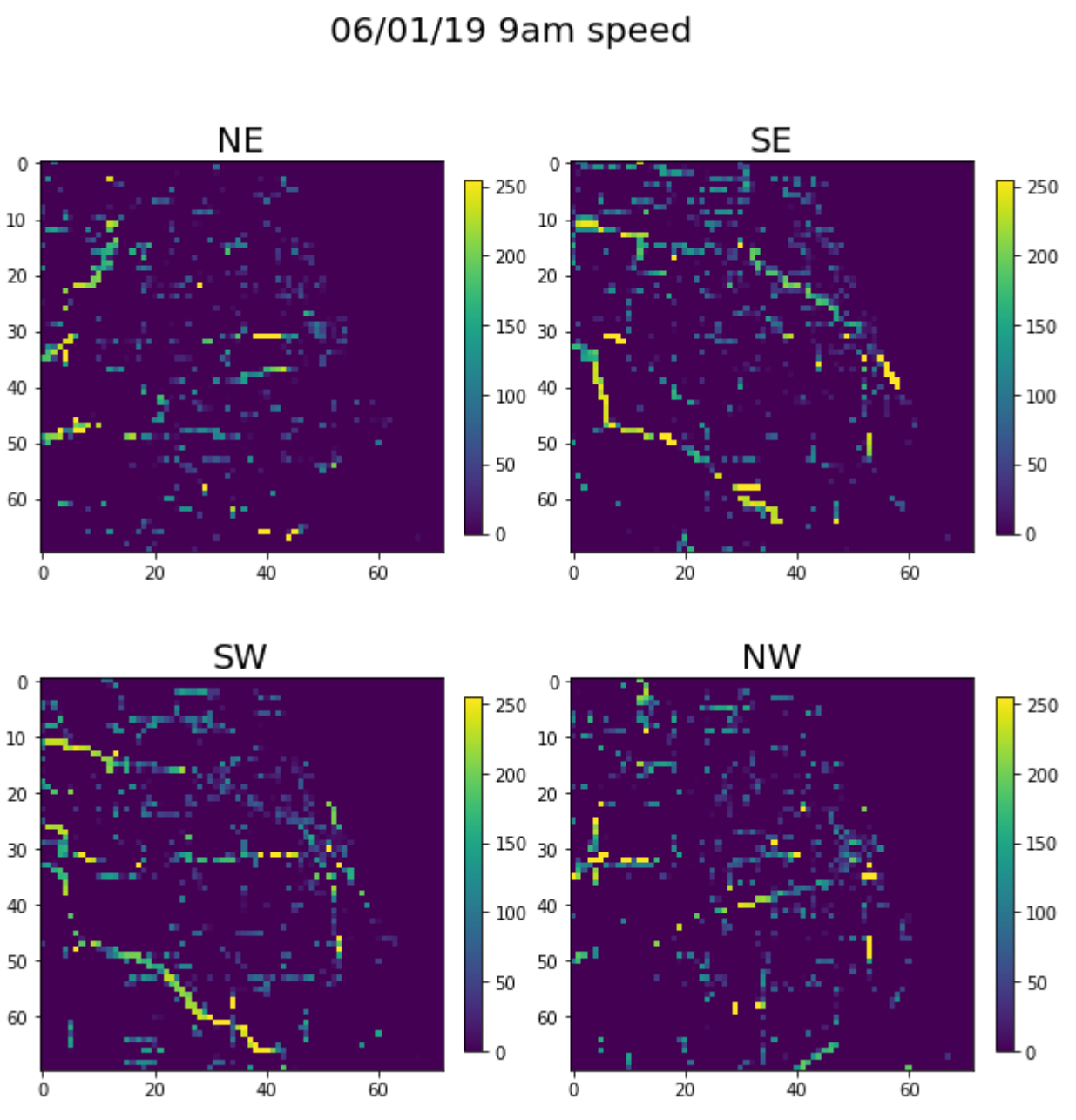
In [4]:
```
Test_data = np.array(h5py.File('/content/gdrive/MyDrive/CEE498_Project/Test_7072.h5','r')['data'])
```

In [6]:
```
DIR = ['NE','SE','SW','NW']
```

In [10]:
```
#01/02/19 9 am
plt.figure(figsize=(10,10))
for i in range(4):
  X = Test_data[12*18,:,:,i]
  ax = plt.subplot(2,2,i+1)
  ax = plt.imshow(X)
  plt.colorbar(ax.colorbar, fraction=0.04)
  plt.title(str(DIR[i]),fontsize=20);
plt.suptitle('06/01/19 9am volume',fontsize=20)
plt.show()
```



In [11]:
```
#01/02/19 9 am
plt.figure(figsize=(10,10))
for i in range(4):
  X = Test_data[12*18,:,:,i+4]
  ax = plt.subplot(2,2,i+1)
  ax = plt.imshow(X)
  plt.colorbar(ax.colorbar, fraction=0.04)
  plt.title(str(DIR[i]),fontsize=20);
plt.suptitle('06/01/19 9am speed',fontsize=20)
plt.show()
```



U-Net Model

In [ ]:
```
import numpy as np
import h5py
Train = h5py.File('Train_7072.h5', 'r')
Test = h5py.File('Test_7072.h5', 'r')
Train_data = Train['data']
Test_data = Test['data']
Train_data = Train['data']
Test_data = Test['data']
Train_data = Train_data[:,6:,:64,:]
Test_data = Test_data[:,6:,:64,:]
Train_data.shape,Test_data.shape
Test_data_mod = np.moveaxis(Test_data,-1,1)
Test_data_mod = Test_data_mod.reshape(-1,64,64)
Test_data_mod.shape
Train_data_mod = np.moveaxis(Train_data,-1,1)
Train_data_mod = Train_data_mod.reshape(-1,64,64)
Train_data_mod.shape
import torch
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self, data, sequence_length=12, pred_length=3):
        self.sequence_length = sequence_length
        self.y = data
        self.X = data
        self.pred_length = pred_length

    def __len__(self):
        length = int(len(self.y)/8-self.sequence_length-self.pred_length+1)
        return length

    def __getitem__(self, i):
        x = self.X[i*8:(i+self.sequence_length)*8,:,:]
        y = self.y[(i+self.sequence_length+self.pred_length-1)*8:(i+self.sequence_length+self.pred_leng
th)*8,:,:]
        return torch.from_numpy(x).float(), torch.from_numpy(y).float()
```

In [ ]:
```
sequence_length = 12
pred_length = 3
TrainDataset = MyDataset(Train_data_mod,sequence_length=sequence_length, pred_length=pred_length)
X, y = TrainDataset[0]
print(X.shape)
print('Dataset length:',len(TrainDataset))
print(len(Train_data))

#Training set
train_dataset=MyDataset(Train_data_mod,sequence_length=sequence_length, pred_length=pred_length)
train_dataloader=torch.utils.data.DataLoader(train_dataset, batch_size=100, shuffle=True)

#Test set
test_dataset=MyDataset(Test_data_mod,sequence_length=sequence_length, pred_length=pred_length)
test_dataloader=torch.utils.data.DataLoader(test_dataset, batch_size=100, shuffle=True)
```

In [ ]:
```
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.modules.activation import Softmax

class Block(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_ch, out_ch, kernel_size = 3, padding=1),
            # nn.BatchNorm2d(out_ch), # added
            nn.ReLU(inplace=True), # added inplace=True
            nn.Conv2d(out_ch, out_ch, kernel_size = 3, padding=1),
            # nn.BatchNorm2d(out_ch), # added
            nn.ReLU(inplace=True) # added inplace=True
        )
    def forward(self, x):
        return self.block(x)
```

```python
class Encoder(nn.Module):
    def __init__(self, chs=(96,128,128,128)):
        super().__init__()
        self.enc_blocks = nn.ModuleList([Block(chs[i], chs[i+1]) for i in range(len(chs)-1)])
        self.pool       = nn.MaxPool2d(2,2)

    def forward(self, x):
        ftrs = []
        for block in self.enc_blocks:
            x = block(x)
            ftrs.append(x)
            x = self.pool(x)
        return ftrs


class Decoder(nn.Module):
    def __init__(self, chs=(128, 128, 128)):
        super().__init__()
        self.chs         = chs
        self.upconvs    = nn.ModuleList([nn.ConvTranspose2d(chs[i], chs[i+1], 2, 2) for i in range(len(
chs)-1)])
        self.dec_blocks = nn.ModuleList([Block(chs[i], chs[i+1]) for i in range(len(chs)-1)])

    def forward(self, x, encoder_features):
        for i in range(len(self.chs)-1):
            x        = self.upconvs[i](x)
            enc_ftrs = self.crop(encoder_features[i], x).to(device)
            x        = torch.cat([x, torch.Tensor(enc_ftrs)], dim=1)
            x        = self.dec_blocks[i](x)
        return x

    def crop(self, enc_ftrs, x):
        _, _, H, W = x.shape
        enc_ftrs   = torchvision.transforms.CenterCrop([H, W])(enc_ftrs)
        return enc_ftrs


class UNet(nn.Module):
    def __init__(self, enc_chs=(96, 128, 256), dec_chs=(256, 128), num_class=8, out_sz=(572,572)): # de
leted argument: retain_dim=False
        super().__init__()
        self.encoder     = Encoder(enc_chs)
        self.decoder     = Decoder(dec_chs)
        self.head        = nn.Sequential(
            nn.Conv2d(dec_chs[-1], 32, 1),
            # nn.BatchNorm2d(32), # added
            nn.ReLU(inplace=True), # added inplace=True
            nn.Conv2d(32, num_class, 1),   # added
            nn.LeakyReLU(negative_slope=1e-4,inplace=True),
            nn.Tanh()
        )
        # self.retain_dim  = retain_dim

    def forward(self, x):
        enc_ftrs = self.encoder(x)
        out      = self.decoder(enc_ftrs[::-1][0], enc_ftrs[::-1][1:])
        out      = 255*self.head(out)
        # if self.retain_dim:
        #     out = F.interpolate(out, out_sz)
        return out
```

In [ ]:

```python
#If there is a GPU avaiable for computations, we want to use it (else we will use a CPU)  "cuda:0" if t
orch.cuda.is_available() else
device = torch.device("cpu")
```

In [ ]:
```python
from tqdm import tqdm
def train_model(train_data_loader, model, loss_function, optimizer, device):

    model = model.to(device)
    model.train()
    loss_individual_epoch = 0
    for i, (images, labels) in tqdm(enumerate(train_data_loader),'Training progress within an epoch'):
        images=images.to(device)
#         labels=labels.float()
        labels=labels.to(device)
        # forward
        outputs=model(images).to(device)
        loss=loss_function(outputs,labels)
        # backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_individual_epoch=loss_individual_epoch+loss.item()
#         if i<1:
#             print("Train individual loss: "+ str(loss_individual_epoch))
#         else:
#             print("Train individual loss: "+ str(loss_individual_epoch/i+1))
    loss_individual_epoch=loss_individual_epoch/i
    print("Train loss: "+ str(loss_individual_epoch))
    return loss_individual_epoch,model
```

In [ ]:
```python
def test_model(data_loader, model, loss_function, device):

    num_batches = len(data_loader)
    # your code here #
    model = model.to(device)
    model.eval()
    total_loss = 0
    all_pred   = torch.tensor([]);
    all_labels = torch.tensor([]);
    with torch.no_grad():
        for i, data in enumerate(data_loader, 0):
            """
            Fill your code after this
            """
            images, labels  = data
            images          = images.to(device)
            labels          = labels.to(device)

            # Perform forward pass
            outputs = model(images)

            # Compute loss
            loss = loss_function(outputs,labels)

            total_loss = total_loss + loss.item()

            all_labels   = torch.cat((all_labels,labels))
            all_pred     = torch.cat((all_pred,outputs))

#             if i<1:
#                 print("Test individual loss: "+ str(total_loss))
#             else:
#                 print("Test individual loss: "+ str(total_loss/(i+1)))

    avg_loss = total_loss / num_batches
    print(f"Test loss: {avg_loss}")
    return all_pred,all_labels
```

In [ ]:
```python
model = UNet()          # Define your model
loss_function = nn.MSELoss(reduction='mean')
print("Untrained test\n--------")
test_pred, test_labels = test_model(test_dataloader, model, loss_function, device)
print()
```

In [ ]:
```
learning_rate = 1e-3    # you can modify this parameter
epoch = 20               # you can modify this parameter
# model = UNet()          # Define your model
loss_function = nn.MSELoss(reduction='mean')
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for ix_epoch in range(epoch):
    print(f"Epoch {ix_epoch}\n---------")
    train_loss,model = train_model(train_dataloader,model,loss_function,optimizer,device)
    test_pred, test_labels = test_model(test_dataloader, model, loss_function,device)
    print()
```

In [ ]:
```
torch.save(model.state_dict(), 'unet0428_256_LeakyReLU.pth')
```

In [ ]:
```
h5f = h5py.File('unet0428_256_LeakyReLU_test.h5', 'w')
h5f.create_dataset('Test_pred', data=test_pred)
h5f.create_dataset('Test_labels', data=test_labels)
h5f.close()
```

Results Analysis

In [18]:
```
import numpy as np
import h5py
h5f = h5py.File('unet0428_256_LeakyReLU_test_epoch10.h5','r')
test_pred = h5f['Test_pred'][:]
test_labels = h5f['Test_labels'][:]
```

In [19]:
```
np.sum(test_labels),np.sum(test_pred),np.sum(test_labels)-np.sum(test_pred)
```

Out[19]:
```
(1663833000.0, 1518282600.0, 145550340.0)
```

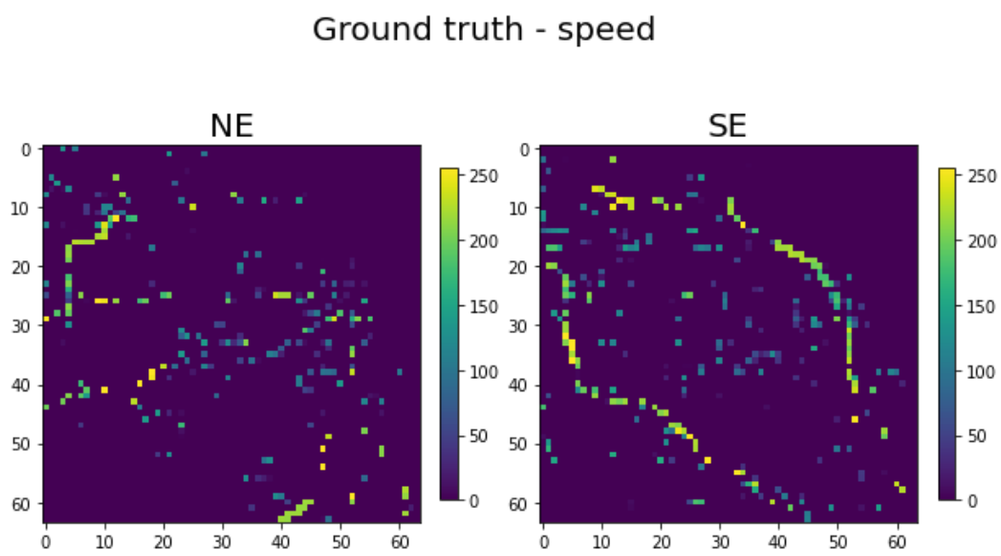
In [20]:
```
test_AE = ((test_pred-test_labels)**2)**0.5
test_AE = np.reshape(test_AE, (-1,))
test_AE = np.array(test_AE)
test_AE.shape, np.mean(test_AE)
```

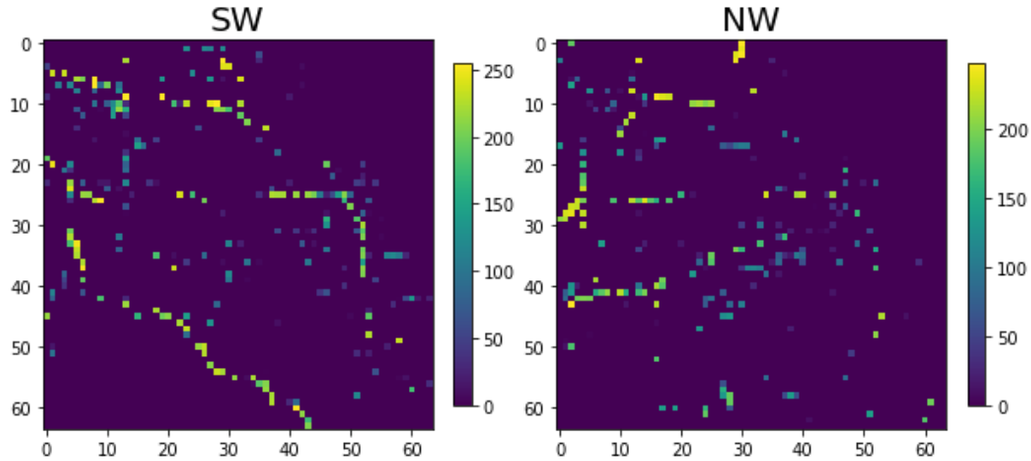
Out[20]:
```
((282656768,), 6.868107)
```

In [21]:
```
test_SE  = (test_pred-test_labels)**2
test_MSE = np.mean(test_SE)
test_MSE
```

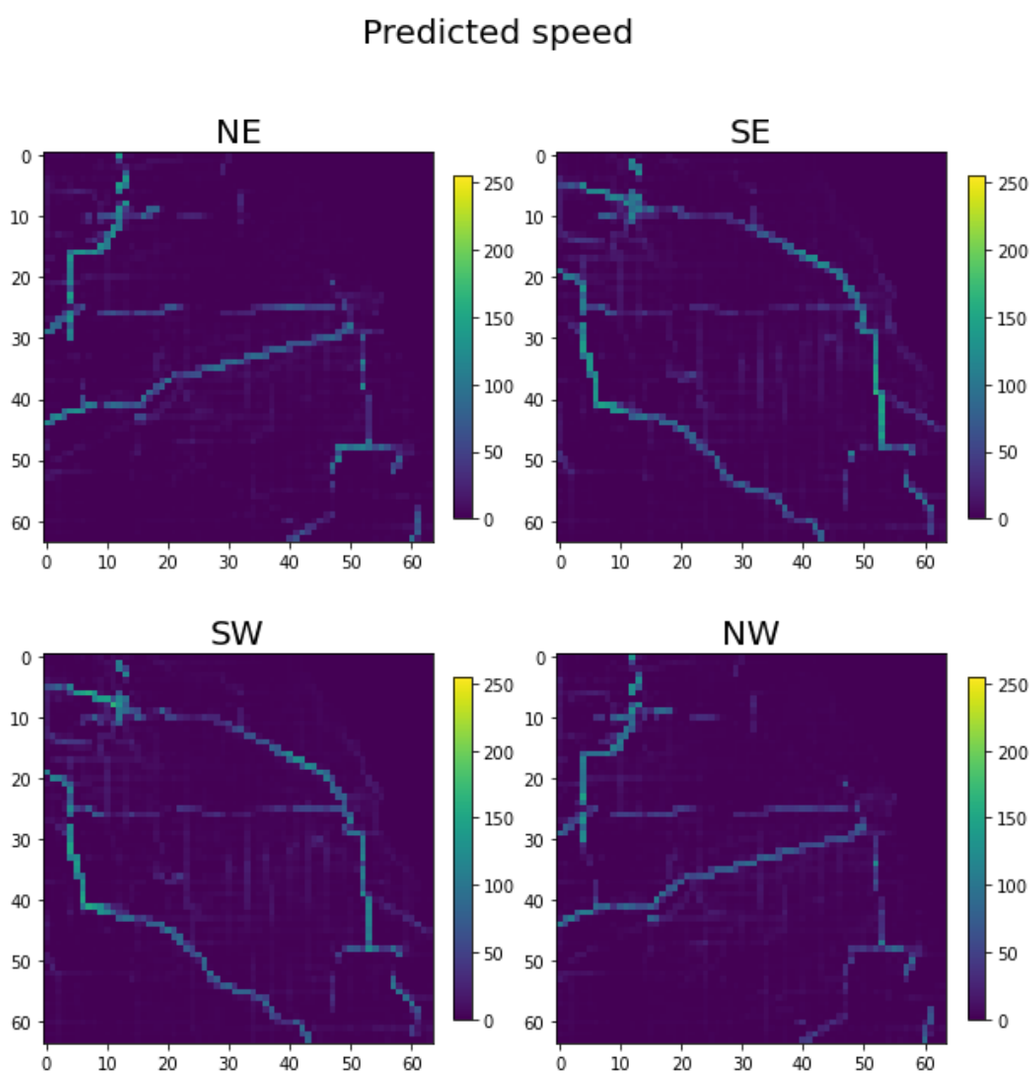
Out[21]:
```
472.80136
```

In [29]:
```
import matplotlib.pyplot as plt
DIR = ['NE','SE','SW','NW']
#2019-06-01 9 am
plt.figure(figsize=(10,10))
for i in range(4):
  X = np.array(test_labels[103*9,i+4,:,:])
  ax = plt.subplot(2,2,i+1)
  ax = plt.imshow(X,vmin=0)
  plt.colorbar(ax.colorbar, fraction=0.04)
  plt.title(str(DIR[i]),fontsize=20);
plt.suptitle('Ground truth - speed',fontsize=20)
plt.show()
```





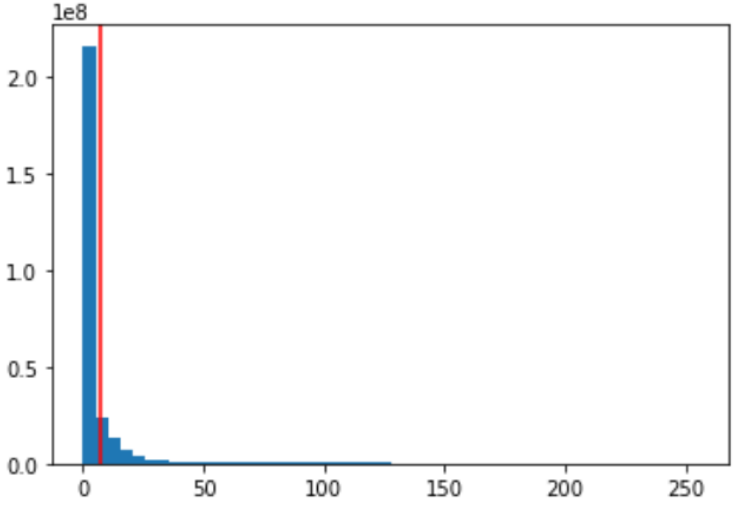
In [30]:
```
DIR = ['NE','SE','SW','NW']
#2019-06-01 9 am
plt.figure(figsize=(10,10))
for i in range(4):
  X = np.array(test_pred[103*9,i+4,:,:])
  ax = plt.subplot(2,2,i+1)
  ax = plt.imshow(X,vmin=0, vmax=255) # vmin=0, vmax=255
  plt.colorbar(ax.colorbar, fraction=0.04)
  plt.title(str(DIR[i]),fontsize=20);
plt.suptitle('Predicted speed',fontsize=20)
plt.show()
```



In [9]:
```
test_MAE=np.mean(test_AE)
```

In [16]:
```
import matplotlib.pyplot as plt
plt.hist(test_AE,bins=50)
plt.axvline(x=test_MAE, ymin=0, ymax=1,color='r')
```

Out[16]: <matplotlib.lines.Line2D at 0x21ec8a36040>



In [17]:
```
plt.hist(test_AE[test_AE<50],bins=50)
plt.axvline(x=test_MAE, ymin=0, ymax=1,color='r')
```

Out[17]: <matplotlib.lines.Line2D at 0x21ec85c9bb0>