**Homework #6 The outline:**

Supasin Wuthikulphakdi (Richard Kuo) #608785068

**Title of the Thesis (official):**

Applications of Machine & Deep Learning in GIS – Spatiotemporal data mining

**Introduction:**

Spatiotemporal data mining (STDM) is a form of GIS data analysis done in both the spatial and the temporal domains. A CNN learns the spatial graph input, while and LSTM/RNN learns the temporal domain of the input data. With both DNN working together, we can forecast a spatiotemporal data (such as traffic, train ridership, etc.) not only the time series (time domain), but also the heatmap (space domain).

My thesis is focused on paper survey of spatiotemporal data mining. A CNN-GCN (Graph Convolutional Neural Network) is used to extract features in the spatial domain (i.e. heatmap). An LSTM-RNN (Long Short-Term Memory RNN) is used to extract features in the temporal domain (i.e. change of data spot coordinates and amplitudes overtime). According to Wang et al [8], with the AI, there are some uses in Spatiotemporal data mining:

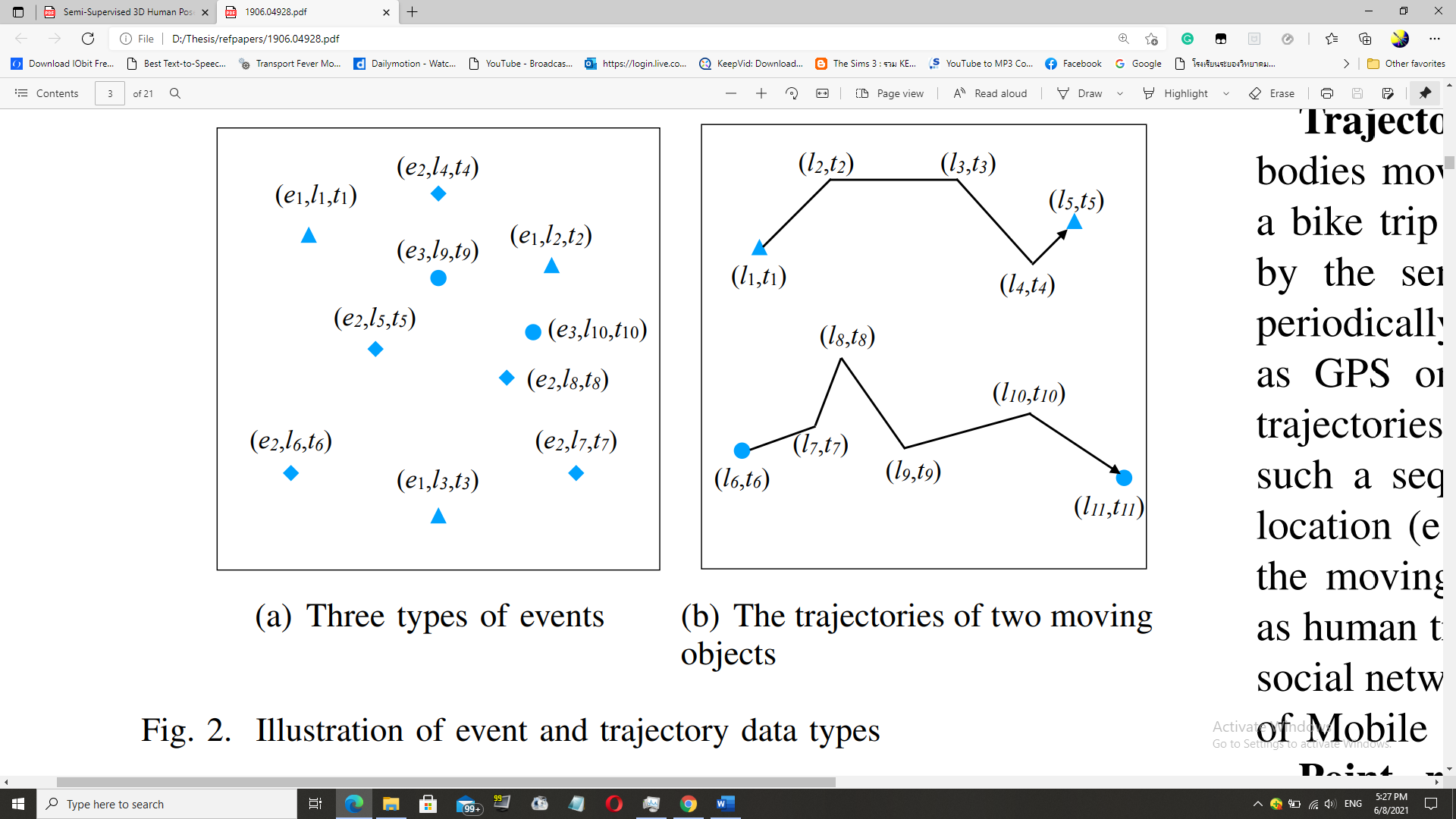
* In survey engineering, some researchers use CNN observe changes in land use, crop growth, and construction progresses.
* To make GPS based pathfinding more accurate, Spatiotemporal Data Mining is often utilized in several papers to train the pathfinding ML/DL-architectures.
* In transportation, deep learning methods learn highly intricate ST-correlations among the traffic data – useful in some tasks such as traffic flow prediction, traffic incident detection, and traffic congestion prediction.
* In On-demand service, such as Uber [10], to perform pathfinding
* All the papers referred, may be added & updated in the future, involve GIS, Computer Vision (CV), and spatiotemporal data mining.
  1. **Technical concepts of STDM**

According to [8], there are some fundamentals of STDM we need to know before proceeding any further, for example:

* Spatiotemporal Data Structures
* Data Instances
* Data Respresentations
* Deep Learning Models in STDM

**1.1.1 Event data.**

Event data comprise of discrete events occurring at point locations and times (e.g., crime events in the city and traffic accident events in a transportation network). An event can generally be characterized by a point location and time, which denotes where and when the event occurred, respectively. For example, a crime event can be characterized as such a tuple (ei , li , ti), where ei is the crime type, li is the location where the crime occurs and ti is the time when it occurs.



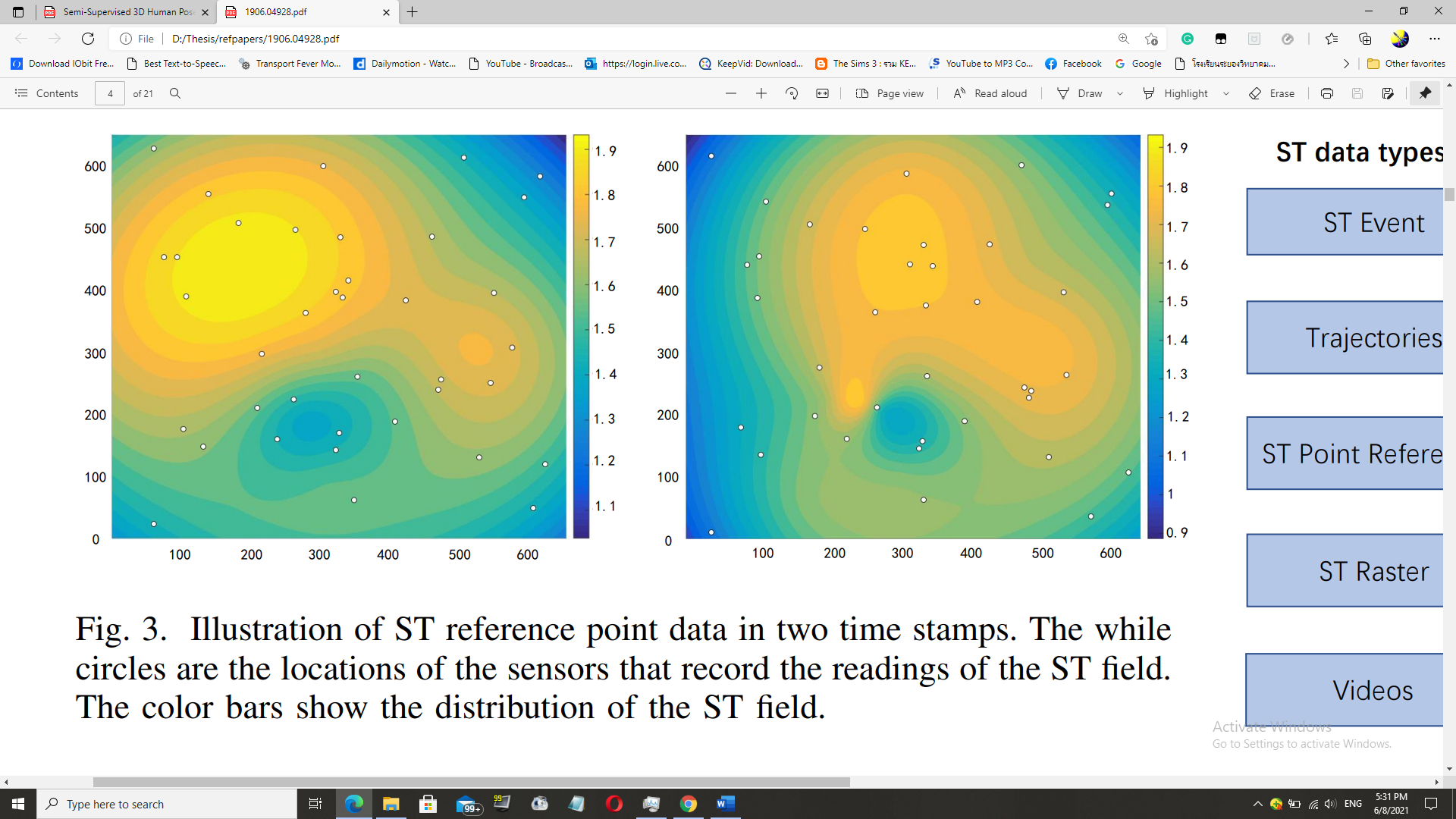
*Fig. 1. Illustration of event and trajectory data types*

**1.1.2 Trajectory data**

Trajectories denote the paths traced by bodies moving in space over time. (e.g., the moving route of a bike trip or taxi trip). Trajectory data are usually collected by the sensors deployed on the moving objects that can periodically transmit the location of the object over time, such as GPS on a taxi. Fig. 1(b) shows an illustration of two trajectories. Each trajectory can be usually characterized as such a sequence {(l1, t1),(l2, t2)...(ln, tn)}, where li is the location (e.g. latitude and longitude) and ti is the time when the moving object passes this location.

**1.1.3 Point reference data.**

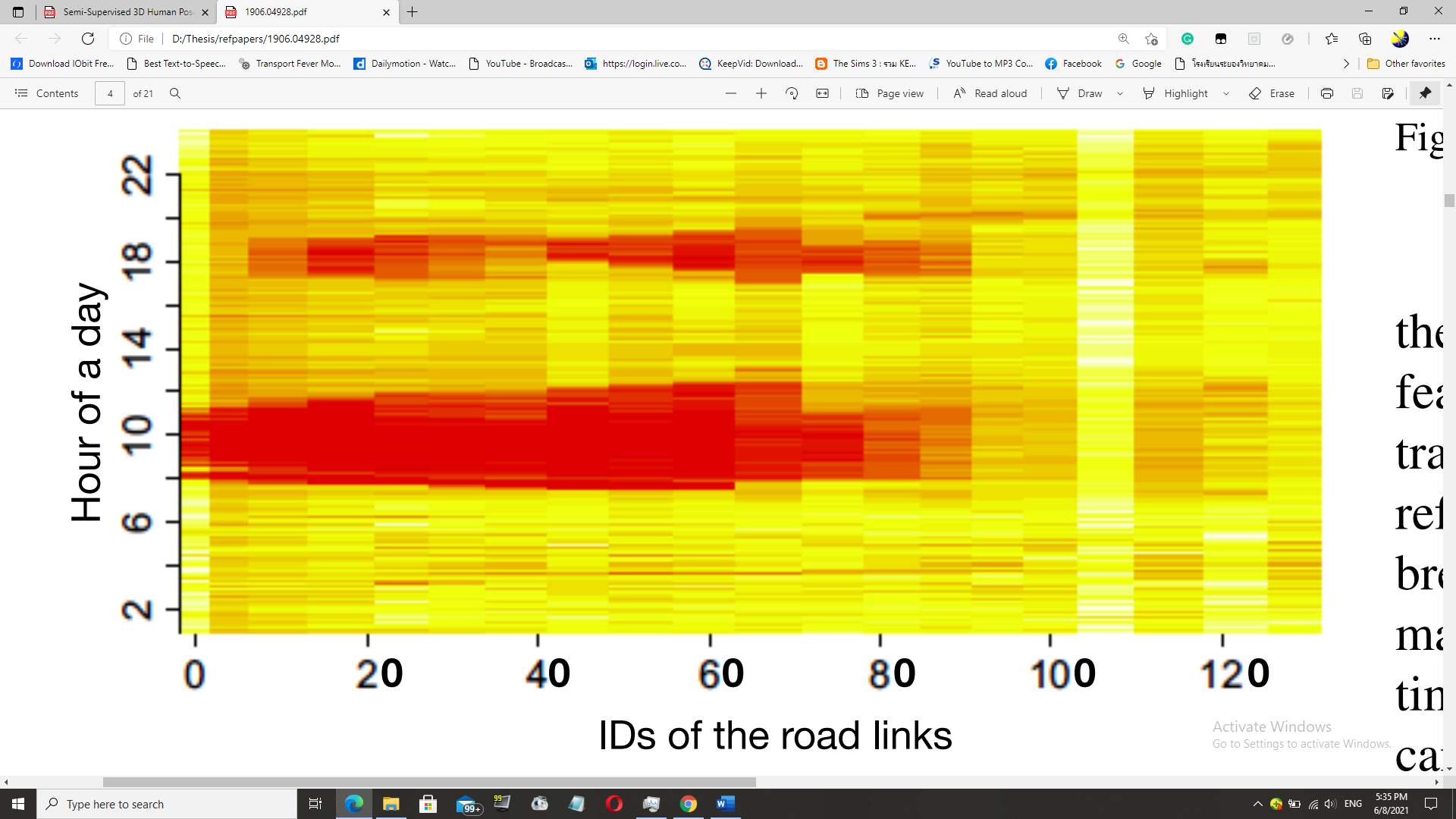
Point reference data consist of measurements of a continuous ST field such as temperature, vegetation, or population over a set of moving reference points in space and time. For example, meteorological data such as temperature and humidity are commonly measured using weather balloons floating in space, which continuously record weather observations. Point reference data can be usually represented as a set of tuples as follows {(r1, l1, t1),(r2, l2, t2)...(rn, ln, tn)}. Each tuple (ri , li , ti) denotes the measurement of a sensor ri at the location li of the ST filed at time ti .



*Fig. 2. Illustration of ST reference point data in two timestamps. The while circles are the locations of the sensors that record the readings of the ST field. The color bars show the distribution of the ST field.*

**1.1.4 Raster data.**

Raster data are the measurements of a continuous or discrete ST field that are recorded at fixed locations in space and at fixed time points. The major difference between point reference data and raster data is that the locations of the point reference data keep changing while the locations of the raster data are fixed. The locations and times for measuring the ST field can be regularly or irregularly distributed. Given m fixed locations S = {s1, s2, ...sm} and n time stamps T = {t1, t2, ...tn}, the raster data can be represented as a matrix Rm×n, where each entry rij is the measurement at location si at time stamp tj . Raster data are also quite common in real world applications such as transportation, climate science, and neuroscience. For example, Fig. 3 shows an example of the traffic flow raster data of a transportation network. Each road is deployed a traffic sensor to collect real time traffic flow data. The traffic flow data of all the road sensors in a whole day (24 hours) form a raster data.



*Fig. 4. Illustration of raster data collected from traffic flow sensors. The xaxis is the ID of the road links in a transportation network, and the y-axis is the hour of a day. Different colors denote different traffic flows on the road links captured by the road sensors deployed at fixed locations.*

**1.1.5 Video.**

A video that consists of a sequence of images can be also considered as a type of ST data. In the spatial domain, the neighbor pixels usually have similar RGB values and thus present high spatial correlations. In the temporal domain, the images of consecutive frames usually change smoothly and present high temporal dependency. A video can be generally represented as a three-dimensional tensor with one dimension representing time t and the other two representing an image. Actually, video data can be also considered as a special raster data if we assume that there is a “sensor” deployed at each pixel and at each frame the “sensors” will collect the RGB values.

* 1. **Data instances.**

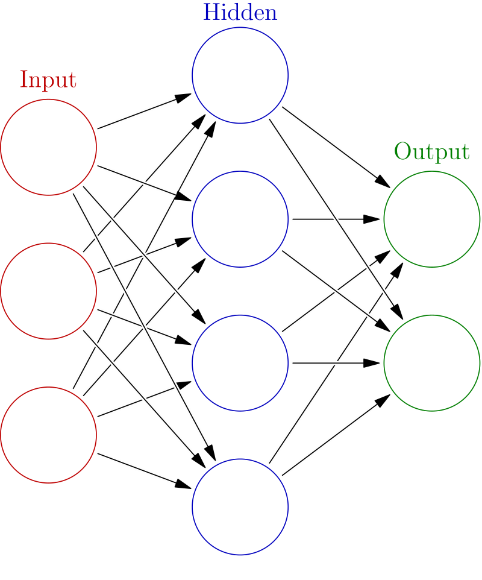
In general, the ST data can be summarized into the following data instances: points, trajectories, time series, spatial maps and ST raster as shown in the left part of Fig. 5. A ST point can be represented as a tuple containing the spatial and temporal information as well as some additional features of an observation such as the types of crimes or traffic accidents. Besides ST events, trajectories and ST point reference can also be formed as points. For example, one can break a trajectory into several discrete points to count how many trajectories have passed a particular region in a particular time slot.

Different data instances can be extracted from ST raster as time series, spatial maps or ST raster itself, depending on different applications and analytic requirements. First, we can consider the measurements at a particular ST grid of the ST field as a time series for some time series mining tasks. Second, for each time stamp the measurements of an ST raster can be considered as a spatial map. Third, one can also consider all the measurements spanning all the locations and time stamps as a whole for analysis. In such a case, ST raster itself can be a data instance.

* 1. **Data Representations.**

For the above mentioned five types of ST data instances, four types of data representations are generally utilized to represent them as the input of various deep learning models, sequence, graph, 2-dimensional matrix, and 3-dimensional tensor as shown in the right part of Fig. 5. Different deep learning models require different types of data representations as input. Thus, how to represent the ST data instances relies on the data mining task under study and the selected deep learning model.

* 1. **Deep learning models.**

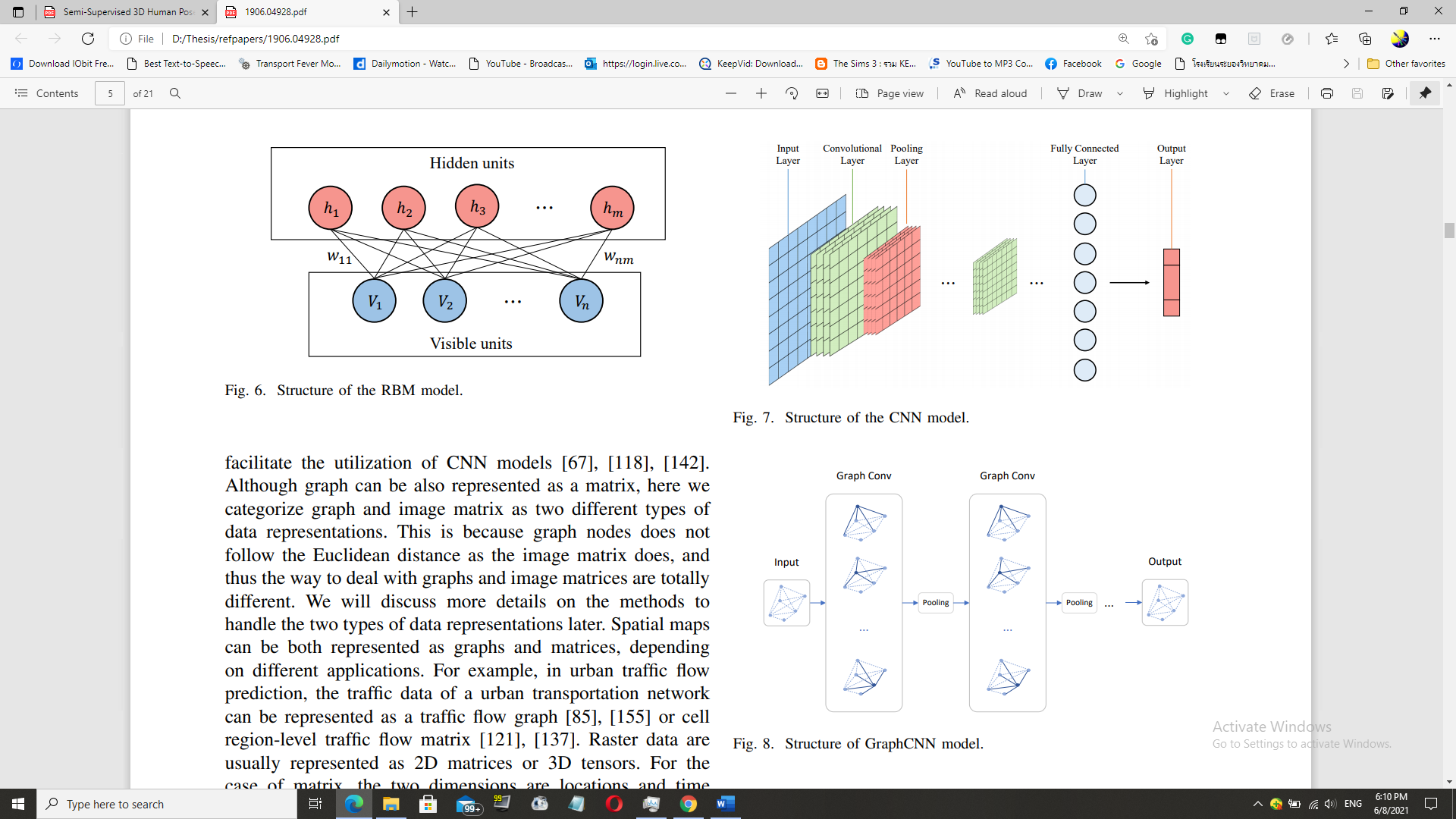
****

*Fig.5 a diagram of an Artificial Neural networks*

Deep learning models, praised for their outstanding performance and accuracy, are often utilized in several STDM papers. There are some notable examples below.

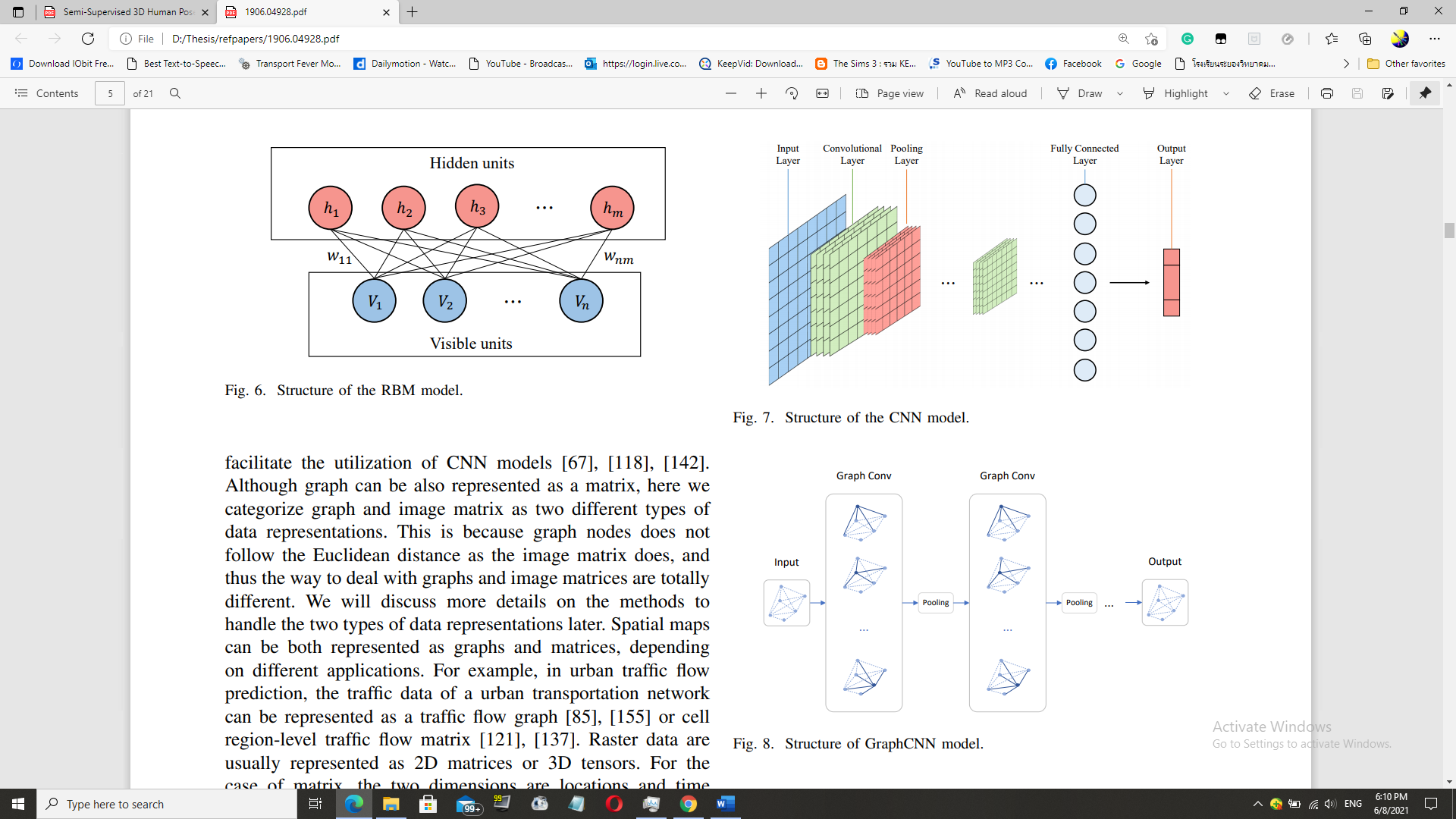
**1.4.1 Restricted Boltzmann Machines (RBM).**

A Restricted Boltzmann Machine is a two-layer stochastic neural network [53] which can be used for dimensionality reduction, classification, feature learning and collaborative filtering. As shown in Fig. 6, the first layer of the RBM is called the visible, or input layer with the neuron nodes {v1, v2, ...vn}, and the second is the hidden layer with the neuron nodes {h1, h2, ...hm}. As a fully connected bipartite undirected graph, all nodes in RBM are connected to each other across layers by undirected weight edges {w11, ...wnm}, but no two nodes of the same layer are linked. The standard type of RBM has a binary-valued nodes and bias weights. RBM tries to learn a binary code or representation of the input, and depending on the task, RBM can be trained in either supervised or unsupervised ways. RBM is usually used for learning features.



**1.4.2 Convolutional Neural Networks (CNN).**

Convolutional neural networks (CNN) is a class of deep, feed-forward artificial neural networks that are applied to analyze visual imagery. A typical CNN model usually contains the following layers as shown in Fig. 7: the input layer, the convolutional layer, the pooling layer, the fully connected layer and the output layer.



First, the convolutional layer will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. Second, the pooling layer will then downsample the spatial dimensionality of the given input to reduce the number of parameters. The fully connected layers will connect every neuron in one layer to every neuron in the next layer to learn the final feature vectors for classification.

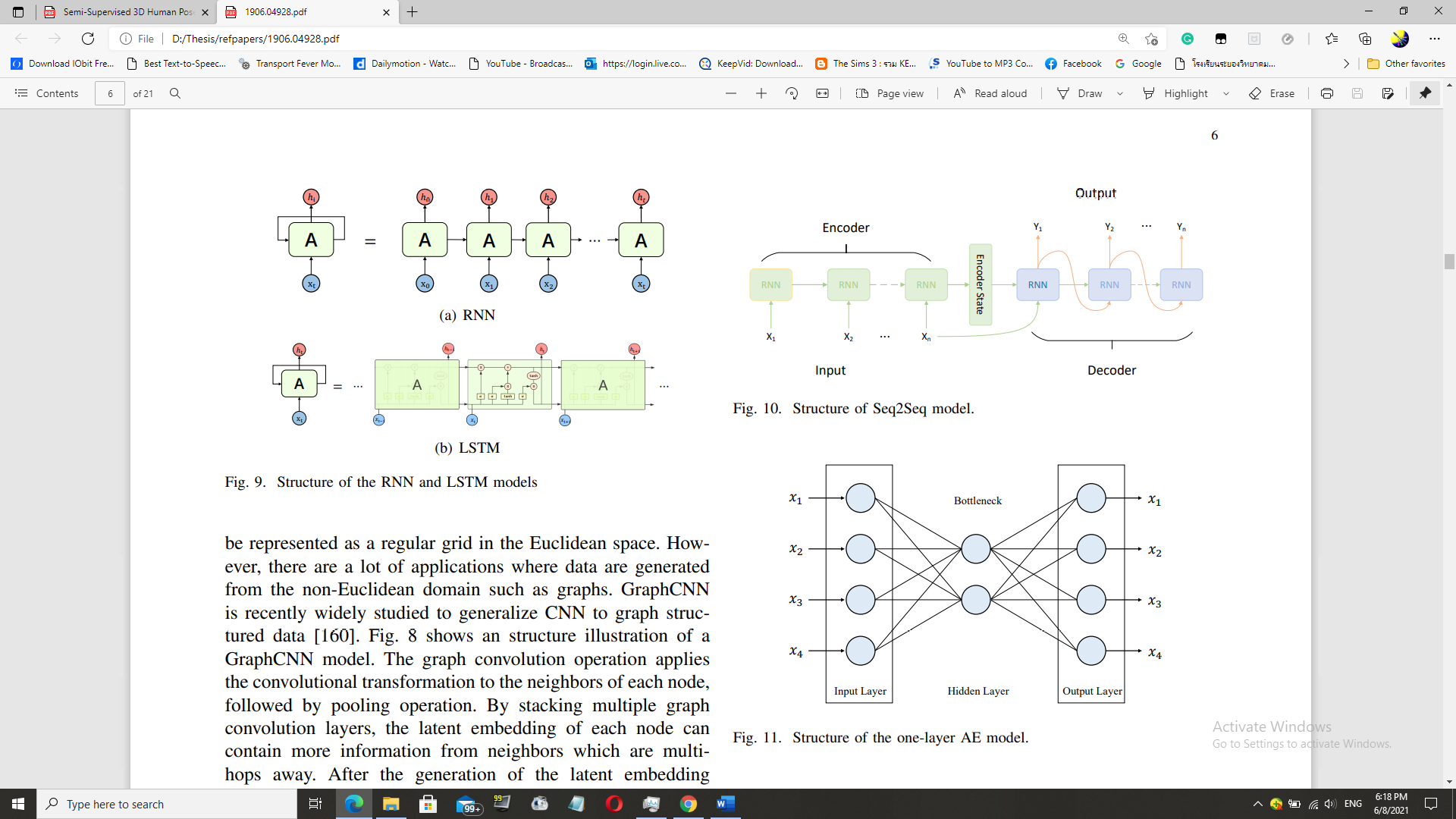
CNN is designed to process image data. Due to its powerful ability in capturing the correlations in the spatial domain, it is now widely used in mining ST data, especially the spatial maps and ST rasters.

**1.4.3. Graph CNN (GCNN).**

CNN is designed to process images which can be represented as a regular grid in the Euclidean space. Graph CNN is recently widely studied to generalize CNN to graph structured data [160]. Fig. 8 shows a structure illustration of a Graph CNN model. The graph convolution operation applies the convolutional transformation to the neighbors of each node, followed by pooling operation. By stacking multiple graph convolution layers, the latent embedding of each node can contain more information from neighbors which are multi hops away. After the generation of the latent embedding of the nodes in the graph, one can either easily feed the latent embeddings to feed-forward networks to achieve node classification of regression goals or aggregate all the node embeddings to represent the whole graph and then perform graph classification and regression.

**1.4.4 RNN and LSTM.**

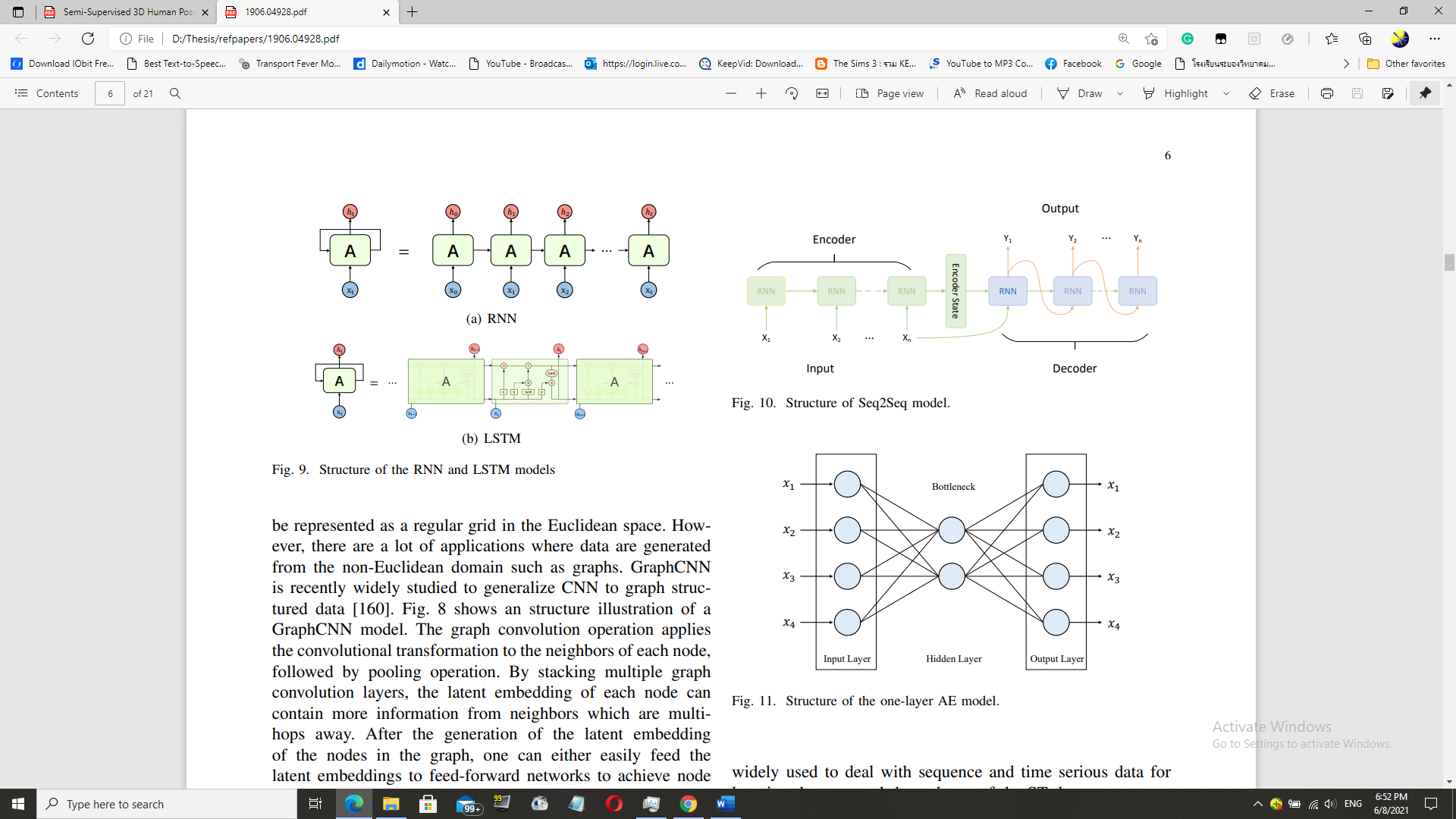
A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. RNN is designed to recognize the sequential characteristics and use patterns to predict the next likely scenario. They are widely used in the applications of speech recognition and natural language processing. Fig. 9(a) shows the general structure of a RNN model, where Xt is the input data, A is the parameters of the network and ht is the learned hidden state. One can see the output (hidden state) of the previous time step t−1 is input into the neural of the next time step t. Thus, the historical information can be stored and passed to the future.



A major issue of standard RNN is that it only has short term memory due to the issue of vanishing gradients. Long Short-Term Memory (LSTM) network is an extension for recurrent neural networks, which is capable of learning long term dependencies of the input data. LSTM enables RNN to remember their inputs over a long period of time due to the special memory unit as shown in the middle part of Fig. 9(b). An LSTM unit is composed of three gates: input, forget and output gate. These gates determine whether to let new input in (input gate), delete the information because it is not important (forget gate) or to let it impact the output at the current time step (output gate). Both RNN and LSTM are widely used to deal with sequence and time serious data for learning the temporal dependency of the ST data.

**1.4.5. Seq2Seq**.

A sequence to sequence (Seq2Seq) model aims to map a fixed length input with a fixed length output where the length of the input and output may differ [138]. It is widely used to various NLP tasks such as machine translation, speech recognition and online chatbot. Although it is initially proposed to address NLP tasks, Seq2Seq is general framework and can be used to any sequence-based problem. As shown in Fig. 10, a Seq2Seq model generally consists of 3 parts: encoder, intermediate (encoder) vector and decoder. Due to the powerful ability in capturing the dependencies among the sequence data, Seq2Seq model is widely used in ST prediction tasks where the ST data present high temporal correlations such as urban crowd flow data and traffic data.



**1.4.6 Autoencoders (AE)**

An autoencoder is a type of artificial neural network that aims to learn efficient data codings in an unsupervised manner [53]. As shown in Fig. 11, it features an encoder function to create a hidden layer (or multiple layers) which contains a code to describe the input. There is then a decoder which creates a reconstruction of the input from the hidden layer. An autoencoder creates a compressed representation of the data in the hidden layer or bottleneck layer by learning correlations in the data, which can be considered as a way for dimensionality reduction. As an effective unsupervised feature representation learning technique, AE facilitates various downstream data mining and machine learning tasks such as classification and clustering. A stacked autoencoder (SAE) is a neural network consisting of multiple layers of sparse autoencoders in which the output of each layer is wired to the inputs of the successive layer [7].

**2. Literature Review (More to be added):**

Please view the file “Literature\_Review\_June\_1” for reference [10-12]

According to Doshi et al, 2018 [1], We propose to identify disaster-impacted areas by comparing the change in man-made features extracted from satellite imagery. Using a pre-trained semantic segmentation model from [2] we extract man-made features, the pre- and post-event images on the “before, during and after” imagery of the event-affected area.

According to Amit et al. [3], CNN is a sequence of layers, the convolution layer (who detects features from a data image), the pooling layer (downsamples the input), and the FC layer (who classifies the features detected earlier). with ReLU as the main activation function of the network. Further explained in the section 2 of [3].

Iglovikov et al. 2017 [4], used an FC-CNN named U-NET, along with an embedded multispectral sensor, which detects frequency reflection by the objects, to detect geo-features in satellite images and yielded satisfying results.

According to Bochkovskiy et al. [5], with the help of YOLO V.4, and TensorFlow Keras, CNN’s performance in image recognition is improved. So that it would be our main CNN model in this thesis. We hope that our deep learning model, written in Python, will work as the goals above.

In Terms of video and sequence type photos (such as slideshow), however, the use of LSTM-RNN is needed. According to Fang et al. [6], LSTM is excellent at predicting flood because it could process time series data.

According to Li et al, 2018. [7], spatiotemporal forecasting is a crucial task for a learning system that operates in a dynamic environment. It can be useful in pathfinding, autonomous vehicles, logistics, city planning etc. They used a Diffusion Convolutional Recurrent Neural Network (DCRNN) model to forecast the road traffic within a specific space and timeframe (The dataset was METR-LA, 2014). Diffusion convolution extracts the traffic features, and the RNN processes the traffic volumes in sequence.

Wang et al, 2019. [8], surveyed and collected several papers about spatiotemporal data mining. And explained the fundamentals and the concepts of STDM.

Yu et al, 2018. [9] proposed a Spatiotemporal Graph Convolutional Networks (STGCN), to tackle the time series prediction problem in traffic domain. They formulated the problem on graphs and build the model with complete convolutional structures, enabling much faster training speed with fewer parameters. Compared with existing models, STGCN more effectively captured comprehensive spatiotemporal correlations through modeling multi-scale traffic networks and consistently outperforms state-of-the-art baselines on various real-world traffic datasets.

Correa et al, 2017 [12] performed a spatiotemporal data mining of Taxi vs Uber ridership in NYC, 2014+15. According to the heatmap inside the paper, the ridership for both taxi systems depended on several factors – such as personal income, education, jobs, car ownership etc. With 3 spatial models for ridership prediction – linear, spatial error and spatial lag models, the last one outperformed not only the first 2 algorithms, but also yielded a considerable accuracy and performance.

Amato et al. [11] designed a deep learning-based architecture called “Empirical Orthogonal Functions principal component analysis” EOF-PCA in which the EOF framework decomposes the spatiotemporal input data, in terms of a sum of products of temporally referenced basis functions and of stochastic spatial coefficients which can be spatially modelled and mapped on a regular grid. Then, the input layer spatial covariates are processed by a “Fully Connected Neural Network” (FCNN) to obtain predictive coefficient to be recomposed altogether with the decomposed data stream, obtaining a spatiotemporal signal reconstruction.

According to [11], the authors stated that their problem is the spatiotemporal data prediction (i.e., climate & environmental change) is not enough to be done with physically based models, so that is why Machine learning algorithms will do the prediction, which are efficient in analysing and modelling spatially and temporally variable environmental data. Deep learning methods, capable of feature auto-extraction (CNN in spatial domain, and RNN in temporal domain), is a promising approach to tackle this challenge. DL also could capture spatio-temporal dependencies through their automatic FRL. Despite the astounding efficiency of the DL architectures, the problem of the interpolation of continuous spatio-temporal fields measured on a set of irregular points in space is still under-investigated - and it still requires an adaptation effort for an environmental S.T. problem.

And the second problem is that it is arduous effort to handle a climato-environmental S.T. data – The data input in DL models are in both time and space domains, a set of irregular points in space—rather than with raster data. To tackle this gap, we introduce here a framework for spatio-temporal prediction of climate and environmental data using deep learning, which is briefly described above. The proposed framework will allow to reconstruct coherent spatio-temporal fields.

Tang et al. [12] designed an LSTM based framework to learn and forecast the rail traffic. The short-term forecast of rail transit is an issue in intelligent transportation system (ITS). Accurate forecast can forewarn travel outburst, helping the passengers with their travel plans. Even though the LSTM is notably effective in temporal data, it cannot correlate the time domain with the space domain. That is why we propose ST-LSTM. Compared with other conventional models, ST-LSTM network can achieve a better performance in experiments.

LSTM network is widely recognized as the most suitable model to deal with traffic forecast. LSTM unit has three gates, namely, input gate, forget gate, and output gate, which can adjust the state of unit dynamically, so LSTM network is able to capture the features on longer time span.

In this paper, the object of study is the short-term forecast of rail transit. With the trains having fixed station, uniform speed, and regular schedule, the spatial correlation between stations can be transformed into the time cost.

In this work, the data in the temporal (directly) and the spatial domains (With the TCM & SCM) before entering the FC-input-layers to combine them as the input. Then, we set the weight matrices & bias vectors for our ST-LSTM, and fine tune them to minimize the cost function output. Then, the ST-LSTM learns and captures the features of both domains. The training-testing data is of the space domain (100 stations) and the time domain (March 2017, daily) Compared with most existing methods, the proposed model has a better performance on accuracy and meets the real-time requirement.

Lu et al. [13], designed a spatial-temporal deep learning network, termed ST-TrafficNet, for traffic flow forecasting, whose architecture works as follows. 1. The Spatial Aware Multi-Diffusion Convolution Bloc (ADC-Block – who introduces Graph Attention Mechanism (GAM) into the MDC) uncovers unseen spatial dependencies from traffic graph signals automatically 2. From the data stream, the multi-diffusion convolution (MDC) block harvests ST-features of the spatial domain. 3. The ST-TrafficNet, an LSTM based framework, harvests the features of the temporal domain. 4. The output from both ANN are summed up to achieve convolutional results. And 5. The ST-TrafficNet is evaluated on two benchmark datasets and compare it with various baseline methods for traffic forecasting.

Intelligent transportation systems (ITSs) are gaining prominence in signal control, navigation, and guidance in road transport. Traffic forecasting, gathered in a form of a spatiotemporal dataset, is challenging due to the natural complexity and uncertainty of traffic patterns. E.g., we need several sensors to collect traffic data, which is collected as traffic graph signals, and there is a seasonality in the data itself. The idea of traffic forecasting is to extract S.T. dependencies of historic TGS to learn traffic patterns and make accurate predictions. There are some neural networks in ITS – the CNN (including GCN) captures the space domain of the input data, and the RNN (including LSTM) captures the time domain one. On the other hand, the GCN also has drawbacks – 1. the GCN-based approaches require a predetermined graph structure - the predetermined graph structure could have some mistakes under several conditions. 2. The socioeconomic-demographic factors also influence the raw input graph (a.k.a. input data) 3. Current traffic forecasting methods are ineffective to learn high-dimension temporal features (HDTF) of intricate traffic graph signals – so that it must work along with an LSTM, which still struggles to learn HDTF from traffic data.

To overcome these, The MDC block, performing diffusion convolution (i.e., forward, backward, and attentive), capture spatial dependencies in parallel with ST-TrafficNet (an LSTM) of the time domain. The ADC block introduces Graph Attention Mechanism (GAM) into the MDC to learn graph structure from traffic graph signals without prior graph structure knowledge.

To cope the nonlinearity of the traffic flow data during the holidays, Luo et al, 2019 [14], designed a discrete Fourier transform (DFT) and support vector regression (SVR) based machine learning model to predict the road traffic flow during the holidays in Jiangsu Province, China, on Tomb-sweeping Day and National Day from 2011 to 2015. With proper training, the model outperformed other ML models – like ARIMA, SVR and EMD-SVR. The model works as follows:

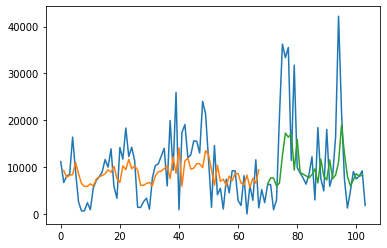
1. With DFT, the traffic flow time series are transformed from the time to the frequency domain, and the trend is extracted through setting the appropriate threshold. 2.Besides the holiday natural traffic growth, the model retains the change of the trend component (TC). Thus, the TC is predicted by extreme extrapolation of the historical trend. 3. For the residual, the fluctuation and burst are defined at first. The mean and variance of the fluctuation are stable, but the burst has great randomness. 4.With SVR, the residual is predicted with its burst preprocessed. And 5.The final prediction result can be obtained by combining the trend with residual prediction result. Their approach is summarized in the file “Literature\_Review\_June\_13.docx”.

Pan et al, 2019 [15], designed the ST-MetaNet to predict the urban traffic in Los Angeles using the METR-LA dataset, and achieved the MAE score of 16.9 and the RMSE of 34.0, outperforming several state-of-the-art models. The model works as follows:  
 1. The RNN captures the sequence of historical urban traffic. 2. The Meta-Knowledge Learner (MK-L), containing 2 FCNs - each for nodes (GPS coordinates) and edges (distance between the nodes), the MK is used to learn the weight of the GAT and RNN. 3. The GAT captures diverse spatial correlations by individually broadcasting locations’ hidden states along edges. And 4. The RNN capture diverse temporal correlations associated with the geographical information of locations.

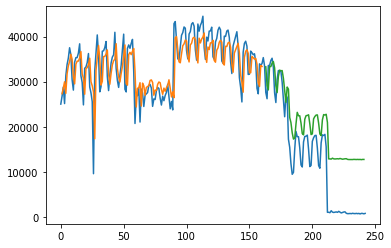
**Methodology**

In this paper, evaluated in the form of RMSE metric, we deploy different state-of-the art models from several referred papers, and test with the data they have been trained with. If applicable, we will then retrain them with the Taxi-Uber data while retaining the original settings they belong to.

In our demo model, to demonstrate how a Keras-LSTM works, we have written the code in appendix A/1, obtaining the visualization results in figure 12. As the architecture is too simple, the KLSTM produced too much error. For instance, the TAXI data achieved the Train Score of 4272.01 RMSE and Test Score of 8685.49 RMSE. For the UBER data, the model achieved the Train Score of 5827.10 RMSE Test Score of 10332.44 RMSE.



(A)



(B)

Figure 12: Keras-LSTM prediction results for (a)UBER and (b) TAXI ridership data

For every model deployed, the algorithms work similarly as follows (i.e. from [15]):

รูปภาพประกอบด้วย ข้อความ

คำอธิบายที่สร้างโดยอัตโนมัติ

**Project Phases:**

* May/June <-(CURRENT PHASE)
  + Perform Literature Reviews, Python coding & other essential skills
* July
  + Perform even deeper LR + coding practice + Validation runs.
* August/September
  + Custom Model Design, Training & Testing
* October
  + Final Validation Runs
* November
  + Result & Analysis
  + Writing a paper.
* December
  + Finish the paper.
* January
  + Defence oral exam

**Works done:**

* Read 4 reference papers
* Run the taxi-uber LSTM time series forecasting model written in Keras.
  + Model & coding are to be improved to extrapolate the future trends.

**To be determined/confirmed by the supervisor:**

* Official Title
* Dataset for:
  + Training
  + Testing
  + Validation
    - Kaggle: Taxi vs Uber Data
* Models & Architectures
  + LSTM
  + No. & Types/Roles of hidden layers
  + Activation Functions
  + Etc.
* Quantitative Metrics
  + K1 Score
  + Accuracy
  + Etc
  + RMSE^2.

**Progresses:**

Current Phase: Literature Review and Essential Skill Training

**References - Papers:**

[1] Doshi, J., Basu, S. and Pang, G., 2018. From satellite imagery to disaster insights. *arXiv preprint arXiv:1812.07033*.

[2] Doshi, Jigar. 2018. Residual Inception Skip Network for Binary Segmentation. Pages 216–219 of:

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.  
  
[3] Amit, S.N.K.B. and Aoki, Y., 2017, September. Disaster detection from aerial imagery with convolutional neural network. In 2017 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC) (pp. 239-245). IEEE.

[4] V. Iglovikov, S. Mushinskiy, and V. Osin, “Satellite Imagery Feature Detection using Deep Convolutional Neural Network: A Kaggle Competition,” vol. June, 2017.

[5] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” ArXiv, vol. abs/2004.10934, 2020.

[6] Fang, Z., Wang, Y., Peng, L. and Hong, H., 2020. Predicting flood susceptibility using LSTM neural networks. *Journal of Hydrology*, [online] p.125734. Available at: <https://doi.org/10.1016/j.jhydrol.2020.125734> [Accessed 18 April 2021].

[7] Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2017). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*.

[8] Wang, S., Cao, J., & Yu, P. (2020). Deep learning for spatio-temporal data mining: A survey. *IEEE Transactions on Knowledge and Data Engineering*.

[9] Yu, B., Yin, H., & Zhu, Z. (2017). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*.

[10] Correa, D., Xie, K., & Ozbay, K. (2017). Exploring the taxi and Uber demand in New York City: An empirical analysis and spatial modeling. In *96th Annual Meeting of the Transportation Research Board, Washington, DC*.

[11] Amato, F., Guignard, F., Robert, S., & Kanevski, M. (2020). A novel framework for spatio‑temporal prediction of environmental data using deep learning. Scientific Reports, 2020(10), 22243. doi:<https://doi.org/10.1038/s41598-020-79148-7>

[12] Tang, Q., Yang, M., & Yang, Y. (2019). ST-LSTM: A Deep Learning Approach Combined Spatio-Temporal Features for Short-Term Forecast in Rail Transit. Journal of Advanced Transportation, 2019, Article ID 8392592. doi:<https://doi.org/10.1155/2019/8392592>

[13] Lu, H., Huang, D., Song, Y., Jiang, D., Zhou, T., & Qin, J. (2020). ST-TrafficNet: A Spatial-Temporal Deep Learning Network for Traffic Forecasting. Electronics, 2020(9), 1474.

[14] Luo, X., Li, D., & Zhang, S. (2019). Traffic flow prediction during the holidays based on DFT and SVR. *Journal of Sensors*, *2019*.

[15] Pan, Z., Liang, Y., Wang, W., Yu, Y., Zheng, Y., & Zhang, J. (2019, July). Urban traffic prediction from spatio-temporal data using deep meta learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 1720-1730).

**Appendix A – Python Code:**

**Appendix A/1 – Keras Basic LSTM code for TAXI**

!wget https://raw.githubusercontent.com/Suppersine/Thesis2021/main/taxiuber/uberday.csv

# LSTM for international airline passengers problem with regression framing

import numpy

import matplotlib.pyplot as plt

import pandas as pd

import math

import tensorflow as tf

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

def create\_dataset(dataset, look\_back=1):

  dataX, dataY = [], []

  for i in range(len(dataset)-look\_back-1):

    a = dataset[i:(i+look\_back), 0]

    dataX.append(a)

    dataY.append(dataset[i + look\_back, 0])

  return numpy.array(dataX), numpy.array(dataY)

# fix random seed for reproducibility

numpy.random.seed(7)

# load the dataset

dataframe = pd.read\_csv('uberday.csv', usecols=[1], engine='python')

# to train the model with the taxi data, just rename to ‘taxiday.csv’

dataset = dataframe.values

dataset = dataset.astype('float32')

dataplot = dataframe.copy()

#training\_set = pd.read\_csv('shampoo.csv')

#dataplot = dataplot.iloc[:,1:2].values

#plt.plot(training\_set, label = 'Shampoo Sales Data')

plt.plot(dataplot, label = 'Taxi Daily Ridership Data (104/01-08)')

plt.show()

# normalize the dataset

scaler = MinMaxScaler(feature\_range=(0, 1))

dataset = scaler.fit\_transform(dataset)

# split into train and test sets

#here, you can change the test size to make future prediction

train\_size = int(len(dataset) \* 0.67)

test\_size = len(dataset) - train\_size

train, test = dataset[0:train\_size,:], dataset[train\_size:len(dataset),:]

# reshape into X=t and Y=t+1

look\_back = 1

trainX, trainY = create\_dataset(train, look\_back)

testX, testY = create\_dataset(test, look\_back)

# reshape input to be [samples, time steps, features]

trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# create and fit the LSTM network

model = Sequential()

model.add(LSTM(4, input\_shape=(1, look\_back)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(trainX, trainY, epochs=100, batch\_size=1, verbose=2)

# make predictions

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

# invert predictions

trainPredict = scaler.inverse\_transform(trainPredict)

trainY = scaler.inverse\_transform([trainY])

testPredict = scaler.inverse\_transform(testPredict)

testY = scaler.inverse\_transform([testY])

# calculate root mean squared error

trainScore = math.sqrt(mean\_squared\_error(trainY[0], trainPredict[:,0]))

print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(mean\_squared\_error(testY[0], testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))

# shift train predictions for plotting

trainPredictPlot = numpy.empty\_like(dataset)

trainPredictPlot[:, :] = numpy.nan

trainPredictPlot[look\_back:len(trainPredict)+look\_back, :] = trainPredict

# shift test predictions for plotting

testPredictPlot = numpy.empty\_like(dataset)

testPredictPlot[:, :] = numpy.nan

testPredictPlot[len(trainPredict)+(look\_back\*2)+1:len(dataset)-1, :] = testPredict

# plot baseline and predictions

plt.plot(scaler.inverse\_transform(dataset))

plt.plot(trainPredictPlot)

plt.plot(testPredictPlot)

plt.show()