CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

This project use Python 3.6 as the programming language to run the system. Before actually using the system, it need to have a dataset first to know what it deal with.

For the dataset schema (simulation schema), it use structure from Illustration 4.3. The entire system, including simulation and the real system use this same schema structure (excluding table simulation in real system) for consistency between testing and the real system.

I. Simulation Database

Database that this study use is Oracle Database (connection from python to oracle use cx_Oracle). The process of filling the tablespace and fetching the dataset carried out by using python script. Dataset is saved to a schema that held raw data.

First the script set the date when the entire system first starting, start date of the current cycle, and date when the program get invoked (current date). Then set the tablespace to be tracked and the table to save the raw dataset.

This program runs for *x* days (not *real* days, only a simulation, 40 days in this study). For each day, it runs *x* times (5 times in this study: 9 a.m., 12 noon, 3 p.m., 6 p.m., 9 p.m.) to add rows to table simulation and fetch data sets to schema. For each days and times iteration get added using timedelta with current date.

To calculate how much rows to fill in the table that belongs to tracked tablespace, it use the function in Code 1.

```
def totIterations(self):
    baseIter = random.randrange(2000,3000)
    baseAdd = 0
    baseExtra = 0
    if self.dt.day in range(1,12) or self.dt.day in range(23,32):
        if self.dt.day in range(23,32):
            if self.dt.month == 12 :
                beda = datetime(self.dt.year + 1, 1, 13) - self.dt
            else:
                beda = datetime(self.dt.year, self.dt.month +1 , 13)
                        - self.dt
            beda = float(beda.days / 10)
        else:
            beda = float(( 13 - self.dt.day ) / 10)
        print(f'Beda : {beda} hari')
        baseAdd = int(beda * random.randrange(2500,3000))
        print("Tanggal muda")
        if self.dt.weekday() in range(5,7):
            print('weekend')
            """ Jika weekend """
            rng_list= ["A"]*60 + ["B"]*20 + ["C"]*20
            chosen = random.choice(rng_list)
            if chosen is "A":
                baseExtra = random.randrange(6000,7001)
            elif chosen is "B":
                baseExtra = random.randrange(5000,6001)
            else:
                baseExtra = random.randrange(4000,5001)
        else:
            print('weekday')
            """ Jika weekday """
            rng_list= ["A"]*50 + ["B"]*25 + ["C"]*25
            chosen = random.choice(rng_list)
            if chosen is "A":
                baseExtra = random.randrange(3500,4001)
            elif chosen is "B":
                baseExtra = random.randrange(3000,3501)
            else:
                baseExtra = random.randrange(2000,3001)
    else:
        print("Tanggal tua")
        selisih = abs(23-self.dt.day)
        baseAdd = selisih * random.randrange(200,401)
        if self.dt.weekday() in range(5,7):
            print('weekend')
            """ Jika weekend """
            rng_list= ["A"]*60 + ["B"]*20 + ["C"]*20
            chosen = random.choice(rng_list)
            if chosen is "A":
                baseExtra = random.randrange(4000,4501)
            elif chosen is "B":
                baseExtra = random.randrange(2000,3001)
            else:
                baseExtra = random.randrange(100,1001)

    return baseIter+baseAdd+baseExtra
```

Code 1

There's 3 part that affecting how much row filled to tablespace: baseIter (line 2, random from 2000-3000 rows, base of how much rows will be filled everytime this function get called), baseAdd, and baseExtra.

Total rows for baseAdd depend on whether it's few weeks after pay day (where people have influx of money and use it to buy things, start from 23 to 11) or the days when the retail company get less buyer (12 to 22 in this study).

And last, one key part that influence total input rows is day of the week. If the current day is weekend, baseExtra will be updated, regardless of few weeks after pay day or not. The amount, however, will be affected if it's few weeks after pay day or not. If it's few weeks after payday and it's weekdays, still add baseExtra, with fewer amount than if it's weekend.

Finally add baseIter, baseAdd and baseExtra then return it to function caller (main program).

Then, after the script calculate how much rows to fill in the table that belongs to tracked tablespace, instert random value to the table matched with the data type each rows.

After filling the simulation table is done for this iteration and doesn't result in error (database full or unable to extend; see oracle error code 1653 or 1654), get and save maximum size, free space, date, and others that's in Illustration 4.3 and leave id_output in tblinput empty since it's not full yet.

If filling the simulation table result in oracle error 1653 or 1654, first get and save maximum size, free space, date, and others that's in Illustration 4.3 and leave the current id_output empty. Then insert the tbloutput with current date and get the id_output. Third, update all rows in tblinput with empty id_output (rows which doesn't know when it will be full) and fill it with id_output from the current date in tbloutput. Finally, stop the script since we get the datasets to train.

This entire process from initializing the script until it's full can be called a cycle (from newly added datafile until tablespace is full). When a cycle is done, we can continue the next cycle by creating new datafile in tracked tablespace and run them again with start date of the current cycle and date when the program get invoked modified to the next day the past cycle ended.

In summary:

a) Loop the program over days and hours

b) Get total rows to be inserted each time

c) Insert rows with random input for total rows in step 2

d) If step 3 is not interrupted, insert to dataset the current data (name of tablespace, current date, date when the cycle start, and date when the tablespace get tracked), maximum size and free space, then leave the output empty because it's not full yet.

e) If in step 3 oracle unable to extend (oracle error code 1653 or 1654, update the size of the current time, update all last data with empty output with id_output of current date.

f) The cycle is over if step 5 is executed. Continue to step 1 again if set 4 is executed.

II. The System

This system is automated part (getting dataset and training) and manual part (setting and predicting using current data) of this project that can be implemented in real world scenario. This system is used for Oracle Database. All the system use Python as the programming language. Database administrator only need to launch the settings and set the cron job. Administrator can use the prediction system after a cycle has passed (model already trained once).

a) Mining Dataset Process

This mining process consists of 2 part, settings process and mining dataset process.

1. Settings process

Settings process used to set up and change the required paramater for the entire system. This settings are saved as json file.

Settings program works like this:

1) Check if configuration file exists in settings directory
2) If step 1 is true, modify current settings (model folder, database, threshold, alert, turn tracking on/off). This modify settings use menu and submenu that modify group up settings for ease of use.
3) If step 1 is false, continue to the next step.
4) Create new configuration file in settings directory.
5) Set the folder (create if not exists) which the Deep Neural Network model will be placed.
6) Set the settings needed for database, which are:
    a) Hostname (default localhost)
    b) Port (default 1521)
    c) Service name (in target database)
    d) Tablespace to be tracked
    e) Schema username and password (encrypted)

7) Set the threshold size to mark the tablespace as full if the current size goes below this threshold size. This will invoke training process and alert in the mining process. The threshold size can use precentage (%) or byte (Byte, Kilobyte, Megabyte, Gigabyte, and Terabyte)

8) Set the alert (sender email and password, with recipient email that will get the alert). This data is ecrypted.

After settings is done, Database Administrator need to set the cron job to run the main program (Mining process and other process that take care of automating the task including alerting and invoking training process).

2. Mining process

This mining process executed using cron job. Mining program do it's task automatically after configuration have been set.

First initialize main class (set global variable for configurations, current date, and class to fetch data. After initializing main class, check if tracking mode in configuration is on, if it doesn't, terminate the program. Then start the main process to save the data, if it goes below threshold invoke alarm and train using last dataset done, close the connection to database.

```
def start(self):
    """ Function to start mining process """
    tblspc_data = self.gtdata.start() # start to save data
    """ Check if free space is below threshold"""
    if tblspc_data[0]:
        """Check if the alert module is on"""
        if self.config["alert"]["alert_on"]:
            cur_data = tblspc_data[1] # get current space
            if len(cur_data)>0:
                self.alert(cur_data) #alert admin
        if len(tblspc_data) == 3:
            self.train(True) # Train using new data (repeat)
        else:
            self.train(False) # Train using new data
```

Code 2

The main process (see Code 2) is the key part to control the entire process. Line 3 starts the process to process all the data and insert it to schema, including the size and date. If the tablespace size is below threshold, gtdata.start() return true at index 0 in list(line 5). If the alert is on, get current space (maximum and free), and send alert using email (line 8-10). Then if past data have no empty id_output, and current data is goes below threshold (when there's no empty id_output in schema), continue training using this new data. If the current cycle wasn't full(there's empty id_output), continue to train using this cycle.

The first thing class function gtdata.start() in Code 2 line 3 do is to get the current size (maximum size and free space) and compare it against the threshold in configuration. Current cycle considered full if the current free space goes below threshold. If the current data considered full and past data with empty output exists, insert current data to dataset schema including output of current date and then update all data in this cycle marked using empty output with current date. If the current data considered full and past data with empty output doesn't exists, insert data to dataset schema with output of last inserted data's output (data still considered full from last cycle).

```
1. def train(self, repeat = False):
2.     """ Function untuk training NN """
3.     diff = self.get_last_dataset() # get start date that isn't in file
4.     if diff:
5.         if len(diff)>0:
6.             dset = cDataset(self.config)
7.             dset.start(diff,False) # make the dataset (csv file)
8.             dset.close()
9.             pwd = os.path.dirname(os.path.realpath(__file__))
10.             for dt in diff:
11.                 """ Train untuk dataset terbaru yang belum ditraining """
12.                 train = Train(dset.id_tblspc, dt, pwd,
   self.config["files"])
13.                 reset = train.start()
```

```
14.         print("Training done")
15.     else:
16.         if repeat:
17.             lst_ds = self.gtdata.get_all_dataset()[-1][0]
18.             dset = cDataset(self.config)
19.             dset.start(lst_ds,True) # add dataset row (file csv)
20.             dset.close()
21.             pwd = os.path.dirname(os.path.realpath(__file__))
22.             train = Train(dset.id_tblspc, lst_ds, pwd, self.config["files"])
23.             reset = train.start()
24.             print("Training done")
25.         else:
26.             print("Couldn't continue training")
```

Code 3

Function train in the main work as controller for the entire training process. In line 3, get cycle that has not been converted to dataset file. When it found cycle that hasn't been converted yet, execute line 6 to 14. When it doesn't found cycle that hasn't been converted yet, check if it repeats is true (it's already full in the last input), then run line 16 to 23.

Line 6 to 8 create dataset file using the raw data from dataset schema (line 7). After processing and converting to dataset files, train the neural network (train.start()) using these new dataset file(s) (line 10 - 13).

When the function doesn't find any cycle that hasn't been converted yet and repeats, first it get last cycle date of the last input (line 17). Then update the cycle dataset file and add new row of the last input data (current data) in line 18 to 20. Finally, train using this cycle data (line 22-23).

Creating dataset file class takes all data for a cycle, process the data to get needed input for the neural network, and then save it as CSV file.

b) Training Process

This training process only get invoked from mining dataset process. The time this get invoked is when the mining dataset process detect that the current size is considered full.

```
def start(self):
    in_total = 6
    train_x = self.dataset[:,0:in_total]
    train_y = self.dataset[:,in_total]
    test_x = self.test[:,0:in_total]
    test_y = self.test[:,in_total]
    if not os.path.isfile(self.pathdir+"/"\
            +self.model_storage+"/model.h5"):
        self.set_model(in_total)
    else:
        print("Using past model")
        self.model = None
        self.model = self.load_mdl(self.pathdir, \
                                   self.model_storage)
    history = self.model.fit(train_x, train_y, \
                validation_data=(test_x, test_y), \
                epochs=150, batch_size=10, verbose=0)
    _,train_mse = self.model.evaluate(train_x, train_y, \
                    verbose=1)
    _,test_mse = self.model.evaluate(test_x, test_y, \
                    verbose=1)
    print('Train: %.3f, Test: %.3f' % (train_mse, test_mse))
    with open("train_nn/loss.csv",'a') as writeFile:
        writer = csv.writer(writeFile)
        for row in history.history['loss']:
            writer.writerow([row])
    writeFile.close()
    with open("train_nn/val_loss.csv",'a') as writeFile:
        writer = csv.writer(writeFile)
        for row in history.history['val_loss']:
            writer.writerow([row])
    writeFile.close()
    predictions = self.model.predict(train_x)
    correct = 0
    false = 0
    one_day_diff = 0
    two_day_diff = 0
    more = 0
    for i in range(len(train_x)):
        if int(predictions[i][0]) != int(train_y[i]):
            if abs(int(predictions[i][0]) \
                - int(train_y[i])) >=3 :
                more = more + 1
            else:
                if abs(int(predictions[i][0])\
46.                 - int(train_y[i])) == 1:
47.                 one_day_diff = one_day_diff + 1
48.             else:
49.                 two_day_diff = two_day_diff + 1
50.         false = false+1
51.     else:
52.         correct = correct+1
53. print(f'Correct = {correct} ; False = {false}')
54. print(f'1 day diff = {one_day_diff}; 2 day diff = '\
55.         f'{two_day_diff}; more or equal than 3 day = '\
56.         f'{more}')
57. acc = (correct+one_day_diff) / (correct+false) *100
58. print(f'Accuracy (with 1 day tolerance): {acc}')
59. self.save_mdl(self.pathdir, self.model_storage)
60. return False
```

Code 4

The process in Code 4 control the entire training process. Training process trains the neural network, evaluate the performance and save the model.

Explanation for Code 4 can be looked in table below:

Line	Explanation
2	This variable tells how much input is provided for neural network
3-4	Divide numpy array for training to input and output.
5-6	Divide numpy array for validating to input and output. This output get chosen randomly and 20% of training input in the class initialization.
7-9	If model doesn’t exists in the model storage, create a new model (Code 45)
10-14	If model does exists in the model storage, load the model.
15-17	Fit/train the model. The neural network doing forward-propagation through the dataset for 100 epoch and update the weight and bias (back-propagate) every 5 data.
18-21	Evaluate the model using train and validating dataset. Store the loss in a list.
23-32	Save the training loss and validating loss in files for plotting later.
33	Predict using training data
34-58	This lines count each correct, one day until more, and count accuracy with one day tolerance
59	Save the model in model folder. For future training or predict.

```
def set_model(self, in_total):
    alph = 0.01
    self.model = Sequential()
    self.model.add(Dense(2*in_total,
                        input_dim=in_total))
    self.model.add(LeakyReLU(alph))
    self.model.add(Dense(int(in_total+(in_total/2))))
    self.model.add(LeakyReLU(alph))
    self.model.add(Dense(in_total))
    self.model.add(LeakyReLU(alph))
    self.model.add(Dense(1))
    self.model.add(LeakyReLU(alph))
    opt = Adam(lr = 0.001)
    self.model.compile(loss='mse', optimizer=opt, metrics=['mse'])
```

Code 5

Function set_model() make a new keras Sequential model. The Deep Neural Network consists of 4 layers (n input neurons for input layer, 3 hidden layer with shape of diamond {2 x total input → total input + (total input/2) → total input}, and 1 output neurons.). In between each layer, there's leaky relu layer to make it linear and so it won't have dead neurons problem. For the loss function it use mean squared error. For the optimizer, it use adam optimizer to reduce loss faster (this use default learning rate from the original paper).

c) Predicting Process

Predicting process get execute manually by the administrator. Before predicting, the process start by initialing needed package, checking connection to database, save the required global variable to run the predicting process, and load the neural network model. It also check if model exists, if it doesn't, terminate the process.

After all preparation done, start by fetching the data of tablespace(maximum size and free space). Then prepare the data so it can be fed through the neural network model for prediction. Processed data (data type and order must be the same as in training process) then get run through the model to predict values of days left until the current data is full. This value is the result of only forward-propagating the data. Finally, show the result of prediction.

5.2 Testing

For testing, the main program is modified to only run to create dataset and train them. Then change the configuration file's tracked tablespace to target simulation tablespace using settings. After that, change all the query using user's schema, for example modify create dataset class to fetch the data from the simulation schema instead of the current login schema.

The main program only being run once only to convert all dataset schema to dataset file then train them. After running main program, run the added function in predict program (Code 6), finally plot the mean squared error loss to a graph (Code 7). This entire process train using the entire dataset file in the directory, which is all finished cycle in the dataset schema from corresponding tablespace.

After running the process above and finished only once, create and run testing train process to train and restrict the training to run certain range of cyles. Then, the next process is to run added function in predict program then plot the loss.

To test the performance for tweaking the neural network, do steps in Illustration 5.1 to take samples.

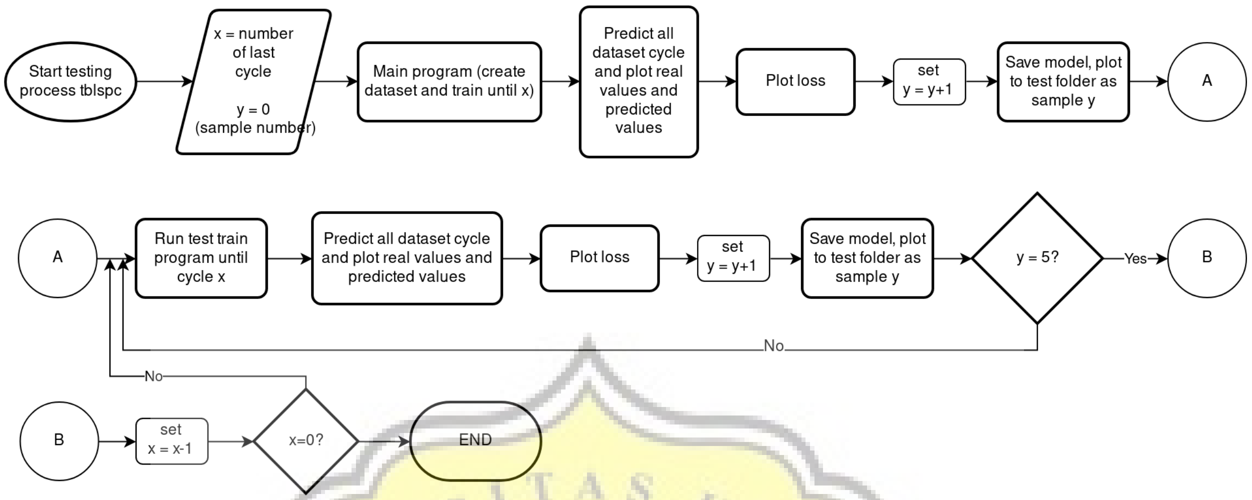
Start testing process tblspc
x = number of last cycle
y = 0 (sample number)
Main program (create dataset and train until x)
Predict all dataset cycle and plot real values and predicted values
Plot loss
set y = y+1
Save model, plot to test folder as sample y
A
A
Run test train program until cycle x
Predict all dataset cycle and plot real values and predicted values
Plot loss
set y = y+1
Save model, plot to test folder as sample y
y = 5?
Yes
B
No
No
B
set x = x-1
x=0?
END

Illustration 5.1: Sampling flowchart

For each training cycle 1...n, it takes sample 5 times. In each sample, it predicts datasets of each cycle and get accuracy percentage (of each predict cycle) with one day tolerance (if the predict is 34.7 rounded to 35 and the real value is 34, it still counted as accurate since the difference is only one day). Each predicted cycle's accuracy inside sample get averaged as average sample accuracy.

Explanation for counting performance can be seen in Illustration 5.2, Illustration 5.3, and Illustration 5.4

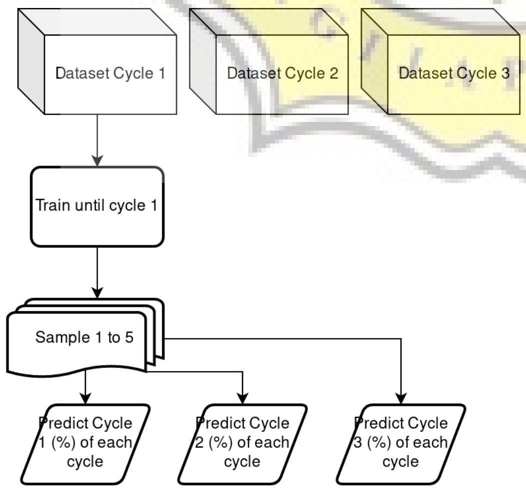
Dataset Cycle 1
Dataset Cycle 2
Dataset Cycle 3
Train until cycle 1
Sample 1 to 5
Predict Cycle 1 (%) of each cycle
Predict Cycle 2 (%) of each cycle
Predict Cycle 3 (%) of each cycle

Illustration 5.2: Train until cycle 1

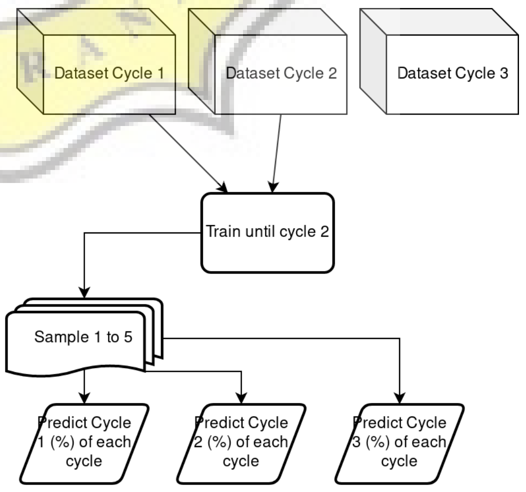
Dataset Cycle 1
Dataset Cycle 2
Dataset Cycle 3
Train until cycle 2
Sample 1 to 5
Predict Cycle 1 (%) of each cycle
Predict Cycle 2 (%) of each cycle
Predict Cycle 3 (%) of each cycle

Illustration 5.3: Train until cycle 2

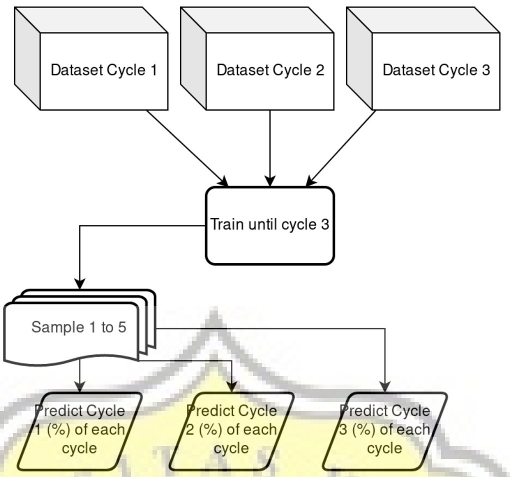
Dataset Cycle 1
Dataset Cycle 2
Dataset Cycle 3
Train until cycle 3
Sample 1 to 5
Predict Cycle 1 (%) of each cycle
Predict Cycle 2 (%) of each cycle
Predict Cycle 3 (%) of each cycle

Illustration 5.4: Train until cycle 3

To select which feature/input is used for training the neural network, this study uses Lasso Model. Because of that, this research only use 3 feature: Cycle number, Free space, and number of data.

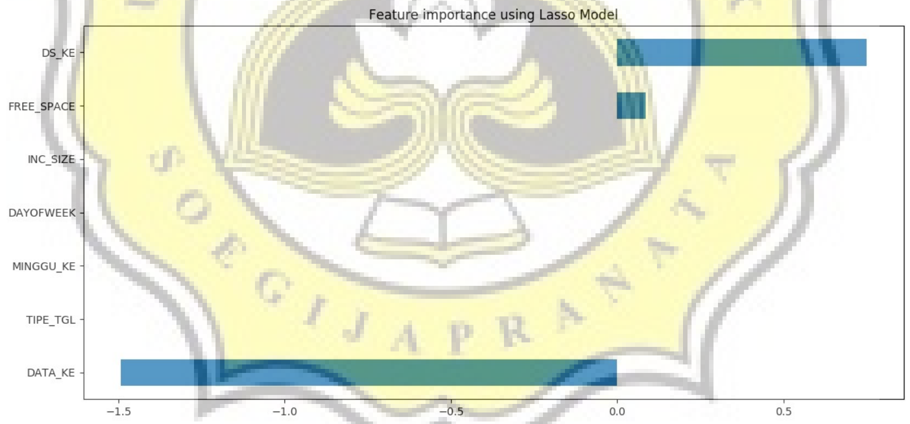

Illustration 5.5: Lasso Model result

Cycle number means the cycle that the dataset cycle belongs in. Number of data means number of row the current data is at. This number increase by 1 every 10 rows of data passed. For example if the system already fetch 16 times before, then the number of data for the current data is 2 since it already passed 10 rows before. If the system already fetch 25 times before, then the number of data for the current data is 3 since it already passed 20 rows before.

```
def _try_file(self):
    ds_list = self.getDsLists()
    tot_acc = 0
    for ds in ds_list:
        test = np.loadtxt("train_nn/datasets/"+ds, skiprows=1,
                          delimiter=',')
        correct = 0
        one_day = 0
        two_day = 0
        three_day = 0
        more = 0
        pred = []
        real = []
        for row in test:
            test_data = row
            predictions = self.model.predict(np.array(
                                            [test_data\
                                                [0:self.input_size]
                                            ]
                                            )
                                        )
            pred_abs = abs(predictions[0][0])
            pred.append(round(pred_abs))
            real.append(test_data[-1])
            print(f"{bcolors.OKGREEN+str(pred_abs)} day(s) until full "\
                    f"(rounded: {round(pred_abs)}), actual:"
                    +str(test_data[-1])+bcolors.ENDC)
            if int(round(pred_abs)) == int(test_data[-1]):
                correct = correct + 1
            else:
                if abs(int(round(pred_abs))- int(test_data[-1])) == 1:
                    one_day = one_day +1
                elif abs(int(round(pred_abs)) - int(test_data[-1])) == 2:
                    two_day=two_day+1
                elif abs(int(round(pred_abs)) - int(test_data[-1])) == 3:
                    three_day=three_day+1
                else:
                    more=more+1
        print(correct,one_day,two_day,three_day, more)
        accuracy=(correct+one_day)/(correct+one_day+two_day+three_day\
                    +more) * 100
        print(f"Accuracy (with 1 day tolerance): {accuracy}")
        print(test[0,1])
        pyplot.title(f"{test[0,1]}-acc:{accuracy}, {str(correct)},"\
                        +str(one_day)+","+str(two_day)+","\
                        +str(three_day)+","+str(more))
        pyplot.plot(pred, label="Prediction")
        pyplot.plot(real, label="Real")
        pyplot.legend(loc='best')
        fname = f"{test[0,1]}-acc:{accuracy}, {str(correct)},"\
                    +str(one_day)+","+str(two_day)+","\
                    +str(three_day)+","+ str(more)+".jpg"
        pyplot.savefig(fname, dpi=300)
        pyplot.clf()
        self.save_acc(ds, accuracy)
        tot_acc=tot_acc+accuracy
    filename='accuracy.json'
```

```
58.    with open(filename) as json_data_file:
59.        data = json.load(json_data_file)
60.    data["total_accuracy"]=tot_acc
61.    data["average_accuracy"]=tot_acc/len(ds_list)
62.    with open(filename,'w') as outfile:
63.        json.dump(data,outfile, indent=2)
64.    self.save_tot_acc(tot_acc/len(ds_list))
```

Code 6

This function loads all the dataset file in dataset directory (line 2 and 5 - 6), do prediction (line 16-21), and then evaluate the accuracy (line 22-41). The accuracy tolerate 1 day difference between actual output and predicted output. Finally, plot the prediction and actual output to graph (line 44-53), then evaluate it and do tweaking.

```
wd='train_nn/loss'
with open(wd+'/loss.csv', "r" ) as readFile:
    reader = csv.reader(readFile)
    loss = list(reader)
with open(wd+'/val_loss.csv', "r" ) as readFile:
    reader = csv.reader(readFile)
    val_loss = list(reader)
loss_mod = []
for row in loss:
    loss_mod.append(float(row[0]))
val_loss_mod = []
for row in val_loss:
    val_loss_mod.append(float(row[0]))
pyplot.title('Loss')
pyplot.plot(loss_mod, label='train')
pyplot.plot(val_loss_mod, label='test')
pyplot.legend()
pyplot.savefig(wd+"/graph_loss.png", dpi=300)
```

Code 7

This function get all the loss (train loss and validating loss), plot it to graph, then save it.

Before continuing to compare the accuracy result, several explanation that need to understand:

1. The Oracle Database store data in form of physical file called datafile.
2. A datafile belong to a tablespace, and a tablespace hold one or more datafile.

3. Tablespace can hold more than one table in different schema (user).
4. To add space in tablespace, database administrator can either add new datafile(s) or resize the current available datafile(s) inside the tablespace.
5. A cycle means the start when new datafile added until the tablespace is full
6. Train until 0...*n*: train the neural network using dataset from the start until *n*
7. Sample : the result of predicting each cycle from all cycle (end up as accuracy with one day tolerance) from trained model. Each train until 0...*n* have 5 samples.
8. Cycle prediction accuracy: percentage of correct prediction compared with real value.
9. Sample average accuracy: each predicted cycle's accuracy inside sample get averaged.
10. Train cycle average accuracy: each sample accuracy in train until 0...*n* get averaged.
11. Tablepace average accuracy: average accuracy of train cycle average accuracy.

After testing several hyper-parameter (number of layer, the width of each layer, optimizer, epoch, and batch size) and inputs, the best settings are:

1. Number of layer and neurons of each layer: the optimal number of the neural network is more than 2 layers with the shape of diamond, this make the information from input can be transformed and broken down to wider neurons (this makes neural network highlight certain neurons which is important information) then narrow it down to infer the output from the pattern. This testing conducted using 3-6-5-3-1 structure from the input layer to output layer.

2. Optimizer: After testing several times, default learning values from the Adam optimizer’s original paper (0,001 learning rate) works the best to make it follow the gradient and not overshoot.
3. Epoch: the epoch doesn’t need more than 50 epoch since we use adam as optimizers. Adam can make the loss lower than 2 in 20 epoch, further than 50 doesn’t improve the performance of the neural network much.
4. Batch size: This study use 10 batch size as the hyper-parameter. More batch size makes it overfit and lower batch size makes it underfit.

In testing the right amount of data training, dataset to train the neural network get divided to 3 types:

1. 20% training set and 80% validation set.
2. 50% training set and 50% validation set.
3. 80% training set and 20% validation set.

The result of each of these split provided in . This result summary is tablespace average accuracy.

Table 5.1: Train cycle average accuracy result

Tablespace	Desc	20% train – 80% validation	50% train - 50% validation	80% train - 20% validation
1	2 tables, 6 cycles	49.39%	52.67%	55.85%
2	1 table, 3 cycles	57.73%	57.80%	58.56%
3	1 table, 3 cycles	53.60%	54.75%	55.40%
4	1 table, 3 cycles	54.93%	55.01%	55.31%
5	1 table, 6 cycles	50.75%	51.10%	52.37%

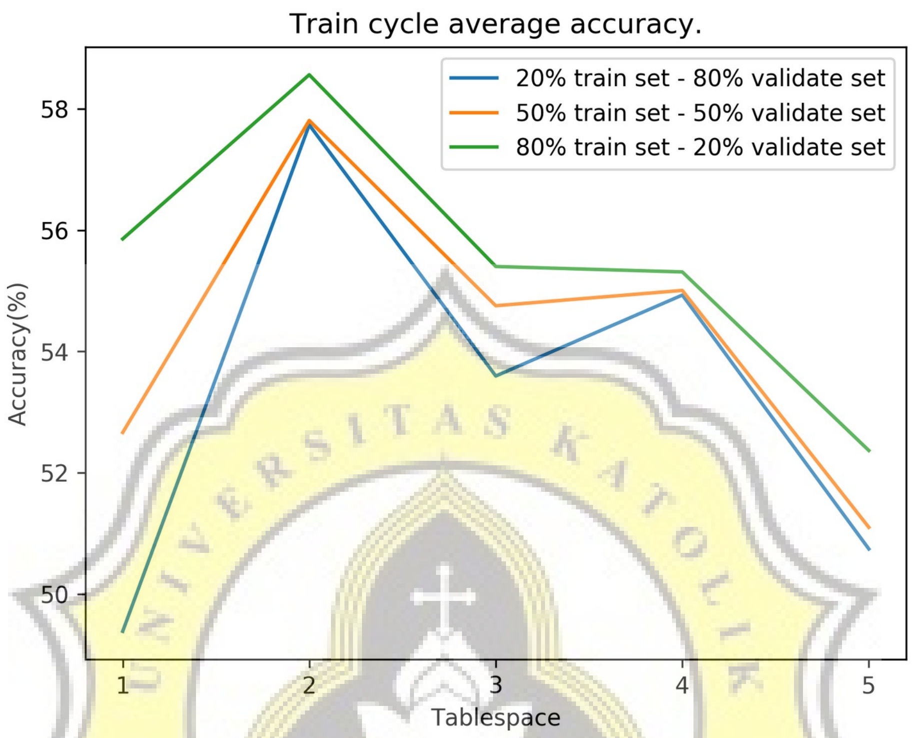

Illustration 5.6: Train cycle average accuracy result in diagram

In detailed result, each best train cycle average accuracy listed below (only taken from 80% training and 20% validation result):

1. Tablespace 1: 63.29% in train until cycle 1
2. Tablespace 2: 65.32% in train until cycle 1
3. Tablespace 3: 64.02% in train until cycle 1
4. Tablespace 4: 63.83% in train until cycle 1
5. Tablespace 5: 60.09% in train until cycle 2 (and train until cycle 1 have average accuracy of 59.19%)

From the testing result, it appears train only cycle 1 generally have high train cycle average accuracy.

For the worst train cycle average accuracy listed below (only taken from 80% training and 20% validation result):

1. Tablespace 1: 46.09% in train until cycle 4
2. Tablespace 2: 50.11% in train until cycle 2
3. Tablespace 3: 50.93% in train until cycle 3
4. Tablespace 4: 41.94% in train until cycle 3
5. Tablespace 5: 36.49% in train until cycle 3

From the result, the train cycle average accuracy can drop to 36.49%. This possibly because the pattern (how it add extent inside datafile or decreasing datafile free space) from one cycle to other is different and the usage of adam as optimizer can't generalize that well. But if the pattern from one cycle and another cycle is similiar, the accuracy goes up significantly. Further research the reason of this is required.