CNN / RNN

bitcoin price up/down

prediction 시도 실패기

— —

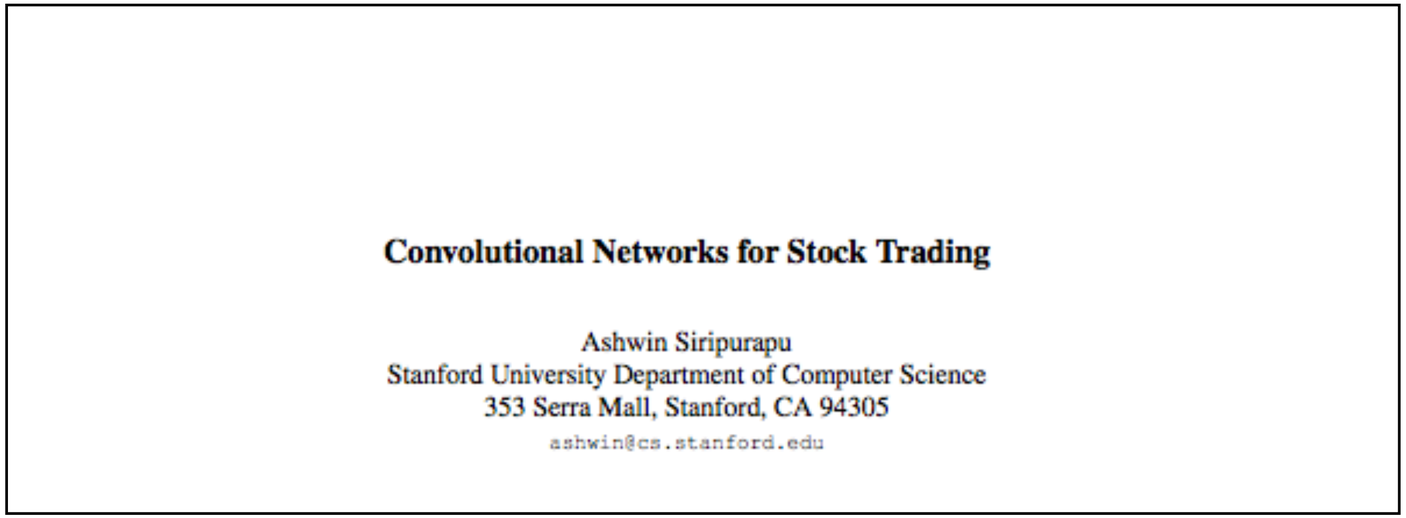
Binary Classification

시계열 문제를 풀 색다른 방법 ...?

넓은 시각, 다양한 시각

CNN 을 선택한 계기

구글링하다 발견한 한 프로젝트 논문



Convolutional Networks for Stock Trading

Ashwin Siripurapu
Stanford University Department of Computer Science
353 Serra Mall, Stanford, CA 94305
ashwin@cs.stanford.edu

논문의 내용

Column Name	Meaning
DATE	Time (which minute of the day)
CLOSE	Closing price (price at the end of the minute)
HIGH	High price (maximum price during the minute)
LOW	Low price (minimum price during the minute)
OPEN	Opening price (price at the beginning of the minute)
VOLUME	How many contracts were offered to be bought/sold in the minute

Table 1. Minute–by–minute data provided by [1]

Concretely, the regression problem is: given as input an image of high and low prices from time $t - 30$ to time t, predict as output the logarithmic return from time t to time $t + 5$, i.e., predict

$$\log\left(\frac{p_{t+5}}{p_t}\right),$$

where p_i denotes the mean of the high price and low price in minute i of the trading day.

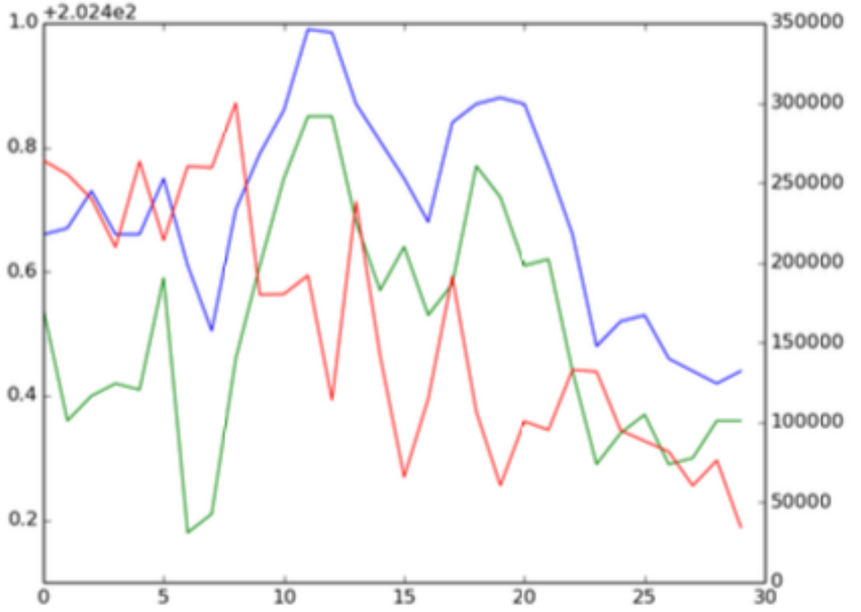


Figure 4. An example image input. As before, high prices in blue, low prices in green. Volume (right axis) in red.

간단 정리

INPUT

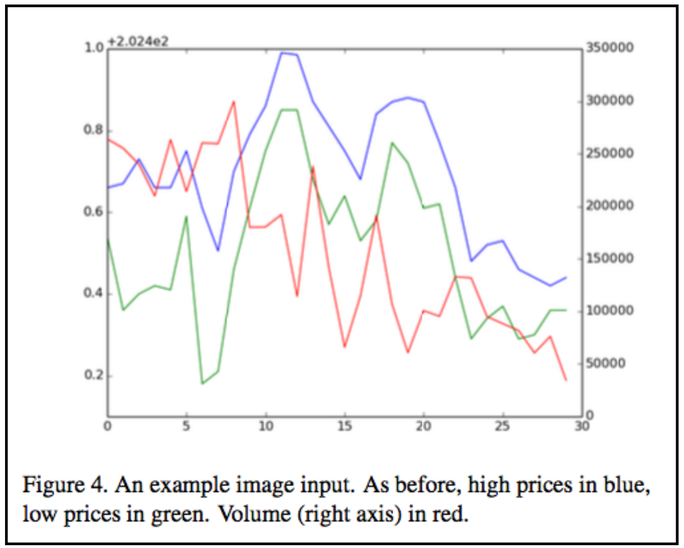


Figure 4. An example image input. As before, high prices in blue, low prices in green. Volume (right axis) in red.

R : Volume
G : High
B : Low

(t - 30) ~ t 까지의 데이터 그래프로 변환

논문에서는 size=32*54 channel=4(RGBA) 이미지 사용

그래프 개형에서 feature 를 잡아내자

CNN

PREDICT

$$\log\left(\frac{p_{t+5}}{p_t}\right),$$

t ~ (t+5) 5분 사이의 return 값

값이 양수로 크면, 투자하면 돈번다
값이 음수이면, 투자하면 손해

논문에서 log return 값 사용

데이터 살피기

bitcoin_ticker.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	date_id	datetime_id	market	rpt_key	last	diff_24h	diff_per_24h	bid	ask	low	high	volume	created_at	updated_at
2	31/05/2017	##########	bitstamp	btc_eur	1996.72	2029.99	-1.6389243	2005.5	2005.56	1950	2063.73	2314.50075	##########	##########
3	31/05/2017	##########	bitflyer	btc_jpy	267098	269649	-0.9460447	267124	267267	267124	267267	70922.8801	##########	##########
4	31/05/2017	##########	korbit	btc_krw	3003500	3140000	-4.3471338	3003500	3004000	3002000	3209500	6109.75287	##########	##########
5	31/05/2017	##########	bitstamp	btc_usd	2237.4	2239.37	-0.0879712	2233.09	2237.4	2154.28	2293.46	13681.282	##########	##########
6	31/05/2017	##########	okcoin	btc_usd	2318.82	2228.7	4.04361287	2319.4	2319.99	2129.78	2318.82	4241.64152	##########	##########
7	31/05/2017	##########	korbit	etc_krw	22740	23150	-1.7710583	22700	22730	21000	25500	855853.37	##########	##########
8	31/05/2017	##########	bitflyer	eth_btc	0.1034	0.08855	16.7701863	0.10315	0.1034	0.10315	0.1034	21670.567	##########	##########
9	31/05/2017	##########	korbit	eth_krw	311800	274500	13.5883424	311800	311950	272500	336000	327416.949	##########	##########
10	31/05/2017	##########	bitflyer	fx_btc_jpy	266600	275066	-3.0778068	266276	266640	266276	266640	70921.7271	##########	##########
11	31/05/2017	##########	okcoin	ltc_usd	25.42	24.931	1.9614135	25.364	25.42	23.2	25.514	171028.35	##########	##########
12	31/05/2017	##########	korbit	etc_krw	22740	23010	-1.1734029	22700	22730	21000	25500	855853.37	##########	##########
13	31/05/2017	##########	korbit	eth_krw	311900	272500	14.4587156	311900	311950	273000	336000	326531.243	##########	##########
14	01/06/2017	##########	bitstamp	btc_eur	2005.56	2013.41	-0.3898858	2005.56	2006.01	1950	2063.73	2317.21965	##########	##########
15	01/06/2017	##########	bitflyer	btc_jpy	268271	269440	-0.4338628	268271	268300	268271	268300	71179.5208	##########	##########
16	01/06/2017	##########	korbit	btc_krw	3003500	3140000	-4.3471338	3003500	3004000	3002000	3209500	6116.84208	##########	##########
17	01/06/2017	##########	bitstamp	btc_usd	2248.39	2242.44	0.26533597	2247.77	2248.38	2154.28	2293.46	13701.6986	##########	##########
18	01/06/2017	##########	okcoin	btc_usd	2320.42	2228.4	4.12942021	2320.99	2321.49	2129.78	2322	4260.26152	##########	##########
19	01/06/2017	##########	bitflyer	eth_btc	0.1034	0.08813	17.3266765	0.10315	0.1034	0.10315	0.1034	21524.4153	##########	##########
20	01/06/2017	##########	bitflyer	fx_btc_jpy	268000	275064	-2.5681296	267858	268000	267858	268000	71179.5208	##########	##########
21	01/06/2017	##########	okcoin	ltc_usd	25.505	25.003	2.00775907	25.437	25.505	23.2	25.514	171285.683	##########	##########
22	01/06/2017	##########	bitstamp	btc_eur	2014.99	2016.7	-0.084792	2014.9	2014.99	1950	2063.73	2330.30475	##########	##########
23	01/06/2017	##########	bitflyer	btc_jpy	268750	269507	-0.2808832	268295	268500	268295	268500	71668.0919	##########	##########
24	01/06/2017	##########	korbit	btc_krw	3017000	3159000	-4.4950934	3017000	3004000	3002000	3209500	6126.34296	##########	##########
25	01/06/2017	##########	bitstamp	btc_usd	2248.35	2238.58	0.43643738	2248.35	2248.69	2154.28	2293.46	13742.1109	##########	##########
26	01/06/2017	##########	okcoin	btc_usd	2323.5	2229.99	4.19329235	2323.5	2323.51	2129.78	2323.5	4263.03152	##########	##########
27	01/06/2017	##########	korbit	etc_krw	22700	23000	-1.3043478	22700	22730	21000	25500	848605.961	##########	##########
28	01/06/2017	##########	bitflyer	eth_btc	0.1034	0.088	17.5	0.10315	0.1034	0.10315	0.1034	21376.2655	##########	##########
29	01/06/2017	##########	korbit	eth_krw	311950	275000	13.4363636	311650	311950	274000	336000	326480.64	##########	##########

(논문에선 high, low, volume 사용)

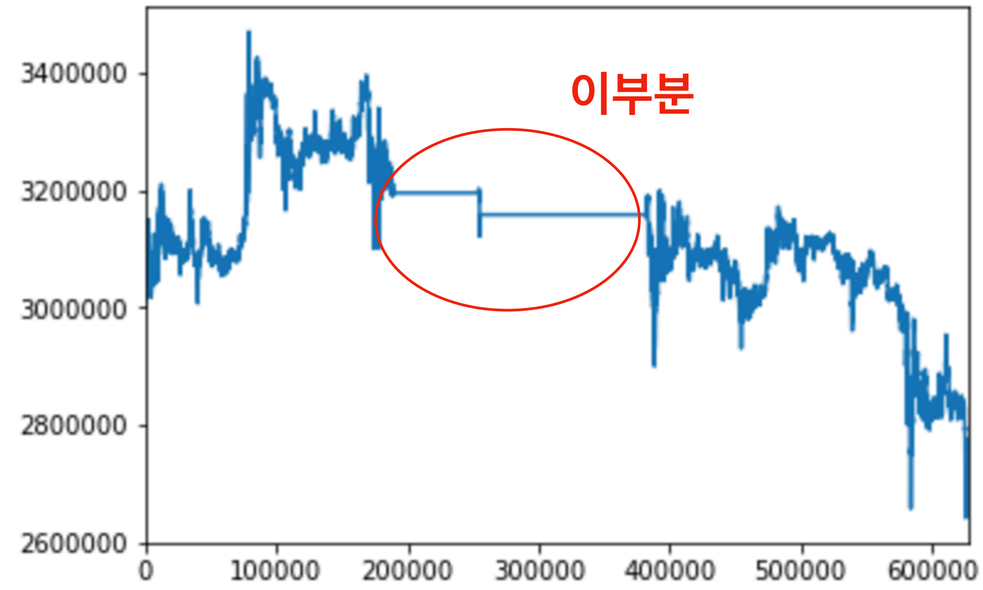
논문과 다르게 high , low 지표가 분단위 측정이 아닌 누적지표이다.

last , volume 지표만 사용하기로 결정

korbit , btc_krw 데이터만 사용

데이터 살피기

```
In [9]:  1 data['last'].plot()
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x11827a320>
```



last 지표 데이터 중반부가 수상하다

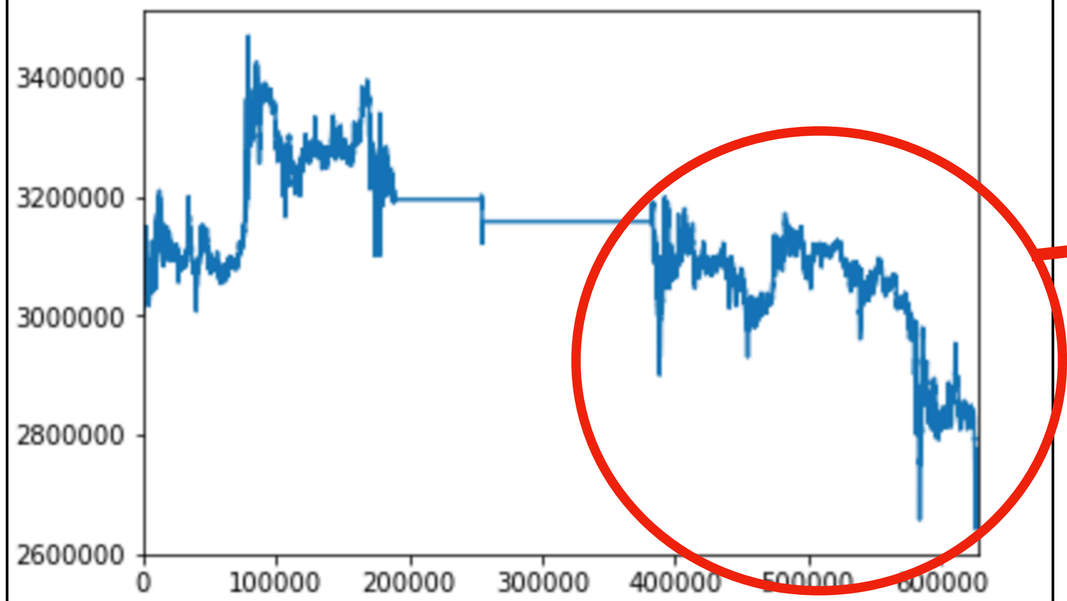
원인을 모르겠으니
일단 39000 index 부터만 사용

(2016년 6월 28일 data부터 사용)

데이터 조정

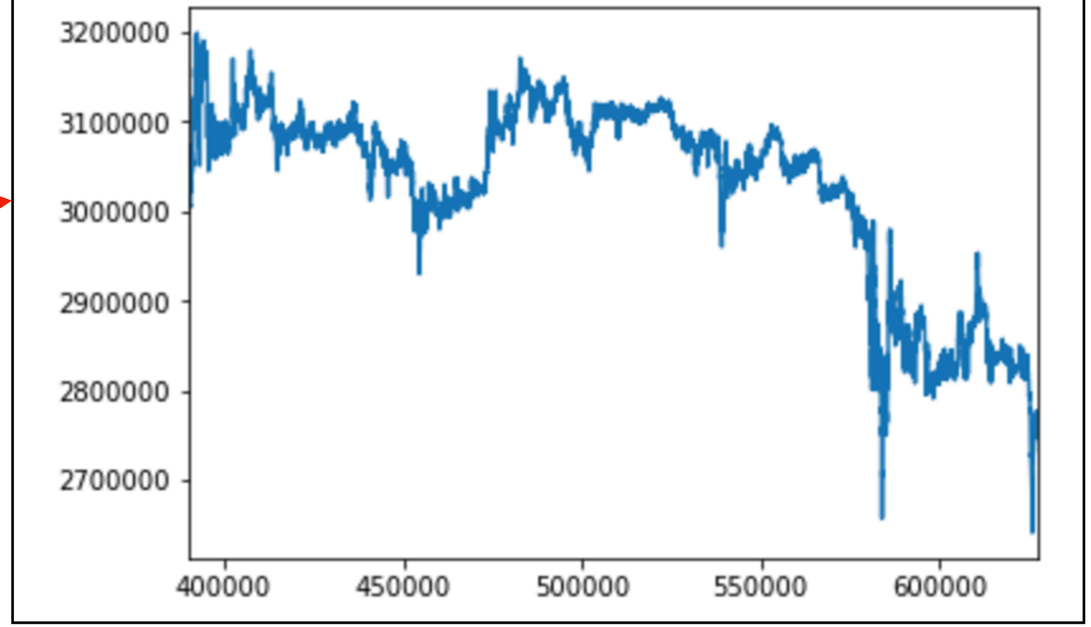
```
1 data['last'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11827a
```




```
1 data['last'][39000:].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11954
```



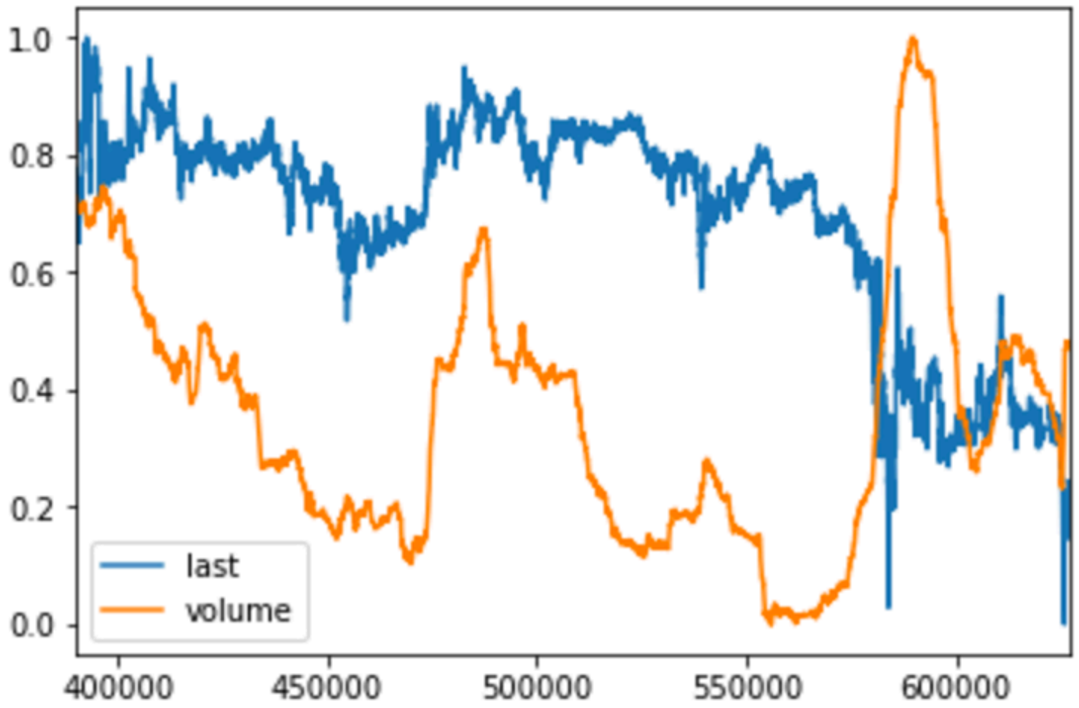
```
data = data[['last', 'volume']]
```

```
def df_norm(df) :
        newdf = (df - df.mean()) /(df.max
        return newdf - newdf.min()
```

```
data = df_norm(data[39000:])

data.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1
```



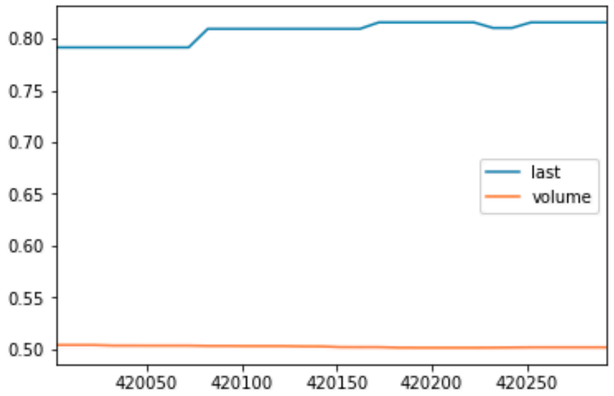
Scaling

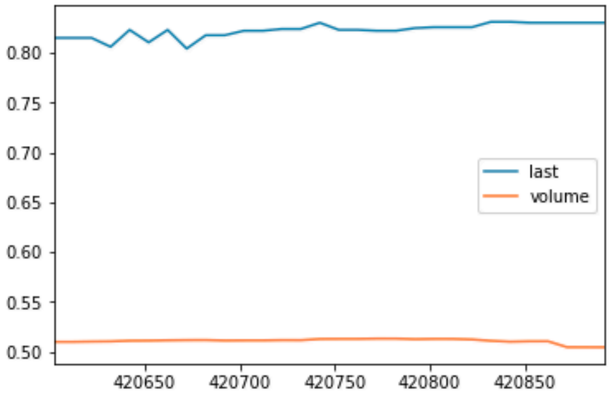
연산량을 줄이고
한 그래프 안에 두 지표를 표현하기 위해

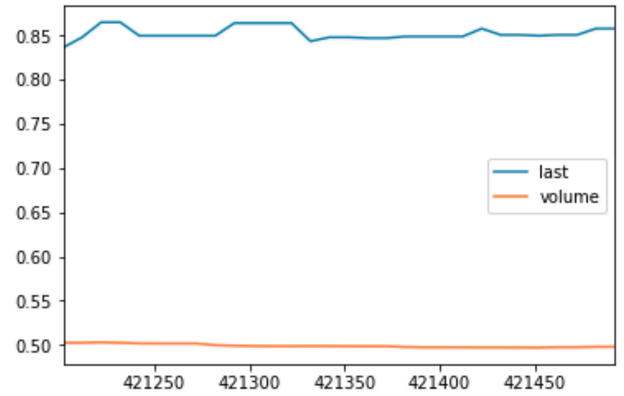
사용할 last 와 volume 값을
max - min = 1
이 되도록
scaling 해줌

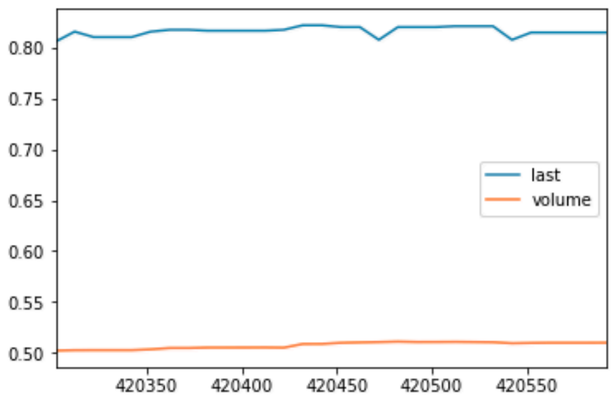
→ 이제 이 그래프를 30분 단위로 슬라이스해서 png로 저장 !!!

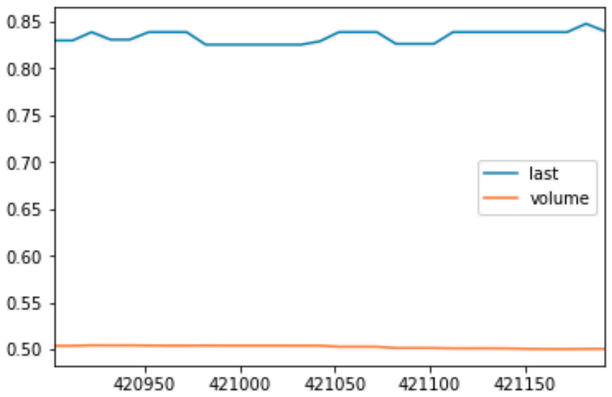
30분 단위로 slice 한 image

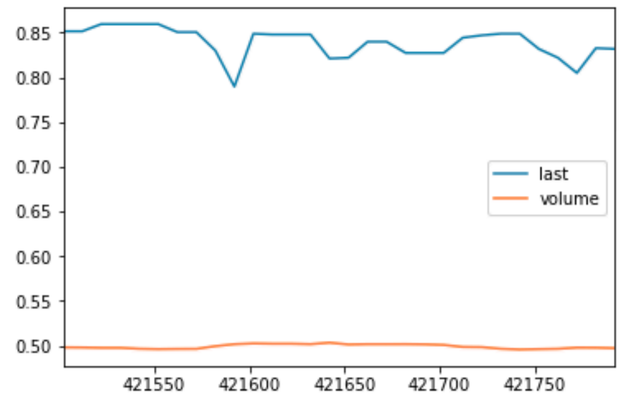






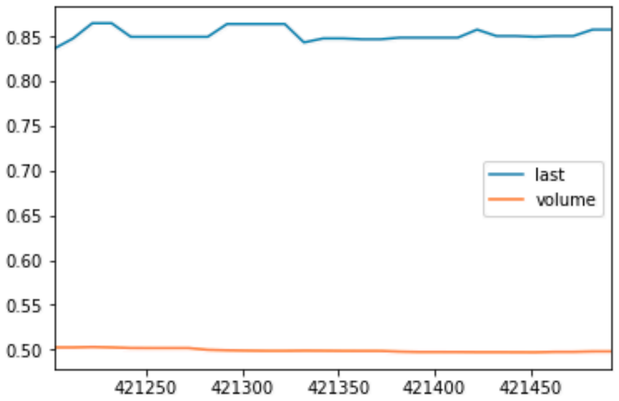


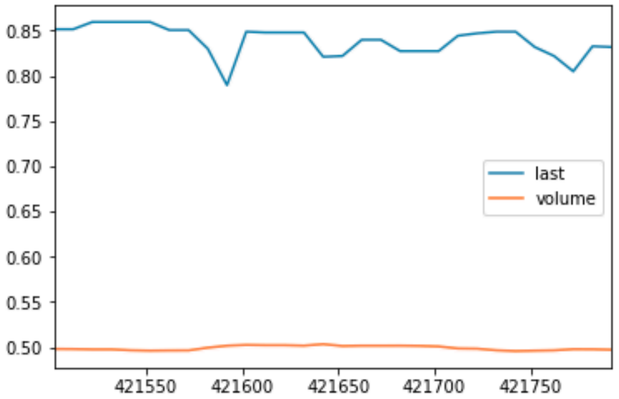




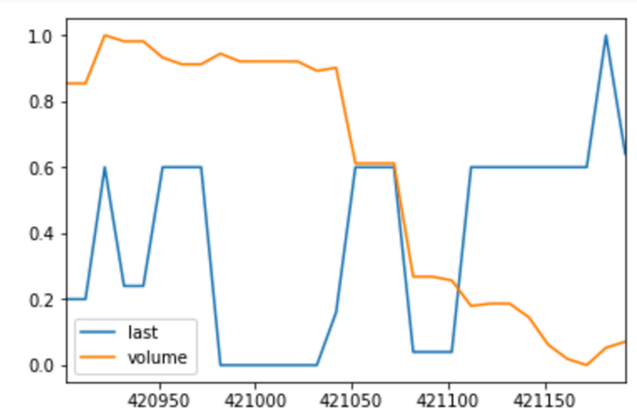
다 비슷비슷해서 Convolution 돌려도 무의미할 듯...

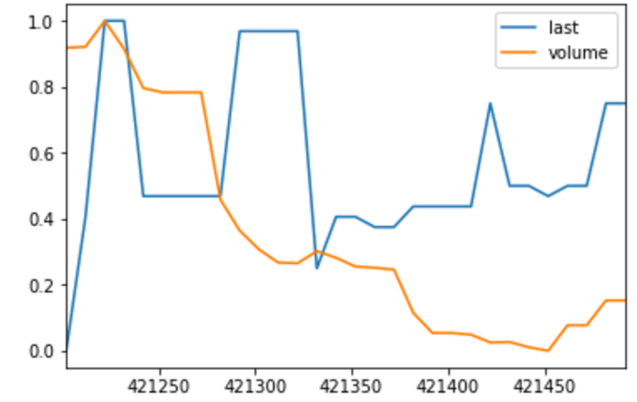
모델 방향을 조정!



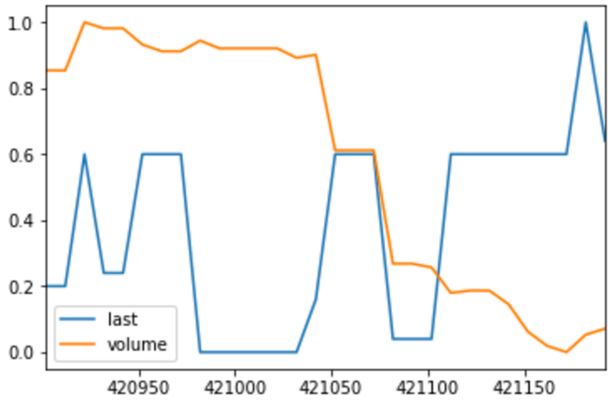


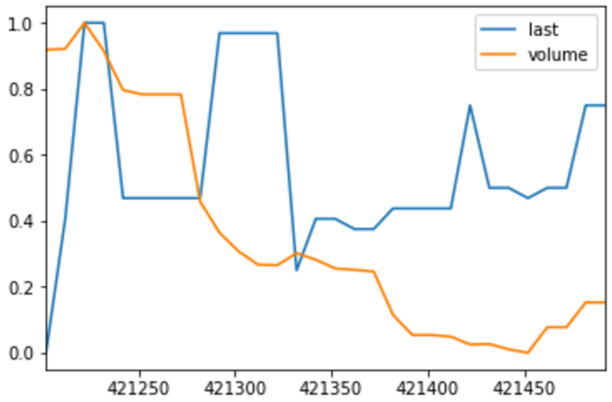
이미지가 너무 단조로우니,
각 30분 단위 window 별로
다시 0,1 scaling 해서
이미지를 극대화!





모델 방향을 조정!

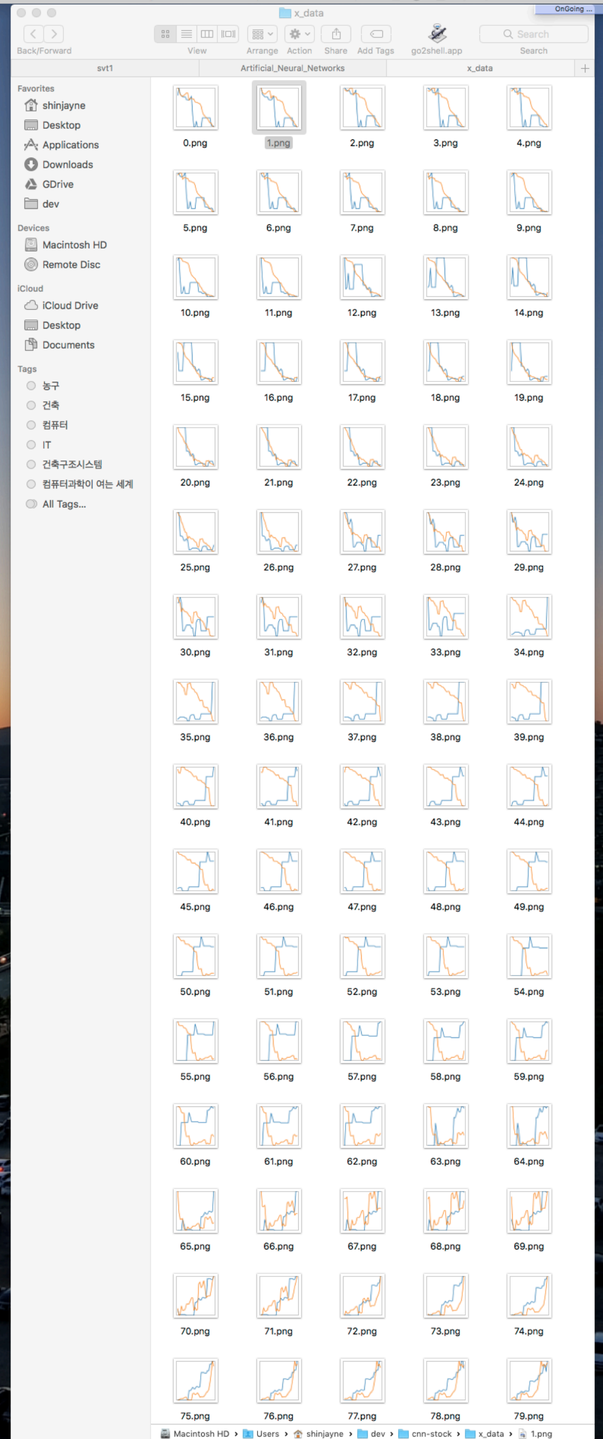




극대화된 이미지를 받으면
(t+5)의 주가가
t 의주가보다
오를것인지 내릴것인지
Binary Classification

y = 1
오른다

y = 0
내린다



신나게 이미지 생성

```
cnn_x.shape: (23683, 50, 50, 4)
cnn_y.shape (23683, 1)
```

```
ctrain_x, ctrain_y :  (18946, 50, 50, 4) (18946, 1)
cvalid_x, cvalid_y :  (4737, 50, 50, 4) (4737, 1)
```

21.png

3958.png

```python
def cnn_graph(learning_rate = 0.001):
    cnn = tf.Graph()
    with cnn.as_default() :

        #feed
        X = tf.placeholder(name = "X", shape = [None, 50, 50 , 4], dtype = np.float32)
        Y = tf.placeholder(name = "Y", shape = [None, 1], dtype = np.float32)

        #TRAINABLE variables
        w1 = tf.get_variable(shape = [3,3,4,32], initializer = tf.random_normal_initializer(stddev = 0.01, mean = 0.0), name = "w1")
        #w2 = tf.get_variable(shape = [3,3,32,64], initializer= tf.random_normal_initializer(stddev = 0.01, mean= 0.0), name = 'w2')

        #w3 = tf.get_variable(shape = [13*13*64 , 256], initializer= tf.random_normal_initializer(stddev = 0.01, mean = 0.0), name = "w3")
        w3 = tf.get_variable(shape = [25*25*32 , 256], initializer= tf.random_normal_initializer(stddev = 0.01, mean = 0.0), name = "w3")
        w4 = tf.get_variable(shape = [256 , 1], initializer= tf.random_normal_initializer(stddev = 0.01, mean = 0.0), name = "w4")

        b3 = tf.get_variable(shape = [256], initializer= tf.random_normal_initializer(stddev = 0.01, mean = 0.0), name = "b3")
        b4 = tf.get_variable(shape = [1], initializer= tf.random_normal_initializer(stddev = 0.01, mean = 0.0), name = "b4")

        #Operations

        #<conv1>
        conv1_c = tf.nn.conv2d(X, w1 , strides = [1,1,1,1] , padding = 'SAME')
        #conv1_c shape : [None, 50,50,32]
        conv1_r = tf.nn.relu(conv1_c)
        conv1_p = tf.nn.max_pool(conv1_r, ksize = [1,2,2,1] , strides = [1,2,2,1], padding = 'SAME')
        #conv1_p shape : [None, 25,25, 32]

        ##<conv2>
        #conv2_c = tf.nn.conv2d(conv1_p , w2, strides = [1,1,1,1], padding = 'SAME')
        ##conv2_c shape : [None, 25, 25, 64]
        #conv2_r = tf.nn.relu(conv2_c)
        #conv2_p = tf.nn.max_pool(conv2_r, ksize = [1,2,2,1], strides = [1,2,2,1], padding = 'SAME')
        ##conv2_p shape : [None, 13, 13 ,64]

        #<flattening>
        #flattened = tf.reshape(conv2_p , [-1, 13*13*64])
        flattened = tf.reshape(conv1_p , [-1, 25*25*32])

        #<2 layer fully connected NN>
        fc_1 = tf.nn.relu(tf.matmul(flattened , w3) + b3)
        fc_2 = tf.nn.sigmoid(tf.matmul(fc_1, w4) + b4)


        #Loss function : square mean
        loss = tf.reduce_mean(tf.square(fc_2 - Y ))

        #Optimizer : Adam Optimizer
        optimizer = tf.train.AdamOptimizer(learning_rate)
        train = optimizer.minimize(loss)

        #Accuracy
        fc_2 = fc_2 > 0.5
```

모델 graph

<configure>

pooling 없이 conv 2번해서 blurring 으로 인한 손해 예방

첫번째 convolution → 첫번째 relu — 5,5 filter — 64개

두번째 convolution → 두번째 relu — 7,7 filter — 32개

↓

max_pooling

↓

flattening

↓

2 레이어 fully-connected NN

↓

sigmoid

↓

square_mean loss

```python
def rnn_graph(learning_rate = 0.001, n_hidden = 128) :
    rnn = tf.Graph()
    with rnn.as_default() :
        #feed
        X = tf.placeholder(tf.float32, [None, 30, 2], name = 'X')
        Y = tf.placeholder(tf.float32, [None, 1], name = 'Y')

        #TRAINABLE variables
        cell = tf.nn.rnn_cell.BasicRNNCell(n_hidden)
        w = tf.get_variable(name = 'w', shape = [n_hidden, 1] ,initializer = tf.random_normal_initializer(stddev=0.01, mean=0.0))
        b = tf.get_variable(name = 'b', shape = [1] ,initializer = tf.random_normal_initializer(stddev=0.01, mean=0.0))

        #Operations

        ##<RNN>
        outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)
        # outputs shape : (None, n_step, n_hidden)

        outputs = tf.transpose(outputs, [1, 0, 2])
        # outputs shape : (n_step, None, n_hidden)
        outputs = outputs[-1]  # 마지막 셀에서 나온 결과값만 사용합니다.
        #shape : (None, n_hidden)

        ##<basic NN>
        fc = tf.nn.sigmoid(tf.matmul(outputs, w) + b)

        #Loss function
        loss = tf.reduce_sum(tf.square(fc - Y))

        #Optimizer : Adam Optimizer
        optimizer = tf.train.AdamOptimizer(learning_rate)
        train = optimizer.minimize(loss)

        #Accuracy
        fc = fc > 0.5
        fc = tf.to_float(fc)
        accuracy = tf.reduce_mean(tf.to_float(tf.equal(fc, Y)), name= 'accuracy')

    tf.reset_default_graph()
    op_list = [X, Y, loss, train, accuracy]

    return rnn , op_list
```

간단한 RNN 도 함께

<configure>

basic RNN cell
(hidden = 128)
|
1 레이어
NN
|
sigmoid
|
square_mean loss

run.py <- utils.py <- models.py

```
Shinui-MacBook-Pro:cnn-stock shinjayne$ python3 run.py --model="rnn" --name="rnn3" --total_epoches=5
###################################
############RNN RUNNING############
###################################
rnn_x.shape: (23683, 30, 2) rnn_y.shape (23683, 1)
rtrain_x, rtrain_y :  (18946, 30, 2) (18946, 1) rvalid_x, rvalid_y :  (4737, 30, 2) (4737, 1)

Training Start : epoch =  5 , each epoch has  189  batch.

Training Complete!
Model`s Variables are saved at :  ./ckpt/rnn3.ckpt
=> Validation Set Accuracy is  56.9769918919 %.
```

```
Shinui-MacBook-Pro:cnn-stock shinjayne$ python3 run.py --model="cnn" --name="cnn4" --total_epoches=3
###################################
############CNN RUNNING############
###################################
<Start Transfer> : shape =  (50, 50, 4)
<Complete image to Matrix transfer>  : Total  23683  matrix into list => shape =  (50, 50, 4)
cnn_x.shape: (23683, 50, 50, 4) cnn_y.shape (23683, 1)
ctrain_x, ctrain_y :  (18946, 50, 50, 4) (18946, 1) cvalid_x, cvalid_y :  (4737, 50, 50, 4) (4737, 1)

Training Start : epoch =  3 , each epoch has  189  batch.

Training Complete!
Model`s Variables are saved at :  ./ckpt/cnn4.ckpt
=> Validation Set Accuracy is  56.5336704254 %.
```

Tune

1. batch_size
2. total_epoches
3. learning_rate
4. optimizer
5. hidden_dim
6. learning_rate_decay
7. model structure

...

```python
if __name__ == "__main__":

    #Settings
    flags.DEFINE_string("name", "unnamed", "model name for ckpt file")
    flags.DEFINE_string("csvpath", "./bitcoin_ticker.csv", "model directory")
    flags.DEFINE_string("imgdir", "./x_data", "image directory")
    flags.DEFINE_boolean("gen_png", False, "generate new png files or not. extreamly lot of time consumed")

    #hyperparameters
    flags.DEFINE_integer("batch_size", 100, "batch size")
    flags.DEFINE_integer("total_epoches" , 15 , "total_epoches")
    flags.DEFINE_integer("learning_rate" , 0.001 , "learning rate")


    flags.DEFINE_string("model", "cnn", "cnn, rnn")
    #flags.DEFINE_string("mode", "train", "train, valid")

    flags.DEFINE_string("ckptfile", None, "check point file path")

    if FLAGS.model == "cnn":
        run_cnn(name = FLAGS.name, csvpath = FLAGS.csvpath , learning_rate = FLAGS.learning_rate, batch_size = FLAGS.batch_size, total_epoches = FLAGS.total_epoches, ckptfile = FLAGS.ckptfile, img
    elif FLAGS.model == "rnn":
        run_rnn(name = FLAGS.name, csvpath = FLAGS.csvpath , learning_rate = FLAGS.learning_rate, batch_size = FLAGS.batch_size, total_epoches = FLAGS.total_epoches, ckptfile = FLAGS.ckptfile)
```