Hi Mathew,

I generated artificial triangle training data using two function generators. The first created a hard upper and lower boundary that converges linearly over time, centered on a mean. Some of these can be seen in "CryptoCNN/triangles". They are very geometrically triangular– which is exactly what I wanted. Unfortunately, this doesn't accurately represent real candlestick data.

For this reason, I made a second triangle generation function. This one produced more realistic looking candlestick charts. I passed in parameters for mean and sd (a factor like volatility), and generated random movement centered around the mean that dampens over time.

I ended up finding that training with the second function yielded better results, so I ended up separating these triangles and putting them in their own directory – CryptoCNN/varplots

These functions generate rightward pointing triangles – to obtain leftward triangles, we can just rotate the images by 180 degrees (as I mentioned in ktrain.py).

To generate the adversarial example, I simulated random price movements using the function randomGen, then wrote these images to CryptoCNN/noise.

After generating the artificial triangle and random motion data, I used the first tutorial you linked to learn how to write a CNN and kept using their parameters. I found that my computer is too slow to train with all 40~ thousand images I generated, so I just fed in a few thousand triangles and noise charts, then applied to model to classifying the actual, sliced up crypto price images. I then saved these triangles to "CryptoCNN/foundtriangles". I designed this process to be accumulative, so running the neural network repeatedly will gradually save more and more triangles in foundtriangles.

While I approached this problem with binary classification on sliced up images, I would have preferred to use object detection on larger sections of price history to more effectively identify triangles of a wider variety of size. Unfortunately, I was previously unfamiliar with computer vision so I was not able to learn object detection in this timeframe.

Interestingly, I found that optimizing the neural network's hyperparameters wasn't too important – I quickly reach 98+% accuracy on the binary classification problem with any hyperparameter, layer, and data configuration. It seems that this problem is most significantly gated by how representative the artificially generated data is of the pattern of interest.

Nicholas Chen

Included Files:

CryptoCNN/triangle/ - Folder with all artificially generated triangles

CryptoCNN/actualplots/ - Folder with sliced up actual price data for cryptocurrencies

CryptoCNN/varplots/ - Folder with triangles generated with decreasing variance (second) function

CryptoCNN/noise/ - Folder with random movement candlesticks

CryptoCNN/pricedatabase/ - Folder with Crypto price histories

CryptoCNN/foundtriangles/ - **Folder with found triangles**

Advplot.py – python script to generate triangles and random movement charts

Slices.py – script to slice up actual crypto price data and make charts

Ktrain.py – CNN model training and triangle finder

Instructions – All of the python scripts work independently if you want to test them (though the paths might be off because I worked on a Mac). To train the CNN and find new triangles, just run python ktrain.py from the appropriate directory and edit hyperparameters, amount of training data, etc. as you wish.