Fast Candlestick Patterns Detection with Limited Training Samples Using RGB Gramian Angular Field and YOLO-LITE-V1

Heyang Huang
Stanford University
726 Serra Street, CA, 94305
heyangh@stanford.edu

Abstract

This report offers an automatic pipeline for training effective candlestick pattern detectors with only limited labeled training set. The only input data our pipeline needs is a time series data of asset prices, along with thousands or even hundreds labelled samples of candlestick patterns in the form of "T5-T9 is Morning Star pattern." The pipeline consists of four modules: native pattern classification, ARIMA data augmentation, auto-labeling, and Yolo model training. The RGB-GAF pattern classifier module first converts raw time series data into RGB version of Gramian Angular Field (An enhanced GAF algorithm designed by me), then trains a simple CNN classifier on the Granmian Angular field images. The ARIMA data augmentation module then fits an ARIMA model on original time series and extrapolates the time series many times longer. The exact length depends on the length of original sequence. Then, the auto-labeling module uses a moving window to classify whether a segment of the extrapolated time series forms a certain candlestick pattern. In this way, the module augments the training set many times of its original size. Last, the Yolo module pretrains and fits a Yolo-Lite module on the augmented dataset. The fitted model yields a mAP > 0.4 of 0.481. The result is much better than it appears, since the testing set labels are created by human beings and are subjective. When we manually examine each mis-classified cases, we in fact find most of them quite reasonable.

1. Introduction

Candlestick chart was first developed in Japan and then became popular around the world. It is a visualizable tokenization of stock prices

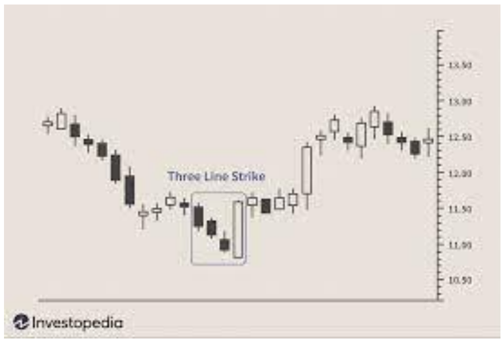


Figure 1: Example of candlestick charts

One candlestick contains the open, close, highest and lowest price information in that timestep. As shown in figure 2 below,

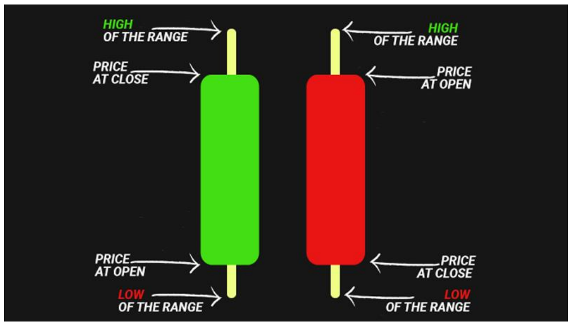


Figure 2: Information contained in a candlestick

As mentioned by Tharavanij [1], "it is very common for investors to use them in conjunction with other technical indicators. In fact, according to a survey by Menkhoff (2010), fund managers apply it in their shorter-term forecasts. Candlestick charting is unique in the sense that it concurrently plots daily open, high, low, and close price movements Morris [2]. As such, it reveals demand and supply changing balance and also investor sentiment and psychology. Proponents of candlestick believe that investors could use these chart patterns to predict short-term price movements or future turning points."

Since candlestick patterns are most useful for high-frequency trading, it's natural to think about applying deep learning algorithms to automate the candlestick pattern detection. There are three biggest challenge: lack of training set, high-quality labeling, and a universal quantifiable definition of patterns. The real-world financial data is limited. Moreover, many candlestick patterns are defined in a way that relies on people's discretion. How many consecutive bars can be seen as a trend? Therefore, the manual labeling process can be time-consuming and error prone.

My proposed pipeline can effectively augment enough number of stock prices in the form as time series thanks to the RGB-GAF transformation module which will be later discussed in methodology section. We will then have enough labeled samples to train a YOLO-LITE model to directly locate and classify candlestick patterns on chart pictures. The ability to classify directly on chart photo rather than numeric time series form is very important. It mimics how real traders think and grant more adaptability to classify candlestick charts in various timeframes without major change of code structure. To summarize, my proposed pipeline tackles the following three biggest problems in deep learning-based candlestick detection:

1. Not enough training samples available
2. Needs for manual labeling
3. Based on numerical values rather than picture, so require complete rework when the timeframe or price range changes.

2. Related Work

2.1. Related work in Gramian Angular Field Encoding

Given a segment of time series, how do we test whether it fits into a candlestick pattern? While traditional machine learning and RNN model fails to do a good job, Chen [3] proposes that we can use Gramian Angular Field to transform the time series data into 2 dimensional pictures and run CNN classifier on it. Chen claims that the CNN classifier achieves a n accuracy of 92.5% on simulated data vs 87.2 for traditional machine learning models. Wang [4] also supports the effectiveness of Gramian Angular encoding on time series classification problems:

"We used Tiled Convolutional Neural Networks (tiled CNNs) on 20 standard datasets to learn high-level features from the individual and compound GASF-GADF-MTF images. Our approaches achieve highly competitive results when compared to nine of the current best time series classification approaches. Inspired by the bijection property of GASF on 0/1 rescaled data, we train Denoised Auto-encoders (DA) on the GASF images of four standard and one synthesized compound dataset. The imputation MSE on test data is reduced by 12.18% – 48.02%"

In their work, they also compare the Gramian Angular Summation/ Difference model with Markov Transition field encoder, and claims that the Gramian Anguar-based encoder outperforms Markov transition field encoder in 16 out of 20 time series benchmarks they selected.

I further work on their approach. The weakness of their method is mentioned in Xiang [5]:

"Let us take a normalized time series X and use the GAF method to imagine it. Despite the presence of both sine and cosine functions in the initial work, now themajority of examples I reviewed use only the cosine function to get the GAF image. Note that for any θ in [0,2π), we have also (2π-θ) in [0,2π). Recall that the cosine function in [0,2π) is symmetric with respect to θ=2π so that cos(θ) equals cos(2π-θ). Back to the GAF matrix, we see that cos(2π-θi-θj)=cos(θi+θj) and therefore π-θi and π-θj give us the same value as the i-j element of the GAF matrix. For all θ in [0,π), we know that π-θ=arccos(-x) and we can therefore get the fact that the time series X can give us the same matrix. To resume, if we reverse the sign of every point on a time series, the transformation of GAF results in the same image."

My solution to this issue is simple: I stack the sine and cosine GAF as well as GAD graph together and get a RGB rather than gray scale image. This RGB scales can show the downward or upward trend with different colors.

2.2. Related work in fast Object Detection Model and Pretraining on Abstract photos.

What object detection model works best for fast object detection? The financial markets require every detection model to be fast, since some of the candlestick algorithms work in min or even second timeframe. As Joseph [6] mentioned in YOLO (You Only Look Once), base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork. YOLO is both fast in recognition and pre-train. Moreover, it can achieve good result on non-natural photos.

YOLO_LITE, designed by Huang [7], which focuses on fast classification "runs at about 21 FPS on a non-GPU computer and 10 FPS after implemented onto a website with only 7 layers and 482 million FLOPS. This speed is 3.8 × faster than the fastest state of art model, SSD MobilenetvI." YOLO-LITE doesn't add prune methods or special designed convolution layer to boost performance, instead it focuses on the optimization tricks available that gears towards fast classification.

Similarly, Zhang [8], Landola [9], Howard [10] also provides solutions to the speed up the pretraining and detection phase on YOLO. Their work finds ways to design original convolution layer and pruning methods to cut down number of parameters in network.

So I built upon Zhang [8]'s shufflenet and incorporate the optimization module used in Huang[7] to formulate the updated version of YOLO-LITE-V1. However it doesn't seem to outperform the vanilla YOLO-LITE model, maybe due to some inefficient implementation I made. So I decide to use YOLO-LITE as my object detection model.

3. Methods

The pipeline consists of four modules: GAF pattern classification, ARIMA data augmentation, auto-labeling, and Yolo model training, as shown below in figure 3. The GAF pattern classifier module first converts candlesticks into an enhanced version of Gramian Angular Field, then trains a simple CNN classifier on the Granmian Angular field images. The ARIMA data augmentation module then fits an ARIMA model on original time series and extrapolates the time series many times longer, given the length of original sequence. Then, the auto-labeling module uses a moving window to classify whether a segment of the extrapolated time series forms a certain candlestick pattern. In this way, the module augments the training set many times of its original size. Last, the Yolo module fits a Yolo-Lite module on the augmented dataset.

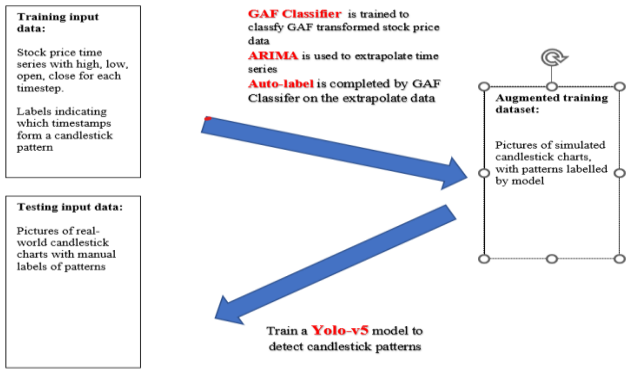

Training input data:
Stock price time series with high, low, open, close for each timestep.
Labels indicating which timestamps form a candlestick pattern
Testing input data:
Pictures of real-world candlestick charts with manual labels of patterns
GAF Classifier is trained to classfy GAF transformed stock price data
ARIMA is used to extrapolate time series
Auto-label is completed by GAF Classifer on the extrapolate data
Augmented training dataset:
Pictures of simulated candlestick charts, with patterns labelled by model
Train a Yolo-v5 model to detect candlestick patterns

Figure 3: Model Pipeline

3.1. RGB-GAF Pattern classification

As mentioned in Chen [3], the Gramian Angular Field Algorithm first converts time series to polar representations.

$$\phi = \arccos(\tilde{x}_i), -1 \leq \tilde{x}_i \leq 1, \tilde{x}_i \in \tilde{X}$$

$$r = \frac{t_i}{N}, t_i \in \mathbb{N}$$

Then, algorithm sums up degrees to calculate the cosine function(Gramian Angular Summation Field).

$$\text{GASF} = \cos(\phi_i + \phi_j) = \tilde{X}^T \cdot \tilde{X} - \sqrt{I - \tilde{X}^2}^{\,\prime} \cdot \sqrt{I - \tilde{X}^2}$$

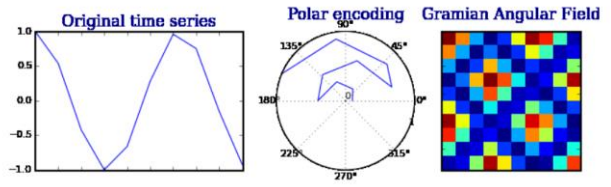


Figure 4: Vanilla Gramian Angular Field Conversion

Following the step above, I reproduce the following GASF representation as mentioned by Chen [3] below in Figure 5 and 6. If we compare them, we can find the angular field representation does capture the fundamental difference in the two patterns and show it in the form of color gradient shifts.

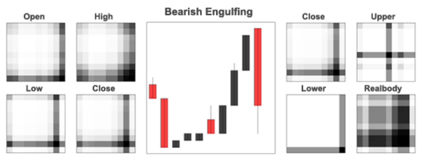


Figure 5 Vanilla Representaion of Bearish Engulfing

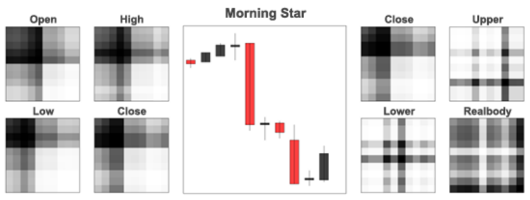


Figure 6: Vanilla Representation of Morning Star

However, the weakness of this method is mentioned in Xiang [5]:
"Note that for any θ in [0,2π), we have also (2π-θ) in [0,2π). Recall that the cosine function in [0,2π) is symmetric with respect to θ=2π so that cos(θ) equals cos(2π-θ). Back to the GAF matrix, we see that cos(2π-θi-θj)=cos(θi+θj) and therefore π-θi and π-θj give us the same value as the i - j elements of the GAF matrix. For all θ in [0,π), we know that π-θ=arccos(-x) and we can therefore get the fact that the time series X can give us the same matrix. To resume, if we reverse the sign of every point on a time series, the transformation of GAF results in the same image."

To resolve this issue, I propose the RGB-GAF algorithm: stacking the sine version, or Gramian Angular Difference Field, and cosine GASF graphs together and get a RGB rather than gray scale image. This RGB scales can show the downward or upward trend with different colors.

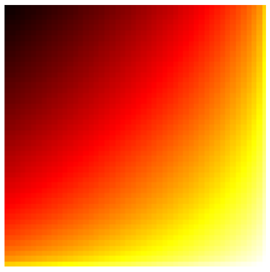


Figure 7: RGB GAF Representation of Morning Star

As shown in Figure 7, we can now differentiate the downward trend from the upward trend of same magnitude by the red/green color gradient.

Then, we apply the following simple CNN models to classify the candlestick patterns as shown in Figure 8. The network structure is inspired by Chen [3] and selected using K-fold validation. Noticed than the input shape is not limited to (10,10,3), for different moving window size, we use different input shape and corresponding input layer shape. We use those CNNS to classify the RGB-GAF with the respective resolution

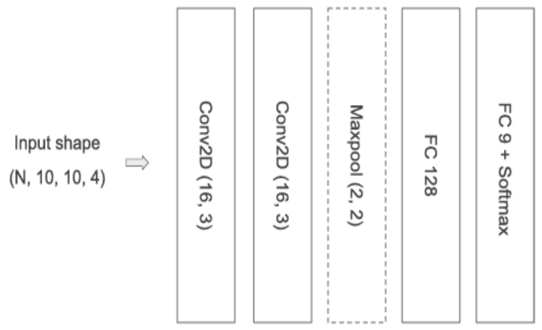
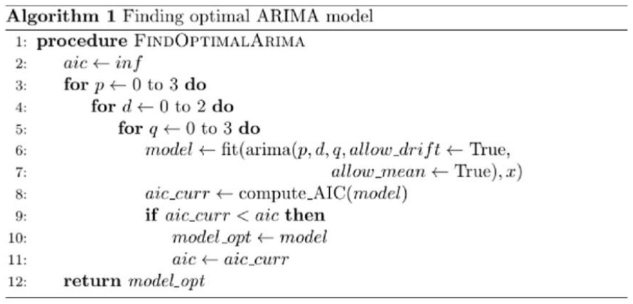


Figure 8: Simple CNN Used for RGB-GAF Classification of size 10*10*3

3.2. ARIMA data augmentation

For the time series, we apply the ARIMA algorithm as mentioned below in Figure 10



Algorithm 1 Finding optimal ARIMA model

```
procedure FINDOPTIMALARIMA
    $aic \leftarrow inf$
    for $p \leftarrow 0$ to 3 do
        for $d \leftarrow 0$ to 2 do
            for $q \leftarrow 0$ to 3 do
                $model \leftarrow$ fit(arima($p, d, q, allow\_drift \leftarrow$ True,
                                            $allow\_mean \leftarrow$ True), $x$)
                $aic\_curr \leftarrow$ compute_AIC($model$)
                if $aic\_curr < aic$ then
                    $model\_opt \leftarrow model$
                    $aic \leftarrow aic\_curr$
    return $model\_opt$
```

Figure 9: ARIMA

After we fit the ARIMA model on the original low price series. We train another three ARIMA models on the differences between low and open, close, high respectively.

We then extrapolate on all of those four ARIMA series and then add the three-difference series with the low price series to derive the augmented time series. This way of implementation is used to ensure high price will always be the high price in each timestamp.

3.3. Auto-labeling

We then transform the augmented data into RGB-GAF and run the fitted simple CNN , which is shown in Figure 9, to predict the label on augmented data, and thereby creates augmented data.

3.4. YOLO-LITE Detection

A summary of YOLO-LITE algorithm:

For each bounding box that the algorithm loops over and propose, we calculate the confidence for that bounding box and each object class, or the probability of an object exists in a bounding box:

$$C = Pr\left(Object\right) * IOU_{pred}^{truth} \tag{1}$$

Then we transform C into class-specific conditional probability:

$$\begin{aligned} &Pr\left(Class_i|Object\right) * Pr\left(Object\right) * IOU_{pred}^{truth} \\ =&Pr\left(Class_i\right) * IOU_{pred}^{truth}. \end{aligned} \tag{2}$$

YOLO-LITE defines loss as

$$\begin{aligned} Loss = & \\ &\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{A} \mathbb{1}_{ij}^{obj} [(b_{x_i} - b_{\hat{x}_i})^2 + (b_{y_i} - b_{\hat{y}_i})^2] \\ &+ \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{A} \mathbb{1}_{ij}^{obj} [(\sqrt{b_{w_i}} - \sqrt{b_{\hat{w}_i}})^2 + (\sqrt{b_{h_i}} - \sqrt{b_{\hat{h}_i}})^2] \\ &+ \sum_{i=0}^{s^2} \sum_{j=0}^{A} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\ &+ \lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{A} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\ &+ \sum_{i=0}^{s^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2. \end{aligned} \tag{3}$$

While the loss function is used to find the sweet spot of center of bounding box, the mAP (for our case >0.4 is used) is used to measure the prediction accuracy which defined as:

$$avgPrecision = \sum_{k=1}^{n} P(k)\Delta r(k). \tag{4}$$

The most difficult part for YOLO-LITE implementation is pretraining. I additionally generated 3500 segments of the augmented data auto-labeled in previous module to candlestick charts and change their background to the same texture as in Yahoo-finance. Notice that those pictures contain different number of candlestick charts but has the same resolution 64*64*3. I then train the model listed in the YOLO-LITE open-source repo with those 3500 labeled photos.

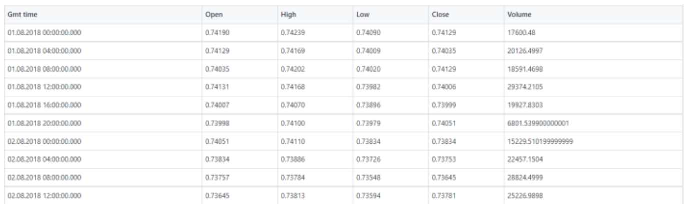
4. Dataset and features

As shown in figure 3, there are 2 datasets that we derive from online source: training input data and testing data. The augmented dataset is generated by ARIMA Simulator and labeled by trained CNN model.

4.1. Training Dataset

The training dataset is used to train a simple CNN model to label time series data with respective candlestick patterns.

The timeseries data is derived from WRDS with 4h, 1h, 30min, and 5min intervals on currency and SPY futures. It is roughly 800M in total.
An example is shown below in figure 4:

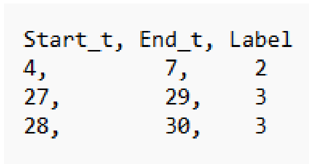


Gmt time	Open	High	Low	Close	Volume
01.08.2018 00:00:00.000	0.74190	0.74239	0.74090	0.74129	17600.48
01.08.2018 04:00:00.000	0.74129	0.74169	0.74009	0.74035	20126.4997
01.08.2018 08:00:00.000	0.74035	0.74202	0.74020	0.74129	18591.4698
01.08.2018 12:00:00.000	0.74131	0.74168	0.73982	0.74006	29374.2105
01.08.2018 16:00:00.000	0.74007	0.74070	0.73896	0.73999	19927.8303
01.08.2018 20:00:00.000	0.73998	0.74100	0.73979	0.74051	6801.539900000001
02.08.2018 00:00:00.000	0.74051	0.74110	0.73834	0.73834	15229.510199999999
02.08.2018 04:00:00.000	0.73834	0.73886	0.73726	0.73753	22457.1504
02.08.2018 08:00:00.000	0.73757	0.73784	0.73548	0.73645	28824.4999
02.08.2018 12:00:00.000	0.73645	0.73813	0.73594	0.73781	25226.9898

Figure 4: Example of Currency time series in 4h bin

For data preprocessing, I use a moving window of size 5, 10, 20, respectively and normalized the high, low, open close price in range [0,1] for each segment, which is mandate by Gramian Angular Field transformation.

The label data is stored in a separate txt file to indicate the timestamp range corresponding to a candlestick pattern. Candlestick patterns that I used are Morning Star, Bearish Reversal, Inverted Hammer, Bullish Reversal, Evening Star, Bullish Abandoned Baby, Bearish Abandoned Baby, piercing, which are encoded as 1,2,3,4,5,6,7, 8 respectively. Noticed that, all other unlabeled intervals have label 0 has default value, indicating that there's no candlestick pattern in this interval. I manually labelled 200-300 for each pattern. An example can be found below in Figure 5:



```
Start_t, End_t, Label
4,          7,     2
27,         29,    3
28,         30,    3
```

Figure 5: Training dataset label

4.2. Testing Dataset

Testing dataset consists of 200 pictures that I cropped from different online broker websites. All of them are candlestick patterns that differ in size, scale, underlying assets, date. I then change their background to the same white-gray grid line as in Yahoo-finance and compress their resolution to 64*64*3. I annotate the pictures manually with online bounding box annotator.

5. Experiments and Results

5.1. Evaluation Metrics and baseline

The metric I use is mAP>0.4:

$$avgPrecision = \sum_{k=1}^{n} P(k)\Delta r(k). \tag{4}$$

For baseline, there's no existing benchmark to evaluate how good a bounding box/ classification for candlestick pattern is. Since the definition of candlestick pattern is itself subject. So instead, I just compare whether the detected pattern in picture is similar to my own judgments.

However, I did implement a random forest classifier as the metrics baseline to compare non-CNN model performance with my RGB-GAF+CNN method. The result is shown below in key result section.

5.2. Hyperparameters

After k-fold validation (k=10), the best setup of hyperparameter for YOLO is listed below:

lr0: 0.01 # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.01 # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937 # SGD momentum/Adam beta1
weight_decay: 0.0005 # optimizer weight decay 5e-4
warmup_epochs: 3.0 # warmup epochs (fractions ok)
warmup_momentum: 0.8 # warmup initial momentum
warmup_bias_lr: 0.1 # warmup initial bias lr

The hyperparameter for Simple CNN families doesn't matter that much based on cross-validation.

5.3. Key results

The mAP>0.4 for YOLO-LITE is 0.481.
The average classification AUC_ROC for simple CNNs are 0.71. I choose to use AUC_ROC because there are much more case 0 than all others, I need to add the F-score component to deal with imbalanced dataset.

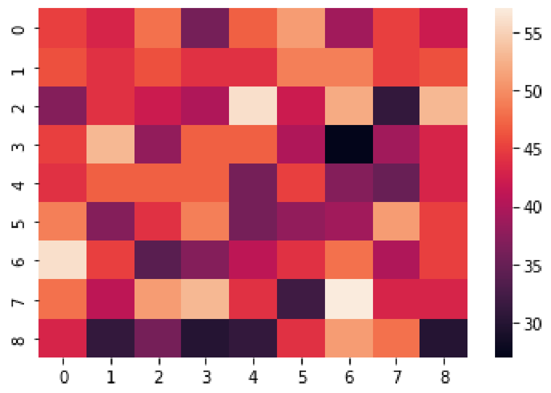


Figure: Confusion Matrix for YOLO-LITE

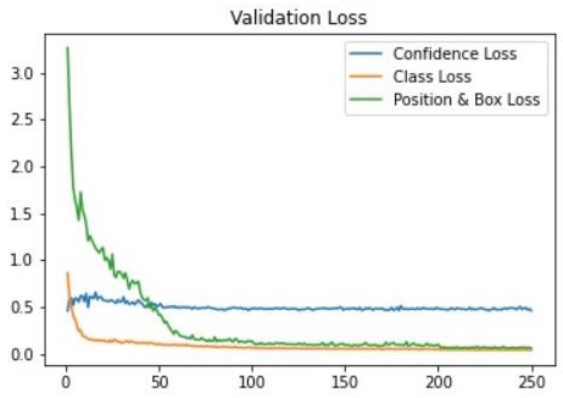


Figure: validation loss for YOLO-LITE

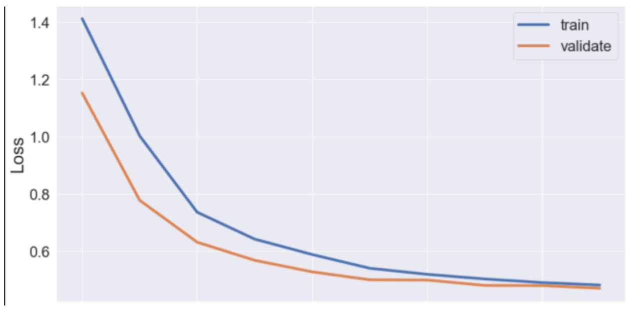


Figure: Average training/validation loss for all Simple CNNs with different moving window size

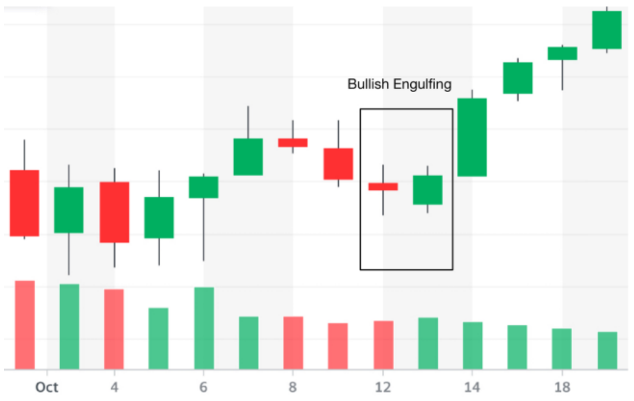


Figure: Top YOLO_LITE Classification Result

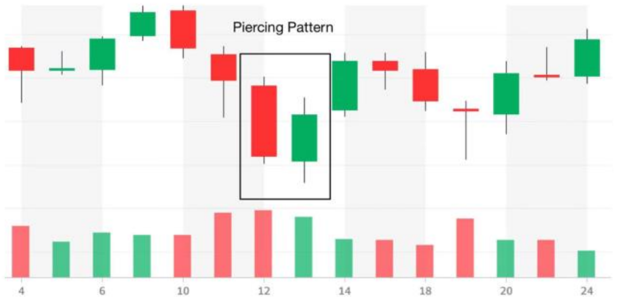


Figure: Top YOLO_LITE Classification Result

When examines misclassified test cases, some of them are caused by including extra candlesticks in the pattern, which shouldn't be considered wrong. However, for other misclassified cases, the YOLO algorithm often just bound box one candlestick and randomly claims it to be a pattern, like the figure below:

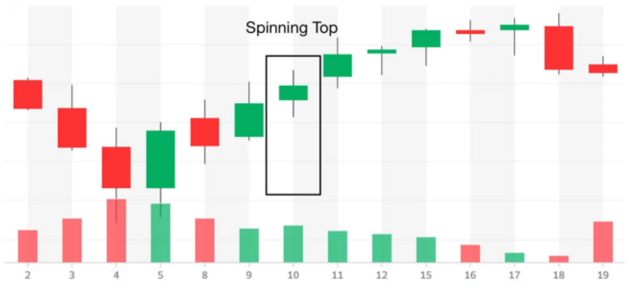


Figure: Misclassified YOLO_LITE Classification
.

I fail to find a solid explanation for these errors, since there are barely any samples in training set that consist of only one bar. I guess it may be due to the background texture I use and bad quality labeling. So, the classifier overfits to background.

If we examine the confusion matrix, we can realize that the class 8: piercing pattern is most misclassified. This is understandable because it is the only non-reversal indicator we include. All other indicators serve to indicate trend reversal while piercing indicates continuation of previous trend.

The convergence performance of both YOLO-LITE and simple CNNs are reasonable. Notice that loss of simple CNNs are much better than random forester classifier and it also converges faster. Given the training loss/validation loss plot, I don't think there exists any overfitting issues. However, the misclassification examples clearly shows that the complexity of models need to be increased or more training samples are needed.

6. Conclusion and future works

While there still exists cases that YOLO_LITE model labels far-off pictures for reasons that I cannot understand, the overall performance of both bounding box precision and classification precision is already reasonably good with a mAP>0.4 at 0.481. Part of the error is due to imperfectness in manual labelled test sets. My design of RGB Granian Angular Field, data ARIMA-CNN-based data augmentation, and model choice of YOLO-LITE contributes to the performance to a great extent. If more time, computational resource, and people are available in future, we will increase the number of precisions of testing labels and explore more completed setup for object detection model.

7. Contributions and acknowledgement

Thanks to friends who help me manually label candle charts: Shijie Guo, Jenifer Lu, and Ren Li
Thanks to the open-source code provider:

1. Yolo-Lite Team https://reu2018dl.github.io/
2. GAF python codebase: https://pyts.readthedocs.io/en/stable/generated/pyts.image.GramianAngularField.html
3. WRDS Database https://wrds-www.wharton.upenn.edu/
4. Yahoo Finance for candlestick charts picture
5. TD_Ameritrade for candlestick charts picture

8. References

[1] Tharavanij, P., Siraprapasiri, V. & Rajchamaha, K. Performance of technical trading rules: evidence from Southeast Asian stock markets. *SpringerPlus* **4,** 552 (2015). https://doi.org/10.1186/s40064-015-1334-7

[2] Morris, Greg L. *Candlestick Charting Explained: Timeless Techniques for Trading stocks and Sutures*. McGraw Hill Professional, 2006. FirstName Alpher and FirstName Gamow. Can a computer frobnicate? In *CVPR*, pages 234–778, 2005.

[3] Chen, Jun-Hao, και Yun-Cheng Tsai. 'Encoding Candlesticks as Images for Patterns Classification Using Convolutional Neural Networks'. arXiv, 2019. https://doi.org/10.48550/ARXIV.1901.05237.

[4] Wang, Zhiguang, and Tim Oates. "Imaging time-series to improve classification and imputation." In Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.

[5] Xiang, RGB GAF image: A possible solution to one weak point of Gramian Angular Field Imaging, 2022

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788

[7] R. Huang, J. Pedoeem and C. Chen, "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers," *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 2503-2510, doi: 10.1109/BigData.2018.8621865.

[8] X. Zhang, X. Zhou, M. Lin and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices", 2017.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size", 2016.

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, et al., "Mobilenets: Efficient convo-lutional neural networks for mobile vision applications", 2017.

[11] Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, κ.ά. 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python'. Nature Methods 17 (2020): 261–72. https://doi.org/10.1038/s41592-019-0686-2.

[12] Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, κ.ά. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. Στο Advances in Neural Information Processing Systems 32, επιμέλεια H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, και R. Garnett, 8024–35. Curran Associates, Inc., 2019. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.