COMS 6998:Practical Deep Learning System Performance

Final Project:
Exploring the Time Series to Image Approach

Will Sickinger (wrs2125), Chang Lei (cl3910)

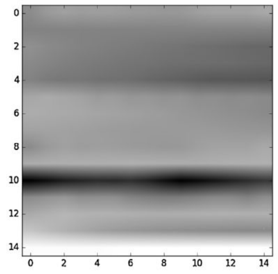
Summary

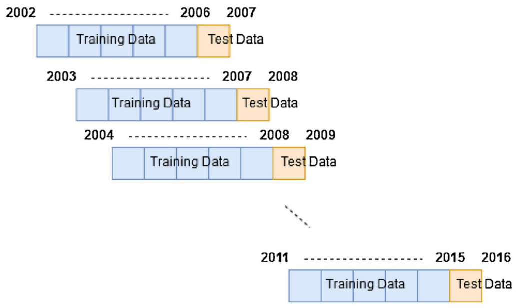
The goal of our project was to try to find the optimal times to buy and sell equities using various financial indicators. Indicators are combined to make an image and a convolutional neural network is trained on the images to find the optimal buy and sell points. We're extending the work done in a previous paper titled Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach.

Motivation

Stock market predictions are the gold standard of Financial ML because large institutions and individual investors all want to invest optimally. There is a lot of research done by banks, hedge funds and proprietary trading firms but it isn't published or shared. Any way to leverage existing methods lets you avoid starting from scratch and can save a considerable amount of time. Financial data makes for an extremely large and complicated dataset that is very interesting to work with. The features we use are calculated from open, high, low and close prices as well as the daily trading volume but there are many other variables that can be used for longer or shorter timeframes.

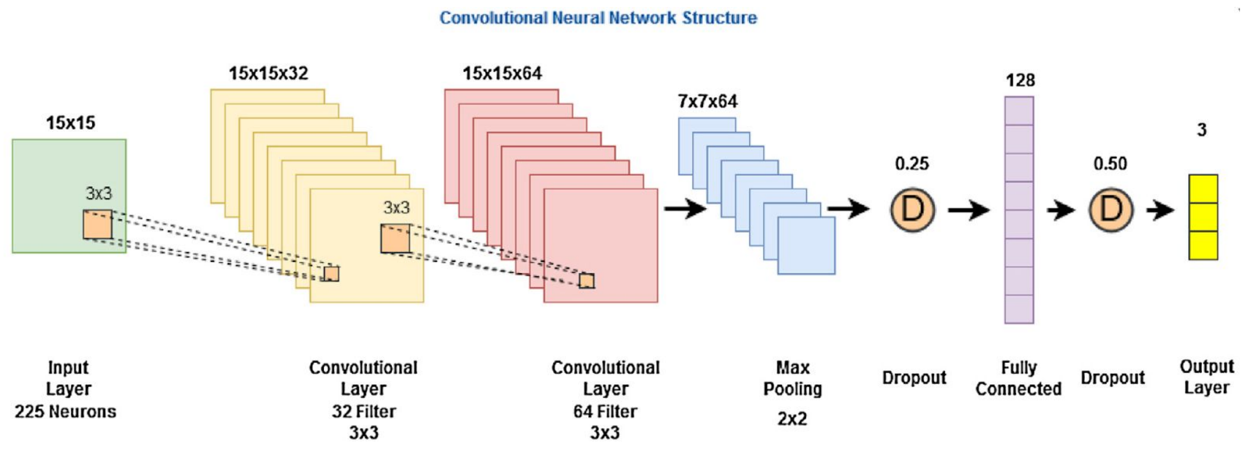
Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach





Written by Sezer and Ozbayoglu, the main idea in this paper is fairly straightforward. 15 financial features from 15 different dates are turned into a 15x15 grayscale image where the x axis is the features and the y axis is time. This image is then labeled buy, hold or sell. The labels are based on the 11 day window with the day being labeled at the center based on the end of day equity price. If that day has the highest value in the window it is labeled sell, if it is the lowest value in the window it is labeled buy. All other days are labeled hold. This pair of image and label is then used as training or test data.

For training and testing, a sliding window is used to avoid training on the test data. The model is trained on a 5 year window with the 6th year used for testing and then the window moves up one year and a new model is trained again on the new 5 year window. This happens 10 times and is meant to show that the model will provide useful predictions on multiple different time windows. For use in the real world, you would only train on the last 5 years of data and make your prediction based on current market indicators.



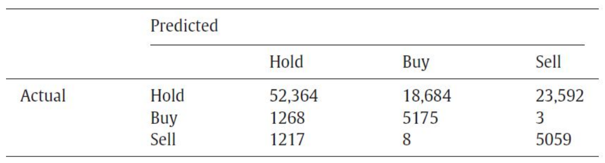
The CNN consists of 2 convolutional layers, a max pooling layer and a dense layer (see figure above).

The features used are Relative Strength Index, Williams %R, Weighted Moving Average, Exponential Moving Average, Simple Moving Average, Hull Moving Average, Trix, Commodity Channel Index, Change Momentum Oscillator, Moving Average Convergence Divergence, Percentage Price Oscillator, Price Rate of Change, Chaikin Money Flow, Average Directional and the Parabolic Stop and reverse. While the features were calculated for the images using the preceding 15 days, it's worth noting that as long as you end up with 15 data points for a feature, you can use any time difference you choose.

The network uses a 3x3 kernel which makes the order of the features in the image important. The 3x3 kernel was used to capture sharper features but the tradeoff is that if features are not near each other changes between the two features won't be captured. In the original paper, features are organized so similar features are near each other and to find local correlations.

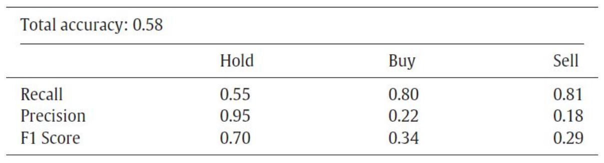
You can see some of their results below:

Confusion matrix of test data (Dow-30).



	Predicted			
		Hold	Buy	Sell
Actual	Hold	52,364	18,684	23,592
	Buy	1268	5175	3
	Sell	1217	8	5059

Evaluation of test data (Dow-30).



Total accuracy: 0.58			
	Hold	Buy	Sell
Recall	0.55	0.80	0.81
Precision	0.95	0.22	0.18
F1 Score	0.70	0.34	0.29

Technical Challenges

One of the primary challenges we ran into was that the data was very imbalanced. After labeling, approximately 90% of the samples were "hold", 5% were "buy" and the other 5% were "sell". Compounding on this difficulty was the fact that each equity has its own network so it was important to find a way to get each network to converge with the imbalanced dataset without having to fine tune them individually (particularly if you wanted to run this on every traded equity). To do this, we adjusted the sample weights used for the loss function. The weights for buy, hold and sell were set to the inverse of their frequency which resulted in the network picking labels other than hold. This wasn't quite enough though as the market performance still wasn't great. We added a parameter which was multiplied by the buy and sell weights and we found that using 2.0 worked best on our test dates.

We also had difficulty choosing useful financial indicators. We do not have any background in finance or economics so we focused on indicators involving volume as they were used less in the original feature set. In general, we just tried to choose the best features we could from the FINTA library but knowledge of what the features were meant to do would have been helpful.

Approach

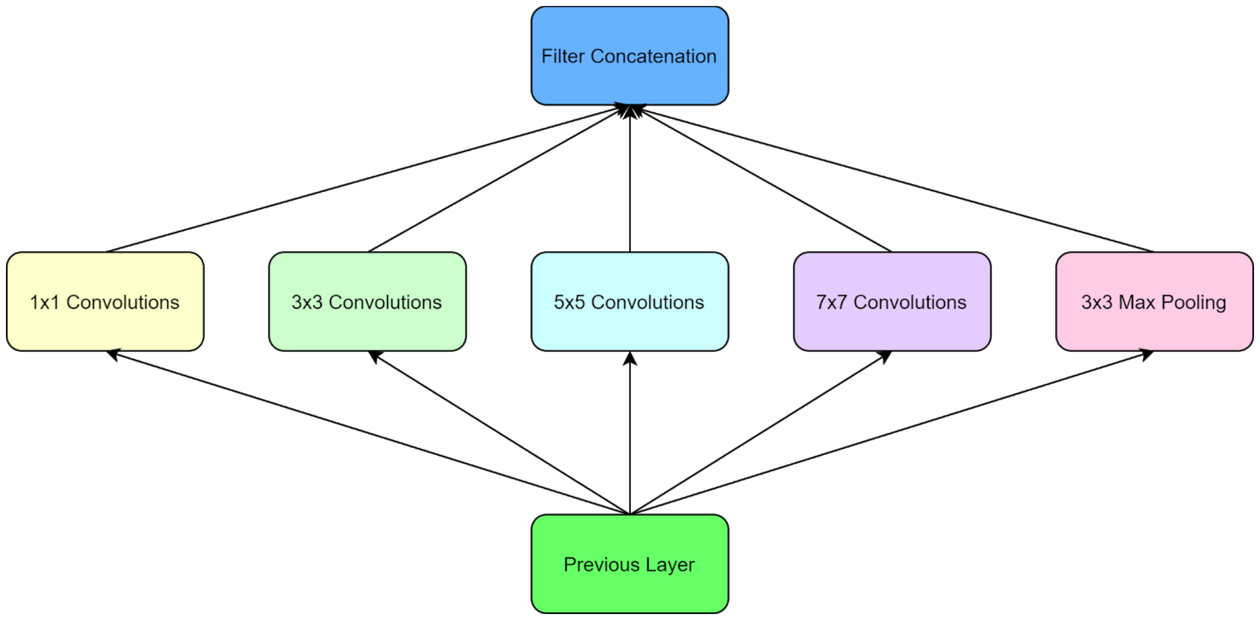
Before we could build and test a network, we had to build the images that would be used as training data. To do this we downloaded the NYSE open, close, low, high and volume data for the DOW-30 equites from Quandl, put it into pandas and processed the features for each day.

Then for the training and test periods, went through each day and looked at the 15 day period ending with the target day and put the values into a numpy array which was then converted to images with PIL. The images were then labeled buy, hold or sell.

Once we had our data, our first objective was to recreate the model from the paper as a baseline. We recreated what Sezer and Ozbayoglu called CNN-TAr using PyTorch, and set up the training loop inside another loop that built the images for each sliding window and made and stored the predictions for each test year.

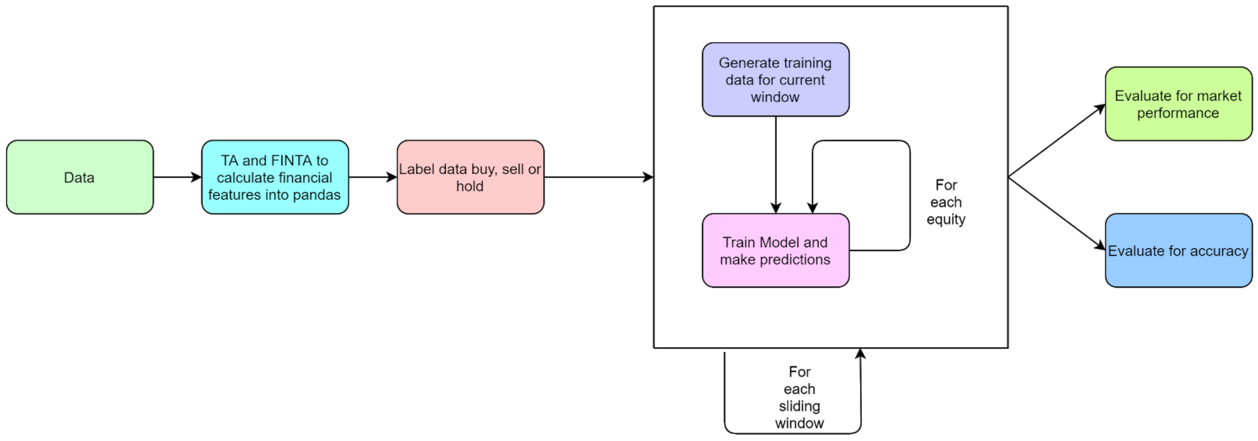
We then wanted to add different financial indicators hoping that more data would mean better results. We selected new features and built a 15x30 pixel grayscale image that contained the new and original features.

For both the baseline and the model with additional features, the data imbalance led to the models predicting only a single class. We tried to fix this using weighted sampling but it wasn't effective. We then tried weighting the losses which performed well. However, even with weighted losses, the models still found a local minima about 4% of the time. To avoid very skewed predictions, we reran the training for any model that only predicted a single class. Normally this could be an unacceptable loss of time but generating the training data and training the individual models took about 1 minute per model so retraining a new model was an effective solution.



Inception

Another potential improvement we were interested in exploring was using different kernel sizes in the convolutional layers. In the paper, they use a 3x3 convolution because they were trying to capture shaper features but this could miss correlations between features that are not spatially near each other in the images. We were considering multiple models with larger kernels but decided on Inception blocks so that we could try all of them at once. The hope was that correlations between more features may be good indicators of future price movements. We ended up using kerns of size 1x1, 3x3, 5x5 and 7x7 (see architecture above). We used the naive version of Inception blocks as we only used 2 layers so we were not concerned with the number of parameters exploding.


Data
TA and FINTA to calculate financial features into pandas
Label data buy, sell or hold
Generate training data for current window
Train Model and make predictions
For each equity
For each sliding window
Evaluate for market performance
Evaluate for accuracy

Pipeline

Our pipeline can be seen in the above diagram. For each equity, data is downloaded from an online data source into a pandas dataframe. TA and FINTA are used to calculate the financial features and each trading day is labeled in the dataframes. Then, for each sliding window in the train/test range, The training data is generated for each equity, then for each equity a model is trained and predictions are made on the test date range. Once predictions have been made for every equity in each sliding date window, the data is evaluated in two ways. First for the precision, recall and F1 score and second for the market performance of the predictions made using annualized returns as the metric.

Implementation Details

All models were trained on either a single RTX2070 or Tesla T4. Ideally these models would have been run on multiple GPUs but at the time these experiments were performed, there was a shortage of available GPUs on Google Cloud Platform. Each date window for each equity trains a new model independent of the others so there is no downside to training in parallel on as many GPUs as are available. The models used by Ozbayoglu and Sezer were run using PySpark to distribute training which would have reduced overall training time significantly.

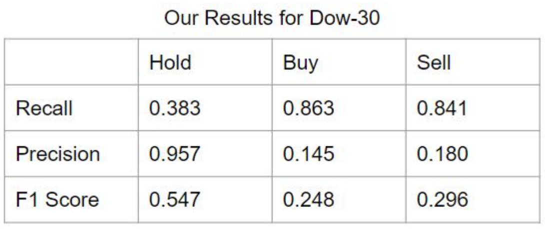
Our models were implemented in PyTorch. The daily equity pricing is downloaded from Quandl (if you are using this code you will need to get your own quandl key or use Yahoo Finance). The Python Technical Analysis (TA) and Financial Technical Analysis (FINTA) are used to calculate the financial features. Scikit learn is used for feature preprocessing.

Each network is trained on approximately 1250 15x15 (15x30 for the additional features) images with batch size of 128 for the baseline and model with additional features and 512 for the Inception model so training was very fast. 100 epochs took approximately a minute. However, for the 10 year window 10 models are trained and 28 equities were evaluated so the overall training time was roughly 250 minutes. This could be reduced by using a faster GPU or distributing training.

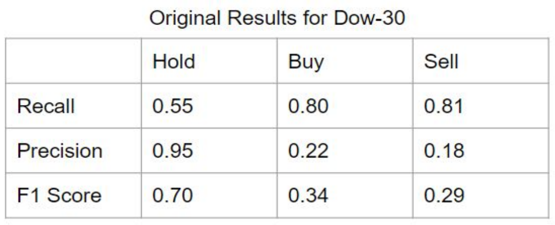
Results - Baseline

The overall performance of our model is evaluated using two different criteria: computational model performance and financial evaluation. Computational model performance evaluation presents the network performance, i.e. how well the classifier distinguishes between the buy, hold and sell classes. Financial evaluation shows the performance of the whole model by implementing the real world financial scenario. Stocks are bought, sold or held according to the predicted label with the actual stock prices.

For computational model performance, we present the recall rate, precision rate, and F1 score. Our model achieves a higher recall rate on buy and sell compared to the results from the original paper. It tended to be more aggressive and lean towards buy or sell over hold. It is difficult to make it more aggressive as it will fall into all buy or all sell. The precisions of buy and sell are relatively low, mainly because buy and sell points appear much less frequently than hold points, which dominate the data distribution. That said, the precision, recall and F1 scores can be lower if it still makes good returns. It's still a good trade if it buys right before or after the lowest point.

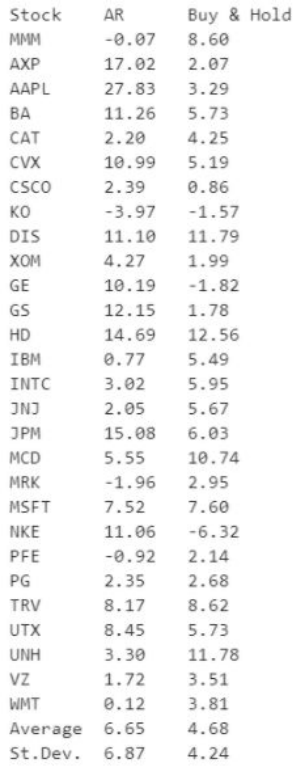

Our Results for Dow-30

	Hold	Buy	Sell
Recall	0.383	0.863	0.841
Precision	0.957	0.145	0.180
F1 Score	0.547	0.248	0.296

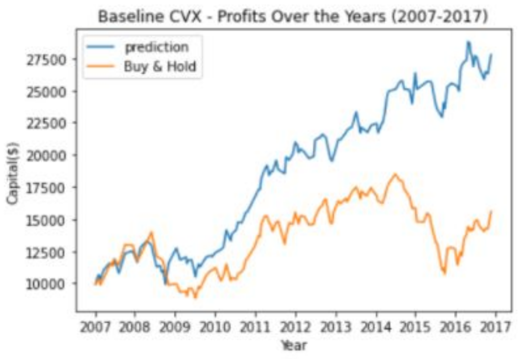

Original Results for Dow-30

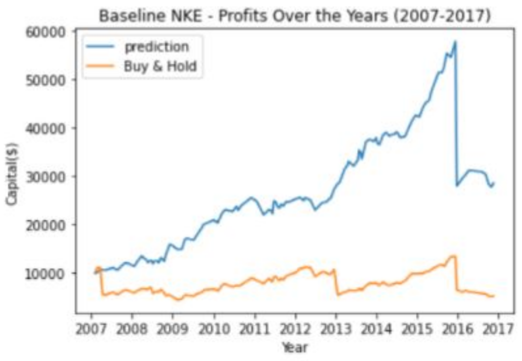
	Hold	Buy	Sell
Recall	0.55	0.80	0.81
Precision	0.95	0.22	0.18
F1 Score	0.70	0.34	0.29

For financial evaluation, we compare our model performance with the buy and hold strategy. In the Buy & Hold strategy, the stock is bought at the beginning of the test period and sold at the end of the test period. For our model, if the predicted label is “buy”, the stock is bought at that point with all of the current available capital. If the predicted label is “sell”, the stock is sold at that price, if any is held. If the predicted label is “hold”, no action is taken at that point. The starting capital for financial evaluation is $10,000 and we use the annualized return for the evaluation metric.



```
Stock    AR      Buy & Hold
MMM      -0.07   8.60
AXP      17.02   2.07
AAPL     27.83   3.29
BA       11.26   5.73
CAT      2.20    4.25
CVX      10.99   5.19
CSCO     2.39    0.86
KO       -3.97   -1.57
DIS      11.10   11.79
XOM      4.27    1.99
GE       10.19   -1.82
GS       12.15   1.78
HD       14.69   12.56
IBM      0.77    5.49
INTC     3.02    5.95
JNJ      2.05    5.67
JPM      15.08   6.03
MCD      5.55    10.74
MRK      -1.96   2.95
MSFT     7.52    7.60
NKE      11.06   -6.32
PFE      -0.92   2.14
PG       2.35    2.68
TRV      8.17    8.62
UTX      8.45    5.73
UNH      3.30    11.78
VZ       1.72    3.51
WMT      0.12    3.81
Average  6.65    4.68
St.Dev.  6.87    4.24
```





The left-hand side chart presents the annual return percent rate of 28 stocks for both our model and the Buy & Hold strategy. Our model's performance is in the middle "AR" column. On average our model achieves 6.65% annual return rate, which is higher than that (4.68%) of the Buy & Hold strategy. We pick two stocks and plot their profit-over-year graph, shown on the right-hand side. It looks like our model primarily holds but it sells before some critical drops. If you look at the NKE (Nike) stock in 2013, it drops but our model has already sold. The same is true of CVX in 2015.

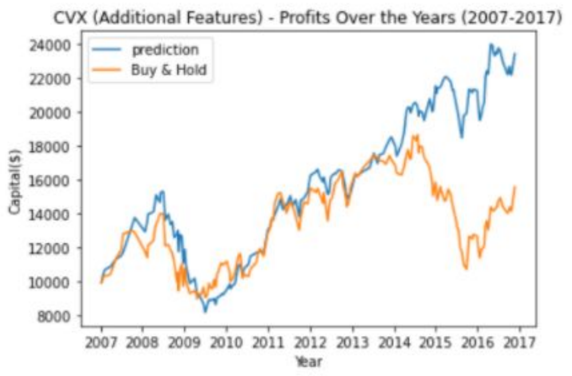
Results - Additional Features

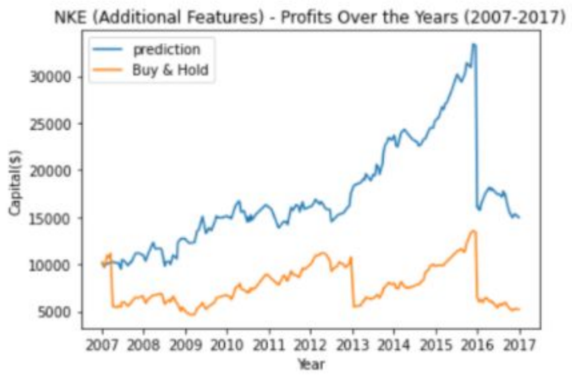
We add new financial features in our next experiment to see if this will bring any improvements. Additional features really didn't lead to a large change in the F1 score. We experimented with a number of features in the FINTA library and saw similar results. It's possible that the network has learned all that it can from features that are derived from the same base information. If this was linear machine learning, it probably would have improved but deep learning may have already derived and used the base features. External features may be worth trying in future work, such as daily news sentiment, percent profits or twitter mentions.

Our Results for Dow-30

	Hold	Buy	Sell
Recall	0.433	0.808	0.792
Precision	0.948	0.151	0.177
F1 Score	0.594	0.254	0.289

```
Stock    AR      Buy & Hold
MMM      -1.32   8.60
AXP      5.47    2.07
AAPL     23.32   3.29
BA       9.72    5.73
CAT      -3.47   4.25
CVX      9.36    5.19
CSCO     3.18    0.86
KO       0.47    -1.57
DIS      14.76   11.79
XOM      3.37    1.99
GE       9.12    -1.82
GS       3.61    1.78
HD       10.46   12.56
IBM      -2.51   5.49
INTC     -5.20   5.95
JNJ      3.56    5.67
JPM      14.34   6.03
MCD      4.67    10.74
MRK      -2.19   2.95
MSFT     7.45    7.60
NKE      4.12    -6.32
PFE      -0.14   2.14
PG       5.61    2.68
TRV      5.66    8.62
UTX      2.77    5.73
UNH      3.62    11.78
VZ       -1.10   3.51
WMT      1.08    3.81
Average  4.64    4.68
St.Dev.  6.13    4.24
```





As for the financial evaluation, there is no obvious improvement either. The average annual return is even slightly lower than that of the Buy & Hold strategy. And for the two stocks we picked for profit-over-year graph plotting, they exhibit similar behavior with the Buy & Hold strategy.

Results - Inception Blocks

In our last experiment, we tried adding inception blocks to the model. Overall, the returns were much lower than the baseline model and the precision, recall and F1 scores were also somewhat lower. Note that, however, the recall rate for buy is significantly lower than before (and therefore a high precision for buy). This indicates that the model becomes very cautious about buying and tends to hold all the time.

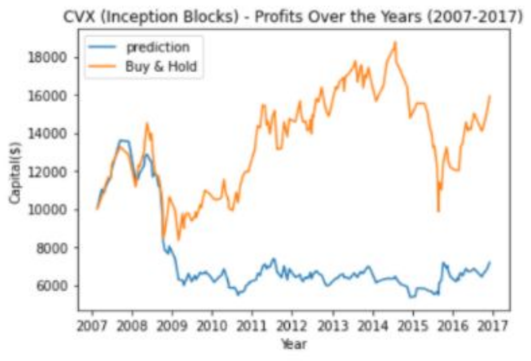
Our Results for Dow-30

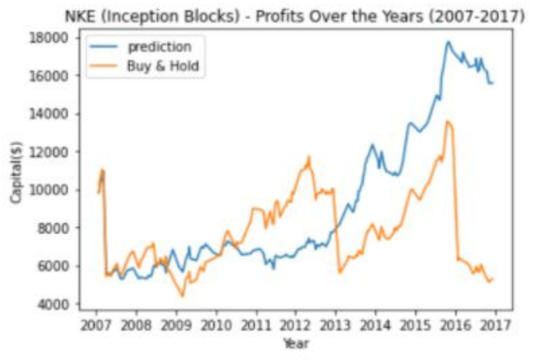
	Hold	Buy	Sell
Recall	0.993	0.441	0.772
Precision	0.869	0.936	0.175
F1 Score	0.927	0.600	0.286

This may be the result of using simple Inception blocks. However, using the full model on images this size was not possible. Inception takes in images of size 299x299 and if we resized our images to that size, the largest filters would still only be able to see the values of 4 pixels at a time which is what the base model does now.

Similar to what was observed for the recall, precision and F1 scores, the average annual return performs poorly compared to the Buy & Hold strategy. Being more cautious about buying, however, seems to make the profits more stable.

```
Stock     AR      Buy & Hold
MMM       8.57    8.60
AXP       5.86    2.07
AAPL      -12.98  3.29
BA        4.38    5.73
CAT       1.96    4.25
CVX       -3.25   5.19
CSCO      4.46    0.86
KO        -3.50   -1.57
DIS       7.51    11.79
XOM       -1.41   1.99
GE        -3.18   -1.82
GS        1.90    1.78
HD        4.85    12.56
IBM       3.05    5.49
INTC      4.52    5.95
JNJ       -0.63   5.67
JPM       12.33   6.03
MCD       7.04    10.74
MRK       5.64    2.95
MSFT      -1.66   7.60
NKE       4.53    -6.32
PFE       5.39    2.14
PG        2.79    2.68
TRV       -2.24   8.62
UTX       9.99    5.73
UNH       19.61   11.78
VZ        4.29    3.51
WMT       3.51    3.81
Average   3.33    4.68
St.Dev.   5.89    4.24
```





Conclusion

Based on our results, the baseline model had the best performance, which was surprising. We're going to continue to tune inception to see if we can improve its predictions.

We believe that there is merit to this method. The returns over the 10 year period were noticeably better than buy and hold. Any trading firm or hedge fund would easily have the resources to run these models daily on every equity they trade given the short training time. Adding extra overhead time, at 2 minutes of training time per model you would only need 12 GPUs to train all models for the 2800 equities traded on the NYSE in 8 hours. This would take even less if faster GPUs were used. That said, we would not recommend using this model on its own but instead would suggest using it in an ensemble of models and methods with the goal of leveraging the better returns but having other methods help maximize returns when there is less significant price movement.

Future Work

There are a number of ideas that came up while working on this project that we believe would be worth exploring. While we didn't see improvement from the financial features we added it is possible that there would be better results from extending the time window for the features, i.e. making the image larger in the direction of time. This might also be useful for making longer term predictions if you were looking to make changes to a portfolio weekly instead of daily.

We were also interested in using an ensemble of multiple models. The order of the features in the image was chosen so that correlated features would be near each other. Rather than the Inception block approach, we were interested in trying different feature orders and then using a majority vote to determine the label. Unfortunately with limited GPUs this wasn't a feasible experiment due to our time constraints.

While it was outside of the scope of what we were trying to accomplish, it would be worth analyzing what equities this works on. It's possible that the equities where it was most effective compared to buy and hold are in the same sector or have certain things in common. Ideally, you would experiment with this on a larger group of equities. If this method was particularly effective in certain areas, you could focus your capital there or simply use different methods for different equities.

We used two inception blocks in our Inception model because of the small size of the images. Using inception blocks with dimension reductions and using more of them could potentially improve network performance. We were surprised that the inception blocks didn't improve market performance and still believe that if the correct architecture was found it would improve the returns.

It's also possible that a completely different architecture would yield better F1 scores and returns. To find this a Neural Architecture Search could be run on the sample images to see if a different approach would be worth exploring.

References

Ozbayoglu and Sezer, Algorithmic financial trading with deep convolutional neural networks:

Time series to image conversion approach:
https://www.sciencedirect.com/science/article/pii/S1568494618302151

Szegedy et al. Going Deeper With Convolutions: https://arxiv.org/abs/1409.4842