Debunking EMH and Making Fat Stacks

Kiersten Engel*, Christopher Ong*, Andrew Pecka*, Steven Rivadeneira*
Georgia Institute of Technology
dlprojectteam@groups.gatech.edu

Abstract

So called 'Chart Patterns' are roundly used by stock traders to discern signals regarding the future direction of returns on equities. Juxtaposed against the Efficient Market Hypothesis - which posits that no such pattern should give an investor an advantage - we endeavored to confirm or deny that is the case. To do so we trained several flavors of Convolutional Neural Networks, and ultimately cutting-edge Gated Recurrent Units using 25 years of historical Candlestick Charts to see if we could outperform benchmark historic returns across several dozen industries.

1. Introduction/Background/Motivation

Our objective was to predict future stock market price movements using deep learning models to test the widely held belief that historical price information cannot be leveraged to achieve higher stock market returns. We tested this theory (formally known as the Efficient Market Hypothesis or EMH) by evaluating the returns of multiple price prediction models and comparing to the overall market.

Forecasting stock market prices is very difficult because price movements can be influenced by many factors such as economic policy, earnings reports, political crisis, weather events, etc.[5] The state-of-the-art forecast models used by top private finance firms in algorithmic trading space is hard to know since there are large amounts of money at stake and private firms are unlikely to make their latest research public. [2]

However, forecasting time series has been studied extensively in the field of Deep Learning and there are various studies applying techniques such as CNN [3, 8], LSTM[9], DQR (Deep Q- learning neural net) [1], and DPP (Deep Predictor for Price Movements) [5] to stock market price prediction. Many of the market-specific studies tend to be focused on individual companies or assets, which are subject to firm-specific risks and other exogenous risk factors which can create noisy data and pose challenges for training. Moreover, they are typically only evaluated on a single asset or market rather than as a tradeable strategy which limits their usefulness to investors due to lack of diversification.

Successfully predicting stock market price movements is the key to making money, the objective of every investor. Investors spend millions on stock market research [10] and are constantly looking for models that consistently outperform the market. Moreover, successfully predicting future stock market prices is further support against EMH, the assumption that financial markets are efficient, which is a cornerstone of model finance and dictates many how many investors formulate strategies.

1.1. Dataset

Our dataset consists of candlestick charts[1] created for this project using pricing and volume data for all U.S. stocks provided by Ned Davis Research (NDR) [2]. The data is maintained by NDR's market data team and sourced from multiple vendors, which enabled us to generate charts using very clean pricing information. All the data is adjusted for dividends, stock splits, and corporate actions and includes all stocks (both currently active and delisted stocks) to ensure we are not introducing look-ahead bias into our analysis.

To mitigate firm-specific risks (CEO dies, Lawsuit, etc.) which can create additional noise in the training data, we grouped similar companies together according to their GICS industry, using historical membership to avoid look-ahead bias. For each of the 56 groups, we computed the aggregate price and volume indexes, using daily data from June 1996 through May 2021. Using the aggregated industry data, we generated two types of candlestick charts shown in Fig. 1.

The 64-by-64-pixel color chart shows a variant of the traditional candlesticks (known as a bar chart) that includes the same information, but is more compact allowing us to show more history in a smaller space. It also includes a 15-week moving average and standard deviation bands. The 48-by-48-pixel grayscale charts show traditional candlesticks, but only include 3 weeks and incorporate volume information.

[1]See appendix for detailed overview of candlestick charts
[2]See appendix for compliance approval

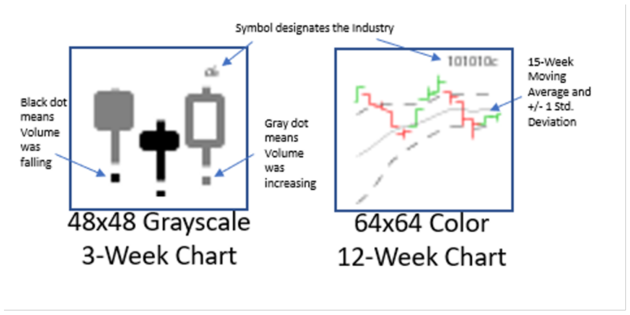

Symbol designates the Industry
Black dot means Volume was falling
Gray dot means Volume was increasing
101010c
15-Week Moving Average and +/- 1 Std. Deviation
48x48 Grayscale 3-Week Chart
64x64 Color 12-Week Chart

Figure 1. Candle Stick Charts

A black dot indicates falling volume, and a gray dot indicates rising volume relative to the prior week. Each chart image is associated with percent return for following week, which was used for training purposes. These returns are discretized to generate labels for our charts, which is discussed in the Experiments section.

2. Approach

To predict stock market returns, we built and compared three models, hand-selected from our literature review: Vanilla CNN [8], Resnet [4], and Deep Predictor for Price (DPP) [5]. These architectures utilize convolutional neural networks (CNN), which are well known for image processing, pattern recognition and classification tasks[3]. Furthermore, the CNN architecture mimics how human analysts examine candlestick charts to identify patterns that are used to predict future price movements. The DP architecture expands on this by incorporating a time component that considers the evolution of the stock price over time using a GRU.

We structured our objective to predict future price movements using our candlestick charts dataset as a classification problem. Given the high noise-to-signal ratio in stock market returns data, we cannot expect a model to predict exact returns, and attempting to do so can lead to severe overfitting. We use thresholds to discretize the returns into classes as discussed in the Experiments section. Structuring the predictions as classes is also easily transitioned into a trading strategy enabling us to evaluate each model's performance statistics.

It is important to note that this is not a pure classification problem, as incorrectly classifying a -.1% return and a -10.0% lead to very different outcomes. We improve upon existing studies by evaluating each model as a trading strategy and assessing the quality using financial metrics on the performance (discussed in detail in the Experiment and Evaluation Section). Another notable improvement is our selection of input data. Not only were we able to incorporate a wider variety of candlestick charts into the training process, but also reduced noise in the training dataset by using aggregate groups of stocks to facilitate learning. Using groups of stocks mitigates exogenous risks which can create excessive noise limiting the model's ability to learn. [7]

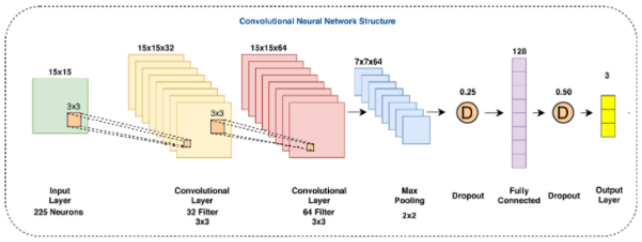


Figure 2. Vanilla CNN Architecture

We hypothesize our model will be successful as many traders have used candlestick charts to predict future price movements with success. Additionally, several papers have demonstrated success. Our changes to the model architectures in hopes of improvement are minimal and we believe success will be translated at worst and improved upon at best.

2.1. Selected Model Architectures

2.1.1 Convolutional Neural Network (CNN)

Convolutional neural networks were amongst the first architectures we experimented with, including our own CNN, VGG, Alexnet, and Resnet. The models accepted a 64x64 image and output a predicted return classification. Our CNN models used cross entropy loss and stochastic gradient descent for the loss function and optimizer. While we tested many alternatives, we focus on the Vanilla CNN and Resnet for our paper.

2.1.2 Vanilla CNN Architecture

Our Vanilla CNN was derived from work by Sezer et al. [8] which used a 6-layer CNN accepting 15x15 images consisting of 15 indicators and classifying into buy, sell, and hold classifications. Due to the size of our input images, we designed a deeper 27-layer model, see Fig. 2.

Our 27-layer CNN has more depth and model capacity than VGG and Alexnet, which allow it to learn more abstract and subtle patterns. The model is built on a convolution layer, max pooling layer, a second convolution layer, a second max pooling layer, a relu layer, and a normalization layer. This sequence is repeated two more times and followed by three fully connected layers. The model uses decreasing kernel sizes to capture more details deeper in the model. We chose a deeper model to accommodate our larger input image size relative to the original paper's model.

2.1.3 Resnet Architecture

The ResNet model was selected because of its notable "residual" architecture that leverages the so-called "identity shortcut" for more efficient learning. ResNet20, ResNet50, ResNet 110, and ResNet 1202 were all explored. All ResNets use the same basic building blocks of convolutional layer, residuals. Deeper ResNets use more of these layers while shallower ResNets use less. Normalization layers are used throughout the ResNets to avoid the vanishing gradient problem. The increased model capacity from the deeper ResNets, coupled with the residual architecture may prove useful in abstracting features from the stock chart data set, so we opted to work with the ResNet 110 architecture in our experiments.

2.1.4 DPP Architecture

The original DPP architecture is broken down into three components: the Chart Decomposer, CNN Autoencoder, and an RNN with a GRU gating mechanism [5]. This architecture was published by Hung C-C et al. in June 2021, therefore there is no code available, so we built it from scratch with several modifications to accommodate our specific dataset.

2.1.5 Chart Decomposer

The original architecture used a "Chart Decomposer" to break a single chart containing 20+ candlesticks into ordered, 3-candlestick image chunks. We were able to bypass this step by generating our 48x48x1 Grayscale 3-week candlestick charts directly. We were also able to add information about the corresponding volume (discussed in Datasets section) which helps corroborate price movements. It is important to note that using smaller, ordered sub-charts is critical because it increases the signal-to-noise ratio by reducing the empty whitespace (noise) in the input images. Additionally, it allows for higher resolution of the 3-candlestick patterns in a smaller pixel size which drastically reduces training time and memory requirements.

2.1.6 CNN Autoencoder

As done in the original paper, we used a CNN Autoencoder extract deep features from the images before passing to the GRU. The CNN Autoencoder contains an encoder layer, responsible for deconstructing the 48x48x1 image into 576 deep features, and a decoder layer, responsible for reconstructing the original image solely using the deep features identified by the encoder. The encoder block is constructed using four convolutional layers, with ReLU activation followed by a pooling. The decoder process reverses encoder process by replacing the pooling layers with up-sampling

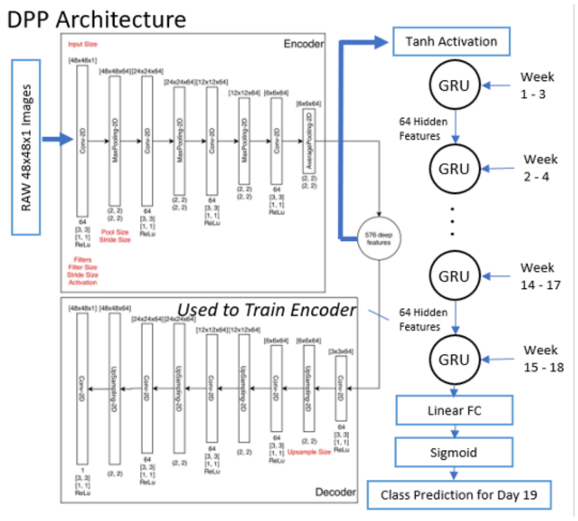


Figure 3. DPP Architecture [5]

layers to return the original 48x48x1 image. Exact parameters are depicted in Fig. 3.

The autoencoder model is trained by minimizing the differences between the original image and the image produced by the decoder. We initialized the weights by training a few Epochs using a binary cross entropy loss function, as specified in the original paper. However, because our images incorporated more shades of gray, we adjusted our loss function to use Mean Squared Error which was better able to capture the grays in our original images. While we do not need the decoder layer when running the final DPP model, it is critical for training the encoder to identify the correct features.

2.1.7 RNN with GRU

The third stage of the DPP model uses a GRU to predict the price movements as Class 0 indicating negative expected return, and Class 1 indicating positive expected returns. We opted to use the GRU architecture as it avoids the vanishing gradient problem that plague RNNs and require fewer parameters than LTSM which reduces training time. The GRU model inputs are the 576 deep features that are produced by passing raw images through the fully trained encoder layer. Our 1-layer GRU model is configured to process 18 encoder-processed images, in sequential order. The output of the GRU is followed by a Sigmoid activation and a fully connected layer that produces a predicted class for day 19. This architecture and associated parameters are depicted in Fig. 3.

2.2. Challenges

Memory pressure of our training set, coupled with a paucity of GPU RAM was our primary concern prior to experimentation. Our original dataset consisted of 600 x 800 pixel images having three color channels of red, green, and, blue. In practice our concerns we're validated.

By default, our deep learning library, PyTorch, marshalled our images into 64-bit floating point tensors consistent with the native architecture of our GPUs. The easiest optimization was simply to cast our tensors to 32-bit floating point tensors, which halved our working set for a batch-size of 64 from 1.475GB to 0.735GB. Coupled with our 1.7M parameters for ResNet-110 we had a working set under ¾ GB which theoretically fit with our 1GB capacity. After only a few iterations however, even with forced garbage collection, we exhausted available memory.

We took two approaches to addressing our capacity limits: scaling and channel reduction. The later entailed using only two-color channels in the images and representing each as a bit array of size 600 x 800. This was an efficient representation but ultimately fruitless as the underlying CUDA subsystem needed to cast each bit back to a 32-bit float before performing multiplication on model parameters.

Finally, we settled on a strategy of image reduction reducing to 64 x 64 and 48 x 48 respectively allowed us (while keeping 32-bit floats) to train batch sizes upwards of 256 without exceeding available GPU resource.

3. Experiments and Results

In this section, we compare the three architectures outlined above. We explore the impact of hyperparameters on model training, test possible discretization options to avoid class imbalance, and finally evaluate the performance of model predictions in a trading strategy context.

3.1. Hyperparameter tuning

We trained our models using a systematic approach. We started by implementing a grid search to iterate over each of our models using a common set of parameters: Epochs, Learning Rate, Momentum, Regularization Parameter, Burn-In, Learning Rate Decay Thresholds, Batch Size, and Shuffle. Definitions of these parameters can be found in the appendix.

We ranged our epoch as high as 200 but observed that in most cases learning converged (or diverged) by 50 Epochs. As such, our Epoch search space was limited to [10, 25, 50]. The learning rates tested consisted of [0.1, 0.01, 0.001, .0001]. For momentum [0.1, 0.5, 0.9] were trained.

After running an initial grid search, we manually tuned each model using trial and error which worked as a defacto Random Search [6]. The initial grid search observations, as mentioned with epochs, where instrumental in being able to cover a large search space in the short time frame we had to execute this project.

The most import observation we made was that our ResNet-110 model could not achieve learning when we increased regularization parameter from the default of 0.00005. This is because our ResNet-110 must fit over 1.7 million parameters using only 52K distinct training images. Even with the default regularization parameter at 0, it was only a matter of time before the model fit entire training set. This overfitting observed in our increasing validation Loss curves for ResNet and CNN shown in Fig. 5.

3.2. Choosing Discretization Thresholds

As discussed above, each chart in our dataset included the raw percent return that we are predicting. This gave us flexibility to discretize the data in three different ways and determine which is most effective.

We started by choosing "common sense" thresholds of [-1.5%, -0.5%, 0.0%, 0.5%, 1.5%] to separate the data into 6 classes. The appeal to this approach is that it is intuitive, and it avoids introducing look-ahead bias that results from choosing thresholds using the full history of training data. The drawback is that it is not guaranteed to produce balanced classes for use in model training.

Given that class balance is a known problem for training deep learning models, our second approach ignored the look-ahead bias issue, and determined the class thresholds by taking the median of the entire dataset over time, computing quartile thresholds, and averaging them. This method ensured classes were well balanced, by separating them into 4 bins using thresholds of [-1.7%, 0.3%, 2.3%].

Our third approach simply set "0.0" as the threshold, and separated returns into 2 classes.

The balanced approach resulted in the Resnet model being able to predict class 0 especially well. This, in part with maintaining good enough overall accuracies across the other classes, improved ResNet's accuracy and performance over the multi-threshold imbalanced approach. (see Fig. 5 for comparison)

We found that the DPP model was most successful using the simple 2 class structure. The simplicity of this approach produced the best model performance and is least prone to overfitting and inadvertent introduction of bias.

3.3. Performance Evaluation

We evaluated the success of our model from two perspectives. First, because our models are predicting return classifications, we computed the overall accuracy and per-class accuracy for each model. Second, because we are interested in the performance as a tradeable strategy, we compute a back-test using our model predictions and evaluate it using standard portfolio performance metrics.

3.3.1 Accuracy

We computed the overall and per-class accuracy for each of the models on validation and test data and report our results in Fig. 5.

Our observed accuracies for our test sets are only slightly above random chance for most of our models, with Resnet producing the highest overall accuracy of 64%.

While overall and per-class accuracy offer transparency and are useful for tuning hyperparameters, they are not representative of how the models performed as trading strategies. This is because we can afford some misclassifications in our portfolio provided, we are correctly identifying the large price movements. For example, misclassifying a -.01% return is not as costly as misclassifying a -10.0% return. Additionally, market data has a very high noise-to-signal ratio, so small differences observed in accuracy readings were likely due to noise and were not useful for distinguishing which model will outperform as a trading strategy.

3.3.2 Model Performance

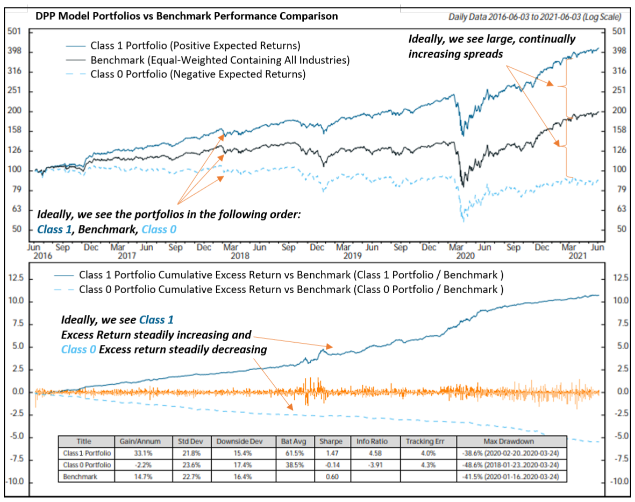
To round out our model assessment, we computed a back-test using our model predictions and evaluate it using Percent Gain and Excess Return. These metrics are computed as follows:

$$Percent\ Gain = \frac{Ending\ Porfolio\ Value}{Starting\ Portfolio\ Value} - 1$$

$$Excess\ Return\ =\ Porfolio\ -\ Benchmark\ Return$$

To generate a back-test from our model predictions, we use our test data set (from 2016 – 2021) and simulate a trading strategy that invests in each class. For example, consider our DPP model that predicts two classes: Class 0, which represents negative expected returns, and Class 1 which represents positive expected returns. Our back-test consists of two portfolios: Class 0 Portfolio that only invests in industries labeled "Class 0" and a Class 1 Portfolio that only invests in industries labeled "Class 1." To simulate performance, we use the model to generate predictions for each week of our test data and rebalance the Class 0 and Class 1 portfolios to invest in the industries assigned to their respective classes. We then compute the actual returns for the week and compound them to generate a simulated performance. To round out the assessment of our model's performance, we also create an equal-weighted benchmark which invests in all industries every week to simulate the "buy-hold" scenario (or the control group).

The top performing model was the DPP, which is likely due to the addition of the time component. Figure 4 shows that our Class 1 Portfolio produces consistent, strong outperformance relative to our benchmark, and our Class 0 Portfolio produces consistent, strong underperformance relative to our benchmark. This is best observed by examining the bottom portion of the chart which shows the excess return of each portfolio relative to our benchmark. The



Title	Gain/Annum	Std Dev	Downside Dev	Bat Avg	Sharpe	Info Ratio	Tracking Err	Max Drawdown
Class 1 Portfolio	33.1%	21.8%	15.4%	61.5%	1.47	4.58	4.0%	-38.6% (2020-02-20..2020-03-24)
Class 0 Portfolio	-2.2%	23.6%	17.4%	38.5%	-0.14	-3.91	4.3%	-48.6% (2018-01-23..2020-03-24)
Benchmark	14.7%	22.7%	16.4%		0.60			-41.5% (2020-01-16..2020-03-24)

Figure 4. DPP Performance

consistent, steady positive slope of the Class 1 Portfolio indicates that our model predictions are consistently adding value. And the consistent, steady negative slope in our Class 0 Portfolio shows that our negative predictions are correctly identifying underperforming companies.

4. Conclusions

Our DPP and Resnet model predictions demonstrated consistent predictive power for both the positive and negative predictions, indicating that we successfully achieved our objective to predict future price movements. This finding also provides strong support against EMH, which states that markets are efficient at capturing information. Our models use the latest machine learning algorithms, many of which did not exist during the back-test period. Therefore, it is not surprising that we were able to find inefficiencies in the market using our models. As these models become more mainstream, we will likely see the performance deteriorate as the market begins to adopt these models, returning it to a fully "efficient" state.

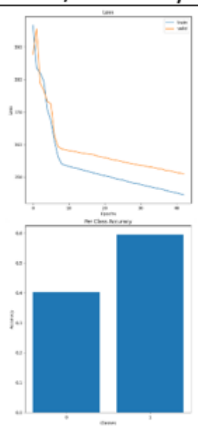
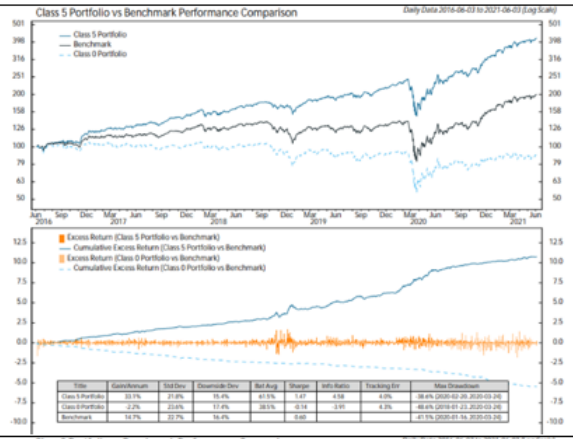
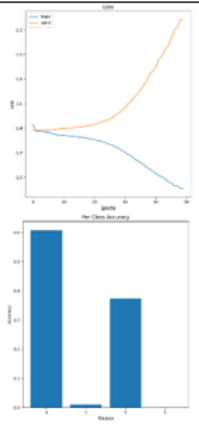
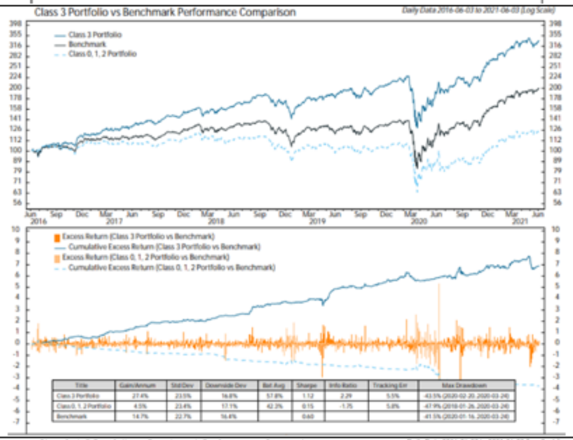
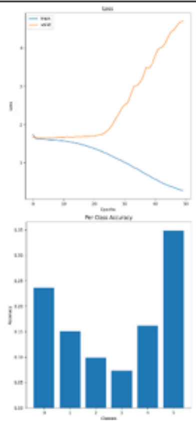
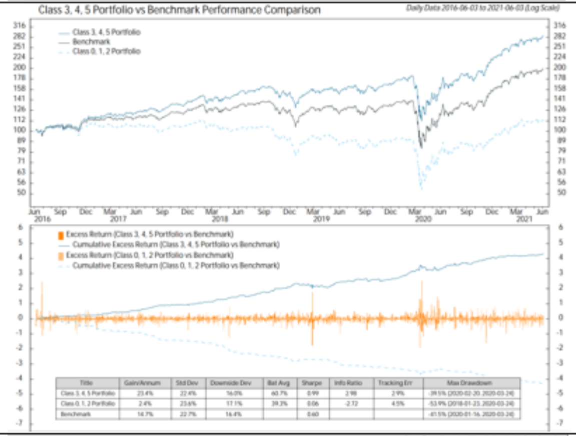
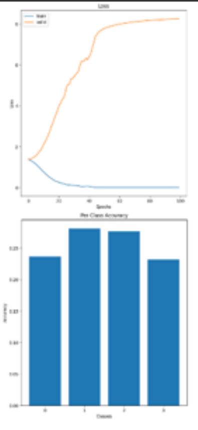
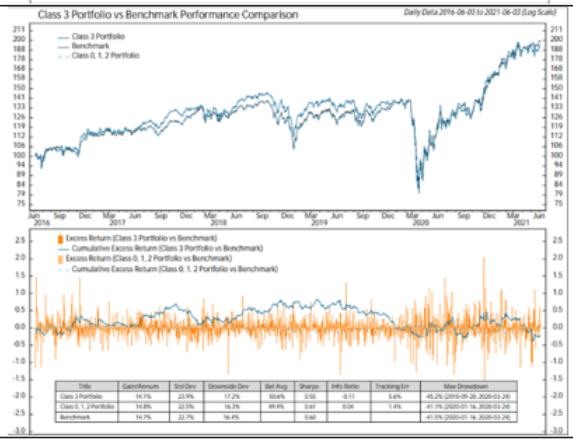
Model	Loss / Accuracy	Characteristics	Performance
DPP Loss: Binary Cross Entropy Optimizer: Adam Epochs: 50 Learning Rate: 0.005 Batch Size: 128 Momentum: 0.1 Label Bins: [0]		Gain +: 33.3% Gain -: -2.2% Sharpe +: 1.47 Sharpe -: -0.14 Accuracy: 35.75% Class*: 59.47%	
ResNet-110 Balanced Loss: Cross Entropy Optimizer: Stochastic Gradient Descent Epochs: 50 Learning Rate: 0.01 Batch Size: 32 Momentum: 0.9 Label Bins: [-1.7,0.3,2.3]		Gain +: 27.4% Gain -: -4.5% Sharpe +: 1.12 Sharpe -: 0.15 Accuracy: 33.15% Class*: 38.97%	
ResNet-110 Unweighted Loss: Cross Entropy Optimizer: Stochastic Gradient Descent Epochs: 50 Learning Rate: 0.007 Batch Size: 64 Momentum: 0.75 Label Bins: [-1.5,-0.5, 0,0.5,1.5]		Gain +: 23.43% Gain -: 2.4% Sharpe +: 0.99 Sharpe -: 0.06 Accuracy: 32.44% Class*: 64%	
Vanilla CNN Loss: Cross Entropy Optimizer: Stochastic Gradient Descent Epochs: 100 Learning Rate: 0.001 Batch Size: 64 Momentum: 0.9 Label Bins: [-1.7,0.3,2.3]		Gain +: 17.2% Gain -: 16.3% Sharpe +: 0.55 Sharpe -: 0.04 Accuracy: 29.08% Class*: 71.13%	

Figure 5. Table of Results

5. Appendix

5.1. Definition of Candlestick Charts

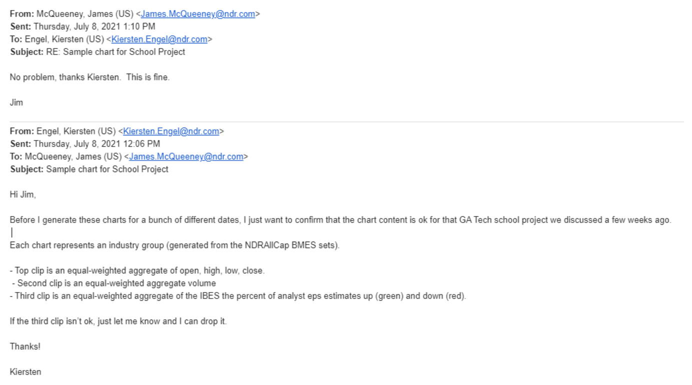
Candlestick charts are used by traders to identify possible trends and patterns in the stock price. Each candlestick shows open, high, low, and closing prices for a given period and shows the range between the highest and lowest trading prices, as well as the range between the opening and closing prices. Hollow candlesticks indicate that the close price was greater than the open, and filled candlesticks indicate that the close was less than the open. The color of the candlestick represents the price movement relative to the prior period's close. Typically, green represents a higher closing price and red represents a lower closing price relative to the prior period.

5.2. Definition of Learning Parameters used

- Epochs – the number of passes through the data set
- Learning Rate – arbiter of step size toward minimal loss
- Momentum – factor controlling rate of change during gradient descent
- Regularization Parameter – coefficient applied in either L1 or L2 regularization
- Burn-In – number of epochs to retard learning while random weights are initialized
- Learning Rate Decay Thresholds – controls learning rate reductions
- Batch Size – number of samples drawn per iteration
- Shuffle – determines if batches are drawn at random

5.3. Compliance E-mail

As requested by Professor Zsolt, please refer to Fig. 6 for e-mail thread regarding compliance with using Ned Davis Research's (NDR) stock charts for this project.



From: McQueeney, James (US) <James.McQueeney@ndr.com>
Sent: Thursday, July 8, 2021 1:10 PM
To: Engel, Kiersten (US) <Kiersten.Engel@ndr.com>
Subject: RE: Sample chart for School Project

No problem, thanks Kiersten. This is fine.

Jim

From: Engel, Kiersten (US) <Kiersten.Engel@ndr.com>
Sent: Thursday, July 8, 2021 12:06 PM
To: McQueeney, James (US) <James.McQueeney@ndr.com>
Subject: Sample chart for School Project

Hi Jim,

Before I generate these charts for a bunch of different dates, I just want to confirm that the chart content is ok for that GA Tech school project we discussed a few weeks ago.

Each chart represents an industry group (generated from the NDRAllCap BMES sets).

- Top clip is an equal-weighted aggregate of open, high, low, close.
- Second clip is an equal-weighted aggregate volume
- Third clip is an equal-weighted aggregate of the IBES the percent of analyst eps estimates up (green) and down (red).

If the third clip isn't ok, just let me know and I can drop it.

Thanks!

Kiersten

Figure 6. Compliance Email

References

[1] João Carapuço, R. Neves, and N. Horta. Reinforcement learning applied to forex trading. *Appl. Soft Comput.*, 73:783–794, 2018. 1

[2] James Chen. Algorithmic trading. Investopedia, 2020. 1

[3] Jun-Hao Chen and Yun-Cheng Tsai. Encoding candlesticks as images for pattern classification using convolutional neural networks. *Financial Innovation*, 2020. 1, 2

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, 2016. 2

[5] Chih-Chieh Hung and Ying-Ju Chen. Dpp: Deep predictor for price movement from candlestick charts. *PLoS ONE*, 2021. 1, 2, 3

[6] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for NAS. *CoRR*, 2019. 4

[7] Nick Lioudis. The importance of diversification. Investopedia, 2021. 2

[8] Omer Berat Sezer and Murat Ozbayoglu. Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, 2018. 1, 2

[9] D. Smirnov and E. M. Nguifo. Time series classification with recurrent neural networks. *3rd ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, 2018. 1

[10] Rick Wayman. The changing role of equity research. Investopedia, 2020. 1

Student Name	Contributed Aspects	Details
Kiersten Engel	Data Procurment, DPP, Testing, Writing	Gathered a variety of different datasets, approval, and compliance, implemented DPP model, ran experiments on the model and contributed in writing final paper.
Christopher Ong	Paper CNN, Custom CNN, Class Imbalance, Testing, Writing	Implemented vanilla CNN from a paper and a derivative custom CNN, ran experiments on both models, contributed in locating and debugging class imbalance, and contributed in writing final paper.
Andrew Pecka	Data Wrangling, Framework Implementation and Testing, Resnet, VGG, Writing	Implemented experiment and testing framework, formatted training and validation data, implemented Resnet and VGG models, ran experiments on both models, and contributed to writing final paper.
Steven Rivadeneira	AlexNet, Resnet, Class Imbalance, Testing, Writing	Implemented AlexNet and Resnet, implemented Resnet Grayscale, ran experiments on both models, contributed in locating and debugging class imbalance, investigating data loaders, and contributed in writing final paper.

Table 1. Contributions of team members.