OZYEGIN UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

CS 401

2021 Fall

SENIOR PROJECT REPORT

Stock Price and Direction Prediction via Deep Attention-Based Convolutional Neural Networks

By
Onur Alaçam, Tuğcan Hoşer, Uygar Kaya, Tuna Tuncer

Supervised By
Assistant Prof. Emre Sefer

Declaration of Own Work Statement/ (Plagiarism Statement)
Hereby I confirm that all this work is original and my own. I have clearly referenced/listed all sources as appropriate and given the sources of all pictures, data etc. that are not my own. I have not made any use of the essay(s) or other work of any other student(s) either past or present, at this or any other educational institution. I also declare that this project has not previously been submitted for assessment in any other course, degree or qualification at this or any other educational institution.

Student Name and Surname:
Onur Alaçam, Tuğcan Hoşer, Uygar Kaya, Tuna Tuncer

Signature:

Place, Date:
Istanbul, Turkey - 09.01.2022

Contents

Stock Price and Direction Prediction via Deep Attention-Based Convolutional Neural Networks

Onur Alaçam, Tuğcan Hoşer, Uygar Kaya, Tuna Tuncer

2021 Fall

Abstract

Since stock prices are time series data, most academic studies have focused on time series forecasting in machine learning models. Unfortunately, these models do not perform as well as they need to. In this project, we designed a novel model that predicts Stock Price and Direction with the help of Deep Attention-based Convolutional Neural Networks based on image processing properties using 2-D images after converting the 1-D time-series signal images to 2-D images with the help of the technical indicator. We tested the model we have built and compare its performance to state-of-the-art models at the end of this project. If the results are as expected, we aim to put the model to the test on a major crypto exchange platform such as Binance. We start by gathering price and technical indicator data for a variety of cryptocurrencies. Then, utilizing this information, we constructed a two-dimensional image with the stock price as one dimension and technical indicators as the other. We obtain our model by training deep attention-based convolutional neural networks with these images. The implementation part of the project is done in Python Programming Language.

1 Introduction

This report shows how to do machine learning in finance in general through images. We've known for many years that the convolutional neural network has dominated this field, but the recently released Transformer-based models, some of which are Vision Transformer, outperform CNN in most cases. ConvMixer, which was released in 2021, was understood to perform better than both Transformer-based models and CNN models in most cases. Our aim is to compare these models in the field of finance and show which one is the most successful in this field.

The background will be covered in the second chapter, where we will discuss the approaches, tools, and technology that we have used in our project.

The third chapter will discuss our problem statement and the problem we are attempting to solve. The fourth chapter will describe how we tackled the problem and how we came up with answers. In the fifth chapter, we examined our findings and what we were able to accomplish, and what we were unable to accomplish. We've mentioned flaws we found when looking at the outcomes in the findings, as well as how we might fix them in the future. We compare our technique and model to other comparable works in the sixth chapter. The seventh chapter contains our conclusion as well as our recommendations.

2 Background

In this project, many different techniques, methods, tools, data and indicators were used.

PyTorch: PyTorch is an open source machine learning library based on the Torch library. The CNN-TA network model was implemented using this library.

Nvidia CUDA: Nvidia Cuda is a set of development tools that enable algorithms to run on the GPU. We used Nvidia Cuda with PyTorch to train the model faster.

S&P: The S&P 500 includes major American companies. It covers about 75% of the American stock market. We used data from the last 20 years of S&P.

Yahoo Finance: Yahoo Finance provides financial news, data and commentary including stock quotes, press releases, financial reports, and original content. We downloaded the S&P 500 dataset from here. In addition, we will consider companies individually in the future and we plan to download their data from Yahoo Finance.

Pandas: Pandas is a software library written in the Python programming language for data manipulation and analysis. We processed numerical data and tables with using this library.

Numpy: NumPy is a library for the Python programming language that supports large, multidimensional arrays and matrices, adding high-level mathematical functions to operate on these arrays.

TA-Lib: TA-Lib is a library that contains 150 different indicators, especially used for trading algorithms. We also used 11 different indicators thanks to this library.

TensorFlow: TensorFlow is a free & open source software library for machine learning. The architectures in the articles we compared used TensorFlow because they were implemented in TensorFlow.

Keras: Keras is an open source neural network library written in Python. Our architectures are implemented using Keras.

Sklearn: Scikit-learn is a free software machine learning library for the Python programming language. It is used to separate data, scale, and measure metrics as a result of the architecture.

Matplotlib: Matplotlib is a plotting library for the Python programming language and numerical math extension NumPy. We used it to visualize the results from our data.

Indicators

We got the data of S&P 500 from Yahoo Finance from 2001 to 2021. We have turned these data into pictures by adhering to 11 economic technical indicators. In this part, we will talk about the indicators we have used.

RSI: Relative Strength Index
The Relative Strength Index (RSI) is an indicator that provides predictions about the direction of the short and medium-term trend calculated by comparing the closing values of the relevant period with the previous closing values of the period.

$$RSI = 100 - \frac{100}{1 + \frac{averagegain}{averageloss}}$$

Figure 1: RSI Formula

WMA: Weighted Moving Average
A Weighted Moving Average puts more weight on recent data and less on past data. This is done by multiplying each bar's price by a weighting factor. Because of its unique calculation, WMA will follow prices more closely than a corresponding Simple Moving Average.

$$WMA(M, n) = \frac{Sum\ of\ Weighted\ Averages}{Sum\ of\ Weight}$$

Figure 2: WSI Formula

EMA: Exponential Moving Average
EMA, which stands for Exponential Moving Average, is used as an indicator of moving averages. Through the EMA, traders can watch exponential moving averages on the charts. By giving weight to last minute data such as EMA closing prices, the time that can progress with the weights assigned to the relevant data is based on the exponentially decreasing process.

$$(M(t) - EMA(M, t-1, \tau)).\frac{2}{\tau+1} + EMA(M, t-1, \tau)$$

Figure 3: EMA Formula

SMA: Simple Moving Average
Simple Moving Average (SMA) is the simplest form of moving averages. It is obtained by dividing the total of data by the number of data.

$$SMA(M, n) = \sum_{k=a+1}^{a+n} \frac{M(k)}{n}$$

Figure 4: SMA Formula

ROC: Rate of Change
The Rate of Change Indicator (ROC) is a kind of momentum oscillator. Calculates the rate of price change between periods.

$$RoC = \frac{(Latest\ Close - Previous\ Close)}{(Previous\ Close)} * 100$$

Figure 5: ROC Formula

CMO: Chande Momentum Oscilator Indicator
The Chande Momentum Oscillator (CMO) is calculated by dividing the sum of the momentum on the up days and the sum of the momentum on the down days by dividing the sum of the momentum on the up days by the sum of the momentum on the down days and multiplying by 100 to make a percentage.

$$CMO = 100 * \frac{(S_u - S_d)}{(S_u + S_d)}$$

Figure 6: CMO Formula

S_U: *Total momentum of the up days for the analysis period*
S_D: *Total momentum of the down days for the analysis period*

CCI: Commodity Channel Index
Commodity Channel Index (CCI) is an indicator that compares current prices and the average price over a period of time.

$$CCI = \frac{Typical\ Price - 20\ Period\ SMA\ of\ TP}{.015 * Mean\ Deviation}$$

$$TypicalPrice(TP) = \frac{High + Low + Close}{3}$$

Figure 7: CCI Formula

PPO: Percentage Price Oscillator
The Percentage Price Oscillator (PPO) is a technical momentum indicator that displays the relationship between two moving averages in percentage terms. The moving averages are the 26-period and 12-period exponential moving average (EMA).

$$PPO = \frac{(12\ Day\ EMA - 26\ Day\ EMA)}{26\ Day\ EMA} * 100$$

$$Signal\ Line: 9\ Day\ EMA\ of\ PPO$$

Figure 8: PPO Formula

TEMA: Triple Exponential Moving Average
Triple Exponential Moving Average (TEMA) is a type of EMA indicator that provides the reduction of minor price fluctuations and filters out volatility.

$$(3 * EMA - 3 * EMA(EMA)) + EMA(EMA(EMA))$$

Figure 9: TEMA Formula

WILLR: Williams
Williams %R, also known as the Williams Percent Range, is a type of momentum indicator that moves between 0 and -100 and measures overbought and oversold levels.

$$R = \frac{max(high) - close}{max(high) - min(low)} * -100$$

Figure 10: WILLR Formula

MACD: Moving Average Convergence and Divergence
The Moving Average Convergence and Divergence (MACD) indicator is a technical indicator that displays how stock values are trending.

$$MACD\ Line : (12\ Day\ EMA - 26\ Day\ EMA)$$
$$Signal\ Line : 9\ Day\ EMA\ of\ MACD\ Line$$

Figure 11: MACD Formula

3 Problem Statement

It is quite common to convert the image into neural networks and make predictions from it. In our case, it is based on taking financial data and converting them into images with the help of indicators and making predictions on these images. This is a method that has been used before, but our mission here is to show that with whichever model we make these predictions, we will achieve a more successful result.

We decided to implement this project with our supervisor Assistant Prof. Emre Sefer to solve this problem. Afterward, we thoroughly researched the issue and discussed the improvements we could make.
While creating the images, it is necessary to choose the technical indicators well and to create the image according to the connection of these indicators with each other. In addition, if the number of data we pull increases, the images will increase, and if we use CPU when we want to train them, it will take a lot of time at high epoch values, so we plan to use Nvidia CUDA.

4 Solution Approach

4.1 Dataset Preparation

First and foremost, we had to decide whether we would proceed with our work on stocks or cryptocurrencies.

Due to the high volatility of cryptocurrencies, the accuracy of the predictions to be made with the models could be lower. Because, in today's cryptocurrency market, which can change rapidly with the tweets of celebrities, our work could be more challenging. However, once our model for stocks in CS402 is ready, we want to train the same models on cryptocurrencies and compare the outcomes.

In the paper *Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach* [2], it was reported that models with ETF data outperformed the models with Dow30 data in terms of accuracy. That's why we chose to use the S&P500. We also wanted to include the 2008 economic crisis so that our dataset does not exhibit an entirely increasing market trend. As a result, we generated our dataset utilizing data from 2001-10-11 to 2021-11-11.

After we created our dataset, we converted it into images to use in our architectures. We constructed our images by using 11 of the 15 indicators described in the paper *Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach* [2]. As a result, our photos are 11x11 in size. We also arranged the order in which the indicators will be placed on the image by looking at the correlation values between the indicators. Because different ordering will lead to different images this might have affected the performance of models. Each image was created using 11 days of historical data from 11 indicators. As an example, if we are on day t and wish to anticipate what will happen on day t+1, the image will be constructed utilizing indicators from day t through day t-10.

Furthermore, we had to decide on a threshold value for labeling the dataset. Figure 19 depicts the algorithm for labeling photos using the threshold. In short, we decided to label an image as buy if the price increased more than the set threshold, sell if the price dropped more than the threshold, and hold if not both. As presented in Table 1, We chose 0.005 as the threshold value, because we believe it's the most optimal value ensuring that the dataset was both balanced for training the models and both reasonable for trading. Finally, to improve the accuracy of the models we will train, we scaled our data such that the standard deviation is one and the mean is zero.

Table 1: Label Distribution by Different Threshold Values

Threshold	Buy	Hold	Sell	Buy %	Hold %	Sell %
0.003	1814	1686	1460	36.6%	34.0%	29.4%
0.004	1574	2113	1273	31.7%	42.6%	25.7%
0.005	**1379**	**2448**	**1133**	**27.8%**	**49.4%**	**22.8%**
0.01	661	3668	631	13.3%	74.0%	12.7%

4.2 Implementing Different Architectures & Training the Models

The results from part 1 of how our data is retrieved, processed, and transformed heavily influences this section. A total of three different architectures were used in this project. These architectures are Convolutional Neural Networks (CNN), ConvMixer and Vision Transformer, respectively. By training the data from Part 1 on these three different architectures, the results obtained for each architecture were compared.

Different hyperparameters are selected for each different architecture.

Convolutional Neural Networks

In our proposed CNN analysis phase, as can be seen in Figure, nine layers are used. These are listed as follows: input layer (11x11), two convolutional layers (11x11x32, 11x11x64), a max pooling (5x5x64), two dropout (0.25, 0.50), fully connected layers (128), and an output layer (3).

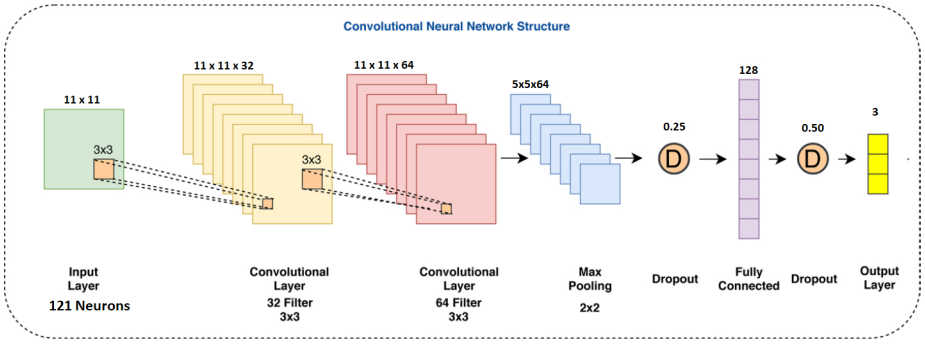


Figure 12: Convolutional Neural Network Structure

By using an image filter in the convolution layer, we enlarge the channel of the image with stride and padding operations. The filter is generally chosen as 3x3, 5x5 or 7x7. We thought that as the size of the image decreases, the smaller the filter will yield better results as it will enable the image to be scanned more. That's why we chose the filter 3x3. In this way, when we apply 2 convolution layers on the 11x11 image, we get an 11x11x64 image. Then we pass it through the max pooling layer, and we get a 5x5x64 image. Dropout layers are added to prevent overfitting. By reducing the final image to one dimension, we get 128 fully connected neural networks. Then we transform these neural networks into our class labels **Buy**, **Hold** & **Sell** outputs on the output layer.

We used PyTorch while implementing CNN. We selected the hyperparameters based on the best result by training the images with different values. We chose the epoch as 200, the learning rate as 0.001 and the batch size as 64.

ConvMixer

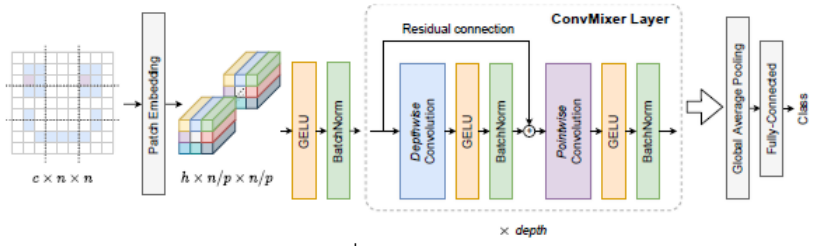


Figure 13: ConvMixer Structure

In ConvMixer, we first split the image into patches by applying patch embedding. Then, we pass each patch through the GELU [*Equation 1*] and Batch Normalization [*Equation 2*] layers separately as an activation function. We pass the resulting patches through the ConvMixer layer. The ConvMixer layer contains Depthwise Convolution, activation function, Pointwise Convolution and activation function, respectively. Depthwise Convolution is a sort of convolution in which each input channel receives a single convolutional filter and Pointwise Convolution is a form of convolution that uses a 1x1 kernel, which iterates through each point.

We used tensorflow and keras while implementing ConvMixer. In order to select the hyperparameters optimally, we trained the images by giving different numbers of epoch, batch size, patch size, kernel size and learning rate. As a result, we gave the 200 epoch, the batch size 128, the patch size 1, the kernel size 3 and the learning rate 0.01.

Equation 1: $GELU(x) = xP(X \leq x) = x\phi(x)$

Equation 2: $y = ((x - E[x])/\sqrt{Var[x] + \epsilon}) * \gamma + \beta$

Vision Transformer

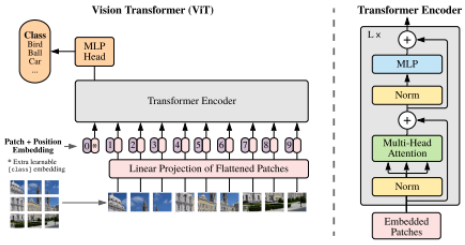


Figure 14: Vision Transformer Structure

In the Visual Transformer model, the normal transformer model is adapted to the images. 2D images are converted into sequence of flattened 2D patches using patch + position embedding. Then these patches are given to the transformer encoder according to their positions. Transformer Encoder contains 3 layers, these layers are Normalization, Multi-Head Attention, Normalization. Multi-Head Attention is an attention mechanism module that runs through an attention mechanism multiple times in parallel. After the patches pass through these layers, a classification model, MLP, has been added for the classification to take place. We used tensorflow and keras while implementing Vision Transformer. We gave hyperparameters epoch 200, batch size 128, patch size 2 and learning rate 0.001.

4.3 Technical, Operational & Financial Feasibility

Technical Feasibility
We could not use Nvidia CUDA on some architectures since these architectures do not support the current version of TensorFlow, in this case, we ran the code on the CPU instead of the GPU, which increased the code's runtime, but when we used Nvidia CUDA in the CNN architecture, the code's runtime decreased by 1 quarter.

Financial Feasibility
We don't have any other cost without needing a better performing and equipped computer to train our model.

4.4 Knowledge & Skill Set

Different skill sets from various courses were used in this project. While the courses mentioned here had the most impact on our project, all the CS courses we've reviewed so far have given us new perspectives in implementing this project.

CS 101: The basics of programming were learned and helped to use the basic knowledge in this project.

CS 320: Helped us to design and write the project reports accurately.

CS 447: Helped us to request Yahoo Finance data.

CS 452: Helped us to create and pre-process the dataset as well as apply the classification model.

CS 454: Helped us to understand the neural network logic and its pros and cons.

MATH 211: Helped us to understand the matrices in the structure of the images.

MATH 217: Help us to compare statistics in results.

4.5 Engineering Standards

- Style Guide for Python Code
- Data Format: NumPy Array
- HTTP Protocols: GET

5 Results and Discussion

We attempted to obtain results with the ConvMixer and Vision Transformer architectures using the dataset we generated. These results, as well as a comparison to the CNN-TA article's results, are shown below.

To begin with, one of the most significant findings was that the distribution of predictions in the ConvMixer architecture was more diversified than that of the Vision Transformer architecture. The ConvMixer model assessed the buy points more accurately than the Vision Transformer model, as demonstrated by the confusion matrices in Table 1 and Table 3. While the Vision Transformer model performed better in hold predictions, both models accurately identified 18 pictures in sell. Because 49.8 percent of our dataset is labeled hold, a model is more likely to predict hold correctly. Therefore, the fact that the ConvMixer model has a better prediction percentage than the Vision Transformer for buy labeled images may indicate that the ConvMixer model is more resistant to imbalanced datasets.

Table 2: Confusion Matrix of ConvMixer

		Predicted		
		Buy	Hold	Sell
Actual	Buy	40	73	22
	Hold	30	204	14
	Sell	29	66	18

Table 3: Classification Report of ConvMixer

Total Accuracy: 0.53			
	Buy	Hold	Sell
Recall	0.30	0.82	0.16
Precision	0.40	0.59	0.33
F1 Score	0.34	0.69	0.22

Table 4: Confusion Matrix of Vision Transformer

		Predicted		
		Buy	Hold	Sell
Actual	Buy	12	103	20
	Hold	1	234	13
	Sell	8	87	18

Table 5: Classification Report of Vision Transformer

Total Accuracy: 0.53			
	Buy	Hold	Sell
Recall	0.09	0.94	0.16
Precision	0.57	0.55	0.35
F1 Score	0.15	0.70	0.22

In Table 5, it is seen that 89.9 percent of the test datasets are labeled as hold. This demonstrates that the dataset used for CNN-TA is much more imbalanced than our dataset. In addition, while the CNN-TA model was evaluated with 33714 test data, the test data we used to evaluate our ConvMixer and Vision Transformer models consisted of 496 images. This may explain why the total accuracy of our models is lower than that of CNN-TA and why both models are subject to overfitting very quickly.

Table 6: Confusion Matrix of CNN-TA

		Predicted		
		Buy	Hold	Sell
Actual	Buy	1215	478	0
	Hold	5180	18629	6498
	Sell	0	587	1127

Table 7: Classification Report of CNN-TA

Total Accuracy: 0.62			
	Buy	Hold	Sell
Recall	0.72	0.61	0.66
Precision	0.19	0.95	0.15
F1 Score	0.30	0.75	0.24

In addition, as mentioned above, the size of the dataset we prepared was considerably smaller than the size of the dataset prepared for the CNN-TA model.

This may have caused our models to experience rapid overfitting. In order to minimize overfitting, we plan to increase the number of data points and perform cross-validation in the future.

Model Accuracy & Loss Graph of ConvMixer

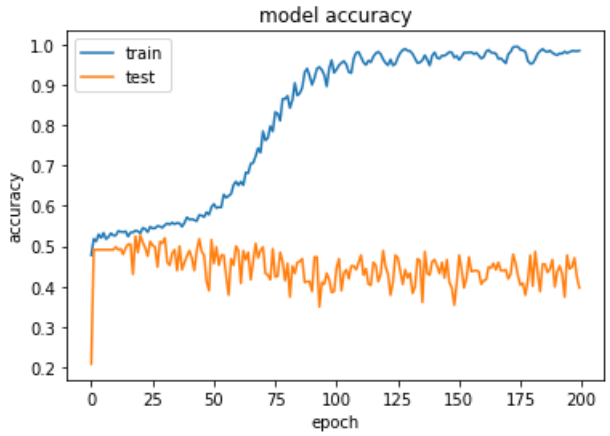


Figure 15: Train - Test Accuracy Graph

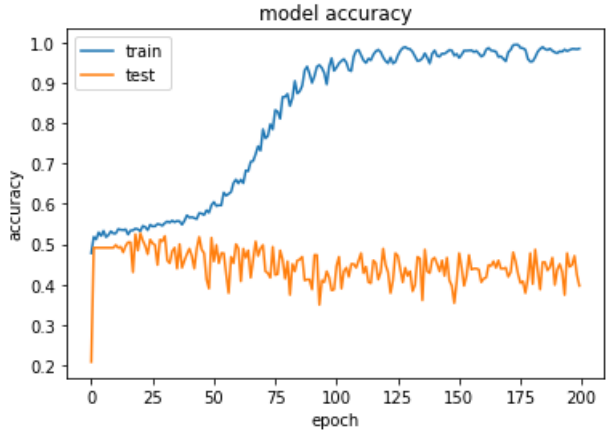


Figure 16: Train - Test Loss Graph

Model Accuracy & Loss Graph of Vision Transform

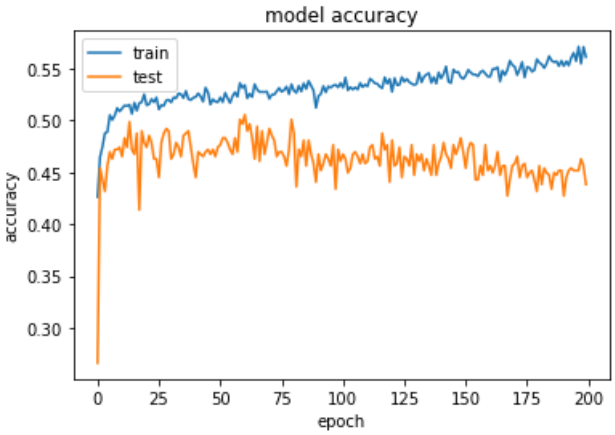


Figure 17: Train - Test Accuracy Graph

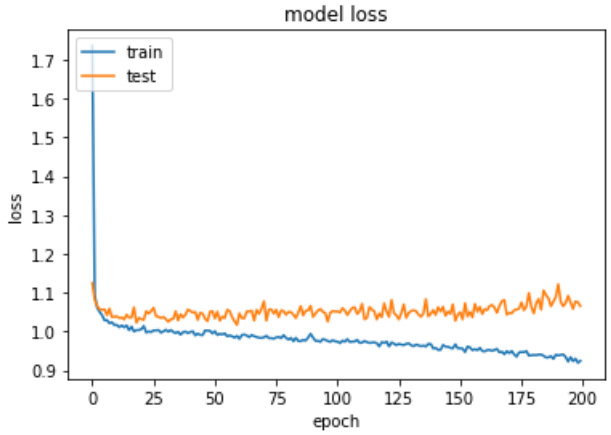


Figure 18: Train - Test Loss Graph

We also tried to implement the CNN-TA architecture using PyTorch and train it with our own dataset, but we could not obtain results that we can graph. Within the scope of CS402, we plan to add this to the models we develop and compare.

Moreover, all models were trained on an Intel i7 4510-U CPU. That's why we had to limit the amount of data we collected.

As the project progresses, we intend to train our models with GPUs on a platform such as Azure, AWS Cloud, or Google Cloud.

6 Related Work

In many academic studies, machine learning models were used when stock prediction was made because stock prices are time series data. Nevertheless, the machine learning models used for stock-price prediction were not performing excellently. Thus, better-performing models were required for stock prices.

In recent years, forecast models based on deep learning have emerged as the best performers in financial forecasts. Moreover, many academic and sectoral studies have been conducted using different deep-learning models. The article named "Attention Augmented Convolutional Networks" [1] implemented by the Google Brain team under the umbrella of Google AI, the article named "Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach" [2], the article named "Patches Are All You Need?" [3] or the article named "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" [4] can be given examples of these studies. Our approach in the senior project is analogous to the "Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach" article, but while using Convolutional Neural Network (CNN) in this article, we use the ConvMixer architecture, introduced in 2021, which is thought to perform better according to research's.

Although CNNs have dominated computer vision tasks for many years, recent research has shown that Transformer-based models, most notably the Vision Transformer (ViT), can outperform CNN in some cases.[3] ConvMixer an extremely straightforward model that is similar to the ViT; hence, we are using this architecture in our senior project.

7 Conclusion and Future Work

In this section, we provide a brief summary of the senior project and we talk about the future work of the project.

7.1 Conclusion

To summarize our senior project, we use the ConvMixer model that predicts Stock Price and Direction with the help of Deep Attention Based Convolutional Neural Networks after transforming 1D time-series signal images into 2D time-series signals with the help of technical indicators. We think that the last version of senior project will have a consequential economic impact.

Companies or investors can buy their stocks in a more optimized way with the last version of our senior project, and as a result, investors can use the money that normally is melt away by their manual trading techniques on different opportunities. Furthermore, We think that our project will have an optimistic social impact because we believe that investors can be mirthful with the right investments they make. Also, After some thought about what ethical issues we might encounter in our senior project, we realized that there might be a problem. If the model we will apply constructs an inaccurate prediction, we can mislead the investors & companies and make an erroneous investment decision.

7.2 Future Work

• Finding the best hyperparameters of our model using sites like Weights & Biases. [5]
• Utilizing the Cross-Validation to make the performance of the model more accurate.
• On a high-dimensional scale, the assessment measures and their correlation may be examined.
• Running models on different Datasets & Technical Indicators and reporting the results.
• Testing the model on a major crypto exchange platform like Binance after doing the work above.
• Finally, implementing a new project that makes an automatic trading bot for cryptocurrencies using this model.

Acknowledgements

For his support and advice throughout our senior project, we would like to express our thanks and gratitude to our advisor Assistant Prof. Emre Sefer.

References

[1] Bello, I., Zoph, B., Vaswani, A., Shlens, J., amp; Le, Q. V. (2020, September 9). Attention augmented convolutional networks. arXiv.org. Retrieved January 4, 2022, from https://arxiv.org/abs/1904.09925

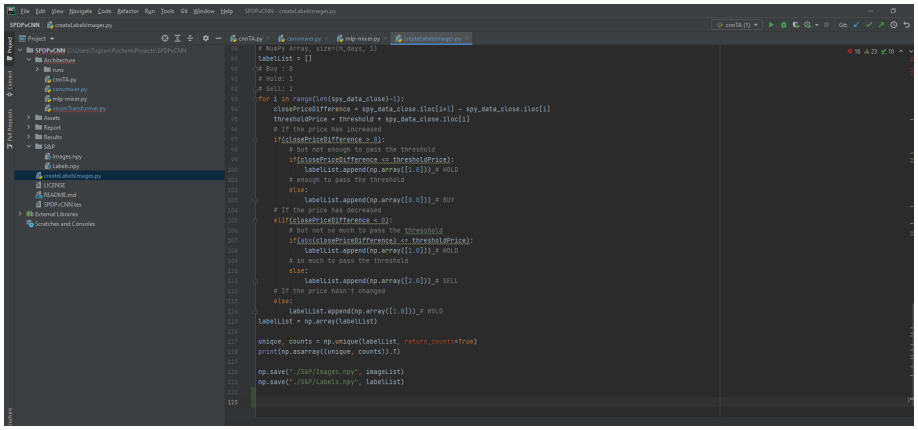
[2] Sezer, O. B., amp; Ozbayoglu, A. M. (2018, April 27). Algorithmic financial trading with deep convolutional neural networks: Time Series to Image Conversion Approach. Applied Soft Computing. Retrieved January 4, 2022, from https://www.sciencedirect.com/science/article/abs/pii/S1568494618302151

[3] Anonymous. (2021, November 23). Patches are all you need? OpenReview. Retrieved January 5, 2022, from https://openreview.net/forum?id=TVHS5Y4dNvM

[4] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., amp; Houlsby, N. (2021, June 3). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv.org. Retrieved January 5, 2022, from https://arxiv.org/abs/2010.11929

[5] Weights amp; Biases – developer tools for ML. Weights amp; Biases – Developer tools for ML. (n.d.). Retrieved January 5, 2022, from https://wandb.ai/site

Appendix

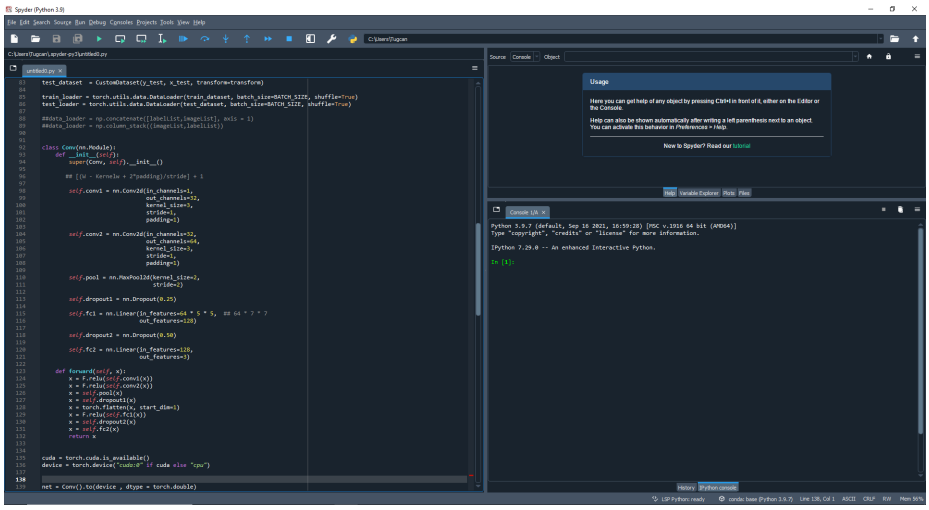


```
# NumPy Array, size=(n_days, 1)
labelList = []
# Buy : 0
# Hold: 1
# Sell: 2
for i in range(len(spy_data_close)-1):
    closePriceDifference = spy_data_close.iloc[i+1] - spy_data_close.iloc[i]
    thresholdPrice = threshold * spy_data_close.iloc[i]
    # If the price has increased
    if(closePriceDifference > 0):
        # but not enough to pass the threshold
        if(closePriceDifference <= thresholdPrice):
            labelList.append(np.array([1.0]))_# HOLD
        # enough to pass the threshold
        else:
            labelList.append(np.array([0.0]))_# BUY
    # If the price has decreased
    elif(closePriceDifference < 0):
        # but not so much to pass the threshold
        if(abs(closePriceDifference) <= thresholdPrice):
            labelList.append(np.array([1.0]))_# HOLD
        # so much to pass the threshold
        else:
            labelList.append(np.array([2.0]))_# SELL
    # If the price hasn't changed
    else:
        labelList.append(np.array([1.0]))_# HOLD
labelList = np.array(labelList)

unique, counts = np.unique(labelList, return_counts=True)
print(np.asarray((unique, counts)).T)

np.save("./S&P/Images.npy", imageList)
np.save("./S&P/Labels.npy", labelList)
```

Figure 19: Algorithm of Labeling



```
test_dataset  = CustomDataset(y_test, x_test, transform=transforms)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=True)

##data_loader = np.concatenate([labelList,imageList], axis = 1)
##data_loader = np.column_stack((imageList,labelList))


class Conv(nn.Module):
    def __init__(self):
        super(Conv, self).__init__()

        ## [(W - KernelSize + 2*padding)/stride] + 1

        self.conv1 = nn.Conv2d(in_channels=1,
                               out_channels=32,
                               kernel_size=5,
                               stride=1,
                               padding=1)

        self.conv2 = nn.Conv2d(in_channels=32,
                               out_channels=64,
                               kernel_size=5,
                               stride=1,
                               padding=1)

        self.pool = nn.MaxPool2d(kernel_size=2,
                                 stride=2)

        self.dropout1 = nn.Dropout(0.25)

        self.fc1 = nn.Linear(in_features=64 * 5 * 5,  ## 64 * 7 * 7
                             out_features=128)

        self.dropout2 = nn.Dropout(0.50)

        self.fc2 = nn.Linear(in_features=128,
                             out_features=3)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = self.dropout1(x)
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.fc1(x))
        x = self.dropout2(x)
        x = self.fc2(x)
        return x


cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if cuda else "cpu")

net = Conv().to(device , dtype = torch.double)
```

Figure 20: Algorithm of Convolutional Neural Network

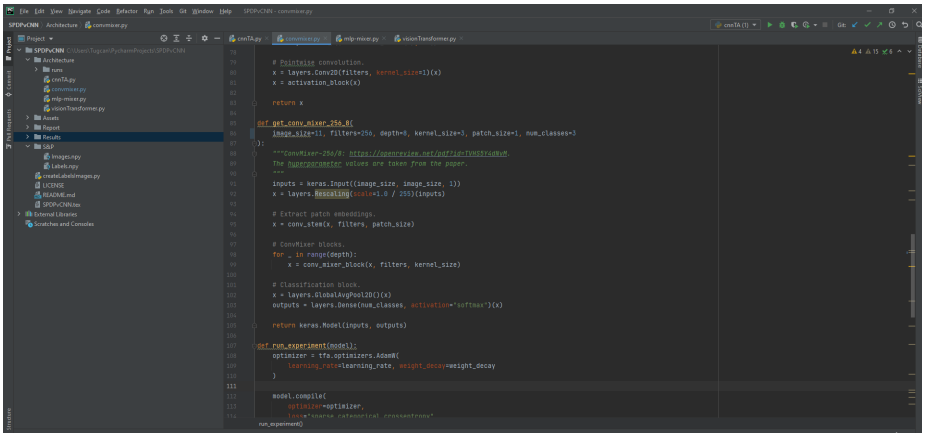


Figure 21: Algorithm of ConvMixer

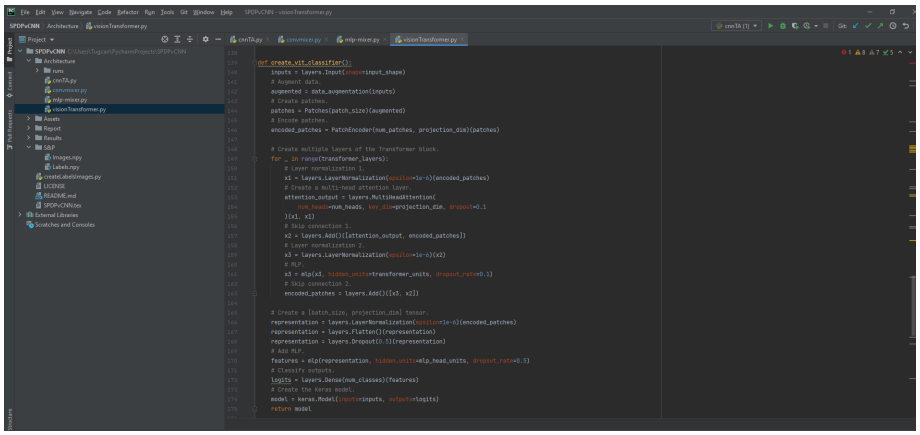


Figure 22: Algorithm of Vision Transformer