OZYEGIN UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

CS 402

2022 Spring

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 - 01.06.2022

Contents

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

Onur Alaçam, Tuğcan Hoşer, Uygar Kaya, Tuna Tuncer

2022 Spring

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 excepted. In this project, we converted the 1-D time-series data to 2-D images with the help of the technical indicators. Then, using these images, we compared a few state-of-the-art models in the field of computer vision such as ConvMixer, Vision Transformer, and the CNN-TA model, which is also used for predicting stock prices in the literature before. We investigated the effects of self-attention and patch embedding in the finance area. Implementations of all models are done in Tensorflow - Keras framework in the Python Programming Language.

1 Introduction

In the last decade, Machine Learning architectures have started to become popular in finance as in other fields, however, most of these architectures do not give the expected performance in the time series data with current technologies. Thus, Deep Learning architectures have recently attracted more attention. Specifically, in recent years, with the discovery of the attention mechanism, there have been crucial developments in the field of Computer Vision, hence we decided to apply this potential to the field of finance. In order to take advantage of Computer Vision models, we had to convert the time series data into images.

In our research, we use the Vision Transformer architecture, which uses patch embedding and applies the attention mechanism to transformer models, and the ConvMixer architecture, which integrates the patch embeddings structure into Convolutional Neural Networks. Also, in our project, we have implemented the main architecture of the article "Algorithmic Financial Trading with Deep **Convolutional Neural Networks**"[2], in order to compare the results we will get from these architectures.

It's a fact for many years that the convolutional neural networks has dominated the computer vision field, but the Vision Transformer claims to outperform CNNs in most cases. It is believed that patch embedding, and self-attention mechanisms, and the complex transformer structure are crucial parts of the success of Vision Transformer. On the other hand, ConvMixer, which was released in January 2022, is claiming to perform better than both Vision Transformer and other CNN models. Moreover, the authors of the ConvMixer, claim that the success of the Vision Transformer is the result of the patch embedding mechanism rather than the complex transformer architecture of the model. Our aim is to show the effects of both patch embedding and self-attention mechanisms on the finance field by comparing these architectures.

As we mentioned above, we used technical indicators to obtain images from time series data for the project, each row of our images represents a different day, while each column represents a different technical indicator value. Since we used 67 different technical indicators in total, the size of our images became 67x67. By putting these images into the 3 different architectures, ConvMixer, Convolutional Neural Network, and Vision Transformer, we compared each of our models in terms of Computational Model Performance and Financial Evaluation.

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

The tools, technologies and various techniques used in our project are explained in detail below.

2.1 Technologies

Nvidia CUDA: Nvidia Cuda is a set of development tools that enable algorithms to run on the GPU. We used Nvidia Cuda with Keras to train our models faster on GPU rather than CPU.

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.

2.2 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.

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

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

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

Figure 6: CMO Formula

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

2.3 ETFs

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.

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 21 years of S&P.

XLF: The Financial Select Sector SPDR Fund (XLF) is a diversified fund with many blue-chip companies in its portfolio. It covers a variety of industries within the financial sector, has generated strong returns, has high liquidity, and is a medium-cost investment.

XLU: The Utilities Select Sector SPDR® Fund seeks to provide investment results that, before expenses, correspond generally to the price and yield performance of the Utilities Select Sector Index.

Seeks to provide precise exposure to companies from the electric utility, water utility, multi-utility, independent power and renewable electricity producers, and gas utility industries.

QQQ: The Invesco QQQ ETF is an exchange-traded fund (ETF) that tracks the Nasdaq 100 Index. Because it passively follows the index, the QQQ share price goes up and down along with the tech-heavy Nasdaq 100

XLP: The Consumer Essentials Industry SPDR Fund aims to provide investment results that correspond to the price and yield performance of the Consumer Essentials Industry Index, prior to expenditure. The Index aims to provide an effective representation of the SP 500 Index's core consumer products sector.

EWZ: EWZ tracks the MSCI Brazil 25/50 Index, which provides a broad-based measure of the performance of the Brazilian equity market.7 The ETF provides exposure to midcap and large-cap companies based in Brazil. Its largest allocations are in the materials, financial, energy sectors of the Brazilian economy.

EWH: Shares, Inc. - iShares MSCI Hong Kong ETF is an exchange traded fund launched by BlackRock, Inc. It is managed by BlackRock Fund Advisors. It invests in public equity markets of Hong Kong. The fund invests in stocks of companies operating across diversified sectors. It invests in growth and value stocks of companies across diversified market capitalization.

XLY: The Consumer Discretionary Select Sector SPDR® Fund seeks to provide investment results that, before expenses, correspond generally to the price and yield performance of the Consumer Discretionary Select Sector Index. Seeks to provide precise exposure to companies in retail (specialty, multiline, internet and direct marketing); hotels, restaurants and leisure; textiles, apparel and luxury goods; household durables; automobiles; auto components; distributors; leisure products; and diversified consumer services.

XLE: One of the most popular exchange-traded funds (ETFs) used by retail traders seeking exposure to energy is the Energy Select Sector SPDR Fund, which tracks the S&P Energy Select Sector Index.

3 Problem Statement

3.1 Project Scope

Our problem was that today's machine learning models were inadequately effective in the area of finance since the majority of today's models struggled with time-series data because time introduces a new piece of information that must be monitored. In addition, many successful studies have been carried out in the world of deep learning in recent years, and these studies have led to successful results in the literature. Our goal was to see if advanced deep learning techniques like patch embedding and self-attention, which have just recently been introduced, can achieve the anticipated results when applied to finance.

The majority of the strategies we employ in our project may be found in the article Algorithmic Financial Trading. In addition to this research, we included the following in our project: While the AFT paper produced findings using a single form of CNN, we acquired results with two additional freshly designed alternative architectures and compared them to the results in the article. Also, we explored how various topologies of these architectures would operate in our problem. Moreover, while the AFT article employed 15 technical indicators, we added 67 technical indications to increase the size of our picture.

3.2 Engineering Problem

Although finance problems often deal with time-series data, we have transformed our problem into an image classification problem rather than a time-series forecasting problem by turning time-series data into images. One of the issues we encountered was the proper evaluation of our models. Because our dataset was distributed in an imbalanced manner in one of the configurations we employed, we had to seek several metrics in order to appropriately evaluate the success of our model. Furthermore, when we couldn't obtain the results we wanted with 11x11 images, we decided to increase the number of technical indicators and make our images 67x67 because the attention mechanism used in the vision transformer worked on the logic of focusing on the key elements of the large image. As a result, we sought to make our image as large as possible.

3.3 Assumptions and Constraints

Since only technical indicators are used in the data creation process in this project, we assume that stock prices can only be modeled with technical indicators. However, as is well known, today's stock values are heavily influenced by a variety of events such as wars, legislation, and tweets. As a result, cryptocurrency volatility is substantially higher than stock market index volatility. This is why we decided to focus our analysis on stocks rather than cryptocurrency.

We chose to conduct this project with our supervisor, Assistant Prof. Emre Sefer. Following that, we carried out extensive research into the problem and discussed potential improvements.

We calculated the correlation of all technical indicators according to RSI when constructing the images since the order in which they are arranged would impact the model's learning. Following that, we arranged the indicators in this correlation order.

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. That's why we decided to make predictions on stocks.
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 decided to use ETFs from this paper. 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 2022-04-15.

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]. Apart from the article, we used 56 more indicators and our total number of indicators became 67. As a result, our photos are 67x67 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 67 days of historical data from 67 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-66.

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. In the labeling process of our images, also as presented in Table 1, we decided to construct two separate label sets utilizing two different thresholds. We selected 0.0038 as the threshold value since we wanted to spread the data as evenly as possible. Thus we were able to generate predictions based on a balanced dataset. On the other hand, we picked 0.01 as our threshold with financial reasons in mind since we concluded that if the stock's value increases by 1% in a day, it's a good idea to purchase it ahead of time, and if it decreases by 1%, it's a good idea to sell it beforehand. Furthermore, a value of 0.0038 lays the foundation for a higher-frequency trading strategy, whereas the value of 0.01 generates more hold-centric predictions and is akin to a buy&hold strategy. 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 %
0.0038	13174	11461	11437	36.5%	31.8%	31.7%
0.01	6547	23486	6039	18.1%	65.1%	16.8%

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-TA), 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.

In this senior peoject, 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 (67x67), two convolutional layers (67x67x32, 67x67x64), a max pooling (33x33x64), two dropout (0.25, 0.50), fully connected layers (128), and an output layer (3).

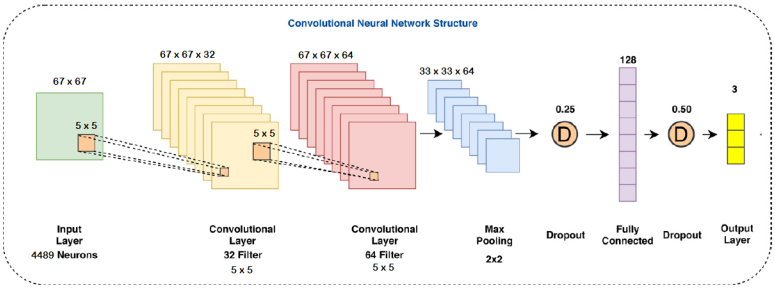

Figure 12: Convolutional Neural Network Structure

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 opted not to adjust the kernel size since Sezer and Ozbayoglu (2018) used a kernel size of 5x5 in their models. In this way, when we apply 2 convolution layers on the 67x67 image, we get an 67x67x64 image. Then we pass it through the max pooling layer, and we get a 33x33x64 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 Keras while implementing CNN. Although we have two different labelsets, we kept all our hyperparameter values same during the training of the model for two different threshold values. We also utilized Ada Delta, which was developed by Zeiler Matthew (2012) and was used in the Algorithmic Financial Trading paper, as the optimizer.

ConvMixer

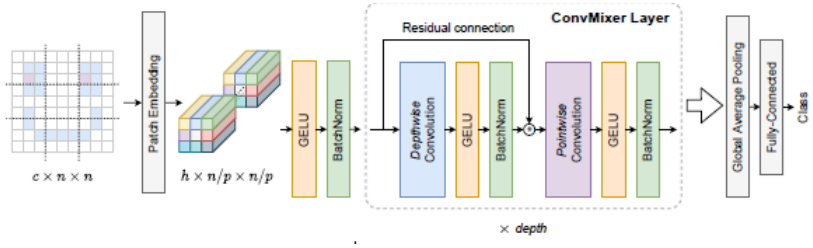

Figure 13: ConvMixer Structure

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 trained a 256x8 model for 166 epochs using kernel size, patch size of 5, and batch size of 128 on our dataset with a threshold value of 0.0038. Moreover, we trained a 128x8 model for 114 epochs using a kernel size of 7, patch size of 2, and batch size of 32 on our dataset with a threshold value of 0.01. Also for both configurations, we utilized Ada Delta as the optimizer.

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

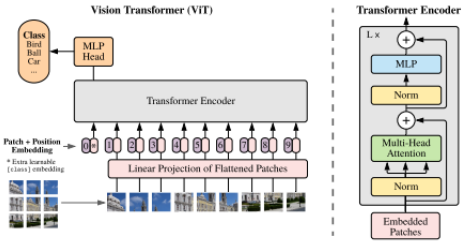

Figure 14: Vision Transformer Structure

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. Finally, we preferred the same hyperparameters for both threshold values. We got patch size as 11, transformer layer as 10, number of heads as 4, and projection dimension as 64. We employed one cycle scheduler as a learning rate scheduler while utilizing AdamW as an optimizer. The weight decay was 0.0001 and the base learning rate was 0.002.

4.3 Technical, Operational & Financial Feasibility

Technical Feasibility

In general, Deep Learning architectures take excessively extended to train, therefore we trained our models on the GPU instead of the CPU in our senior project. When we are going to use the CPU, we have trained the models that we can train for days or weeks, in theory, using NVIDIA CUDA for an average of 8-10 hours.

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 our senior 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, Vision Transformer and CNN-TA architectures using the dataset we generated. Since our dataset is different from the Algorithmic Financial Trading paper, the results we got were also different. We decided to use two different strategies, computational model performance evaluation and financial evaluation, to evaluate performance. In the performance section of the computational model, metrics such as accuracy, recall, precision, and f1 score are used to assess neural network performance; in the financial evaluation section, metrics such as return and sharpe ratio are used to assess the model's financial success. As previously stated, since we trained the models using two separate threshold values, the results for both the 0.01 and 0.0038 threshold values are presented in detail below.

As we observed an increase in validation loss in ConvMixer and Vision Transformer architectures while training the models, we noticed that these models were overfitted. That's why we trained the ConvMixer at 200 epochs and the Vision Transformer at 100 epochs. Since we did not encounter such a problem in the CNN architecture, we trained the model at 500 epochs.

5.1 Computational Model Performance

5.1.1 Threshold Value - 0.01

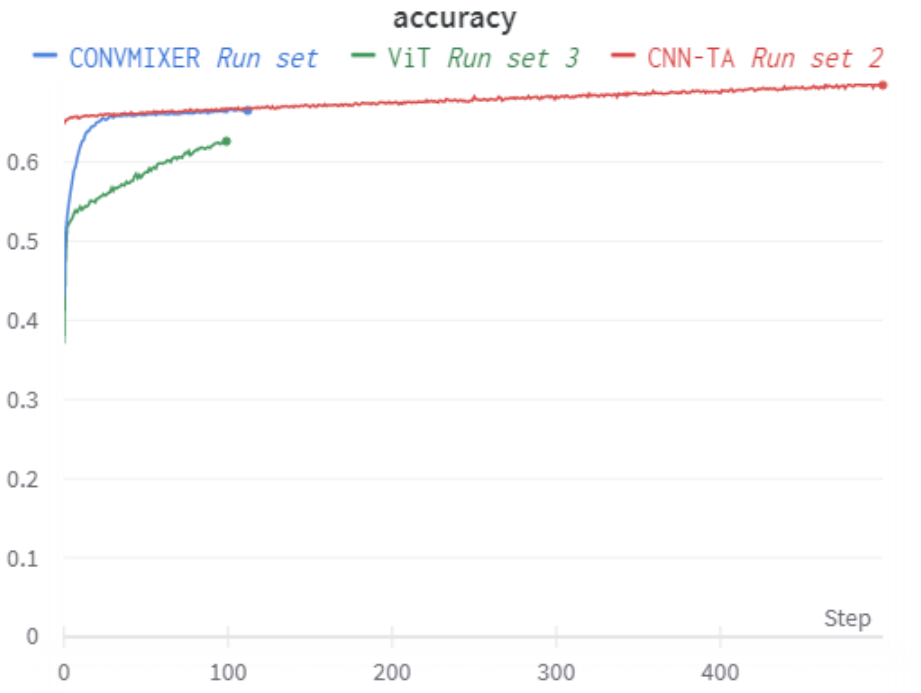

Figure 15: Train Accuracy Graph

When we look at the Figure 15, we can see that CNN-TA started with a high accuracy value, while ConvMixer and Vision Transformer started with a lower accuracy value, but in the following epochs, ConvMixer and CNN-TA reached a very similar Train Accuracy value. However, it is less accurate than ViT, ConvMixer, and CNN-TA.

Figure 16: Train Loss Graph

As seen in the Figure 16, when we look at the Loss values in the graphics where we took the Threshold value as 0.01, in the first epochs, CNN-TA starts from the low loss value, ConvMixer starts from a higher loss value than CNN-TA, also Vision Transformer starts at a higher value than ConvMixer. In following epochs, ConvMixer has achieved a very similar loss with CNN-TA, but ViT remains higher than ConvMixer and CNN-TA.

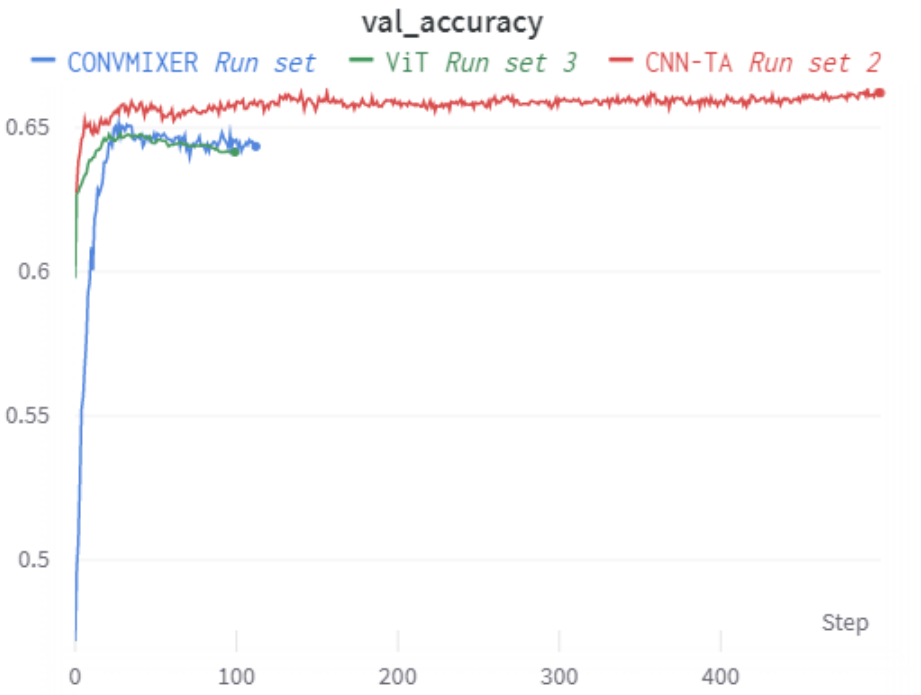

Figure 17: Validation Accuracy Graph

In Figure 17, when we compare the validation accuracy rates of three different architectures, we see that ConvMixer and CNN-TA have oscillation, but as seen in the graph, we see that the highest validation accuracy is in CNN-TA.

Figure 18: Validation Loss Graph

In the Figure 18, when we look at the validation loss values in the graphics where we took the Threshold value as validation loss value, ConvMixer starts from a highest validation loss value. Also Vision Transformer starts at a higher value than CNN-TA.There is an increase in validation loss values in ConvMixer and ViT around the 50th epoch and it is seen that they are slightly overfitting. In addition, oscillations are clearly evident in ConvMixer and CNN-TA.

ConvMixer:

Table 2: Confusion Matrix of ConvMixer

		Predicted		
		Buy	Hold	Sell
Actual	Buy	130	1620	47
	Hold	64	5513	48
	Sell	107	1452	37

Table 3: Classification Report of ConvMixer

Test Accuracy: 0.6299			
	Buy	Hold	Sell
Recall	0.07	0.98	0.02
Precision	0.43	0.64	0.28
F1 Score	0.12	0.78	0.04
Weighted Prec.	0.5362		
Weighted F1	0.5163		

Vision Transformer:

Table 4: Confusion Matrix of Vision Transformer

		Predicted		
		Buy	Hold	Sell
Actual	Buy	73	1690	34
	Hold	57	5528	40
	Sell	57	1517	22

Table 5: Classification Report of Vision Transformer

Test Accuracy: 0.6235			
	Buy	Hold	Sell
Recall	0.04	0.98	0.01
Precision	0.39	0.63	0.23
F1 Score	0.07	0.77	0.03
Weighted Prec.	0.5131		
Weighted F1	0.4995		

CNN-TA:

Table 6: Confusion Matrix of CNN-TA

		Predicted		
		Buy	Hold	Sell
Actual	Buy	82	1626	89
	Hold	45	5500	80
	Sell	54	1450	92

Table 7: Classification Report of CNN-TA

Test Accuracy: 0.6292			
	Buy	Hold	Sell
Recall	0.05	0.98	0.06
Precision	0.45	0.64	0.35
F1 Score	0.08	0.77	0.10
Weighted Prec.	0.5527		
Weighted F1	0.5172		

5.1.2 Threshold Value - 0.0038

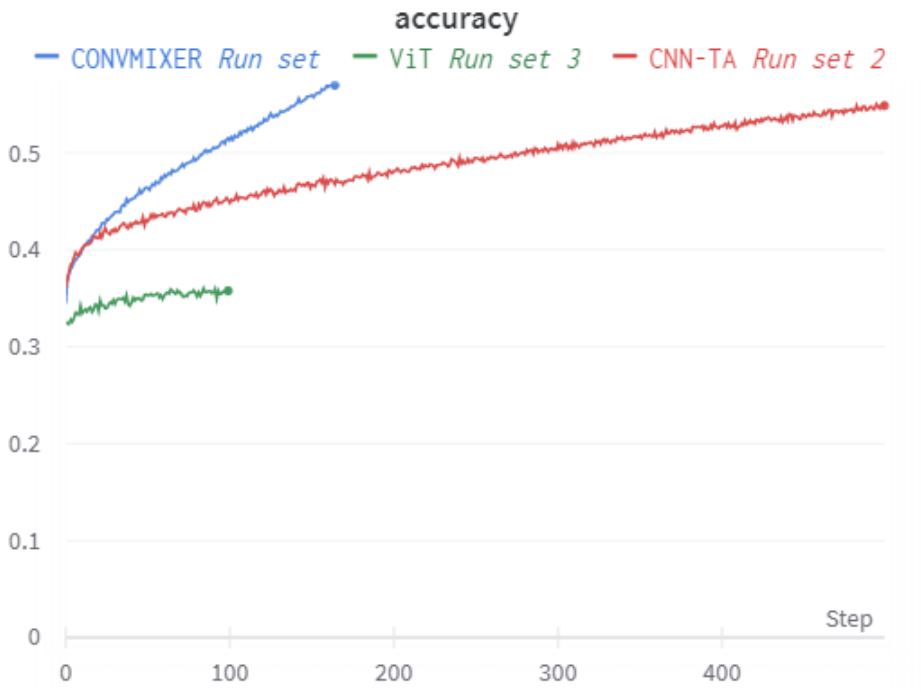

Figure 19: Train Accuracy Graph

When we look at the graphic in Figure 19, we observe that Vision Transformer starts low compared to other architectures and does not increase much. ConvMixer and CNN start with close values, but then ConvMixer rises to higher

values in less epochs. CNN-TA achieves almost the same train accuracy as ConvMixer at the end of the 500 epoch.

Figure 20: Train Loss Graph

As seen in the Figure 20, when we look at the loss values in the graphics where we took the threshold value as 0.0038, in the first epochs, ViT starts from the high loss value, ConvMixer starts same with CNN-TA. In following epochs, There was a decrease in loss values in all three models. However, the decrease in ViT was more sharp than the others.

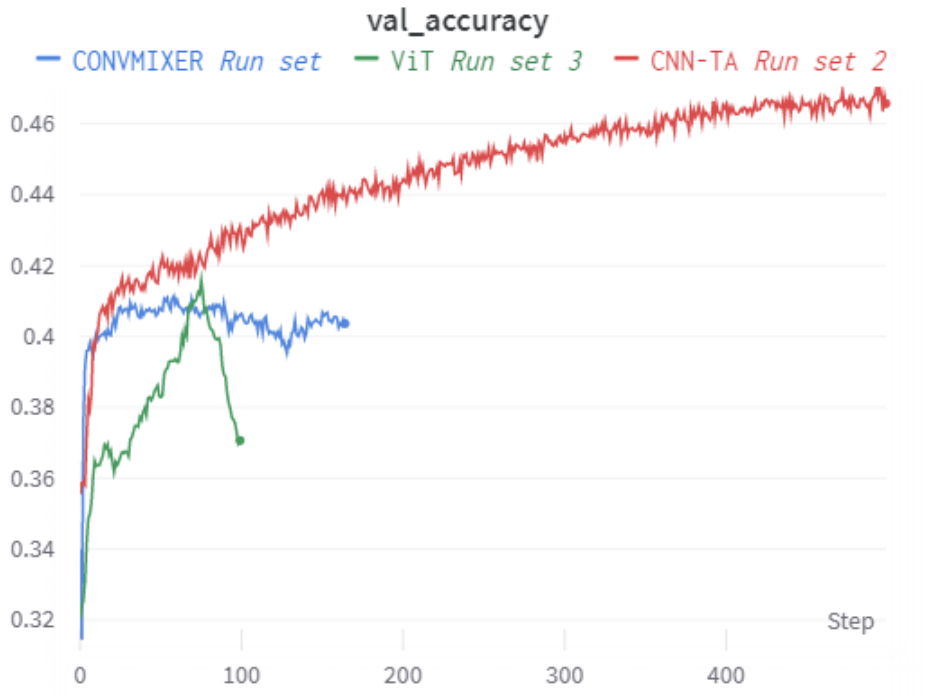

Figure 21: Validation Accuracy Graph

As seen in Figure 21, it starts with the same validation accuracy values in all three architectures. But then, as CNN continues to rise, ConvMixer is slightly overfitted and Vision Transformer is heavily overfitted.

Figure 22: Validation Loss Graph

When we look at the graphic in Figure 22, initially, the validation loss value in ViT is the highest, while ConvMixer has the second highest value and CNN has the lowest. There was a decrease in the validation loss value in all three models. ConvMixer passed ViT in validation loss value after a certain period of time. In addition, overfitting is clearly seen in these 2 models. On the other hand, CNN validation loss value decreased stably.

ConvMixer:

Table 8: Confusion Matrix of ConvMixer

		Predicted		
		Buy	Hold	Sell
Actual	Buy	1623	1036	805
	Hold	918	1202	571
	Sell	1317	873	673

Table 9: Classification Report of ConvMixer

Test Accuracy: 0.3879			
	Buy	Hold	Sell
Recall	0.47	0.45	0.24
Precision	0.42	0.39	0.33
F1 Score	0.44	0.41	0.27
Weighted Prec.	0.3812		
Weighted F1	0.3809		

Vision Transformer:

Table 10: Confusion Matrix of Vision Transformer

		Predicted		
		Buy	Hold	Sell
Actual	Buy	2496	492	476
	Hold	1731	618	342
	Sell	1994	465	404

Table 11: Classification Report of Vision Transformer

Test Accuracy: 0.3901			
	Buy	Hold	Sell
Recall	0.72	0.23	0.14
Precision	0.40	0.39	0.33
F1 Score	0.52	0.29	0.20
Weighted Prec.	0.3762		
Weighted F1	0.3472		

CNN-TA:

Table 12: Confusion Matrix of CNN-TA

		Predicted		
		Buy	Hold	Sell
Actual	Buy	1835	419	1210
	Hold	1128	625	938
	Sell	1428	410	1025

Table 13: Classification Report of CNN-TA

Test Accuracy: 0.3864			
	Buy	Hold	Sell
Recall	0.53	0.23	0.36
Precision	0.42	0.43	0.32
F1 Score	0.47	0.30	0.34
Weighted Prec.	0.3913		
Weighted F1	0.3773		

5.2 Financial Evaluation

Apart from our model's computational performance, we chose to do a financial analysis since, because this model would be utilized in financial trading, one of the best methods to assess its success was to look at how much money it might earn at the end of a test period. We performed the financial analysis in this interval since we selected 2018-2022 as our test dataset. While doing this evaluation, we assumed that we would have 10000$ and we got the results by starting this process from scratch for each etf. The main logic of this process is that as soon as the buy signal comes from the model, it buys the etf at the current price with all the money in its hand, then discards all the incoming signals until any sell signal, that is, it holds the money in its hand. He then sells all the money he has with the sell signal and this logic is repeated until the test period is over. A transaction fee of 1 dollar is taken during all buy and sell transactions. Models were recorded after each epoch. Thus, it was found at which epoch each model got the best Sharpe ratio and average profit percentage values. The reported values are the most accurate versions of the specific metric from all epochs.

5.2.1 Threshold Value - 0.01

Table 14: Best average annual return & best sharp ration for all architecture

	ConvMixer	**ViT**	**CNN-TA**
Avg. Annual Return	12.39%	18.80%	**21.95%**
Sharpe Ratio	**2.74**	2.33	2.57

ETFs	ConvMixer	ViT	CNN-TA	BaH
XLF	1.13%	18.94%	**27.41%**	8.71%
XLU	7.97%	9.91%	12.36%	**12.52%**
QQQ	22.53%	19.16%	15.73%	**29.64%**
SPY	12.03%	20.93%	**24.27%**	17.14%
XLP	13.82%	6.74%	7.95%	**14.34%**
EWZ	12.99%	17.24%	**24.27%**	-2.83%
EWH	**4.39%**	0.00%	2.92%	-3.27%
XLY	9.89%	**25.52%**	21.44%	19.53%
XLE	26.75%	50.75%	**61.20%**	2.27%
Average	12.39%	18.80%	**21.95%**	10.89%
St. Dev.	**7.66%**	13.54%	15.88%	10.26%
Sharpe Rat.	**1.70**	1.52	1.56	1.07

Table 15: Annual returns of the models that gives best average return

ETFs	ConvMixer	ViT	CNN-TA	BaH
XLF	8.98%	**18.04%**	16.95%	8.71%
XLU	7.96%	5.45%	**15.67%**	12.52 %
QQQ	14.38%	11.73%	14.17%	**29.64%**
SPY	5.81%	12.67%	7.45%	**17.14%**
XLP	13.59%	9.78%	8.61%	**14.34%**
EWZ	12.99%	**17.74%**	14.53%	-2.83%
EWH	3.44%	3.92%	**15.35%**	-3.27%
XLY	9.41%	9.68%	11.98%	**19.53%**
XLE	**15.14%**	6.27%	14.37%	2.27%
Average	10.19%	10.59%	**11.70%**	10.89%
St. Dev.	**3.85%**	4.75%	4.66%	10.26%
Sharpe Rat.	**2.74**	2.33	2.57	1.07

Table 16: Annual returns of the models that gives best sharpe ratio

5.2.2 Threshold Value - 0.0038

Table 17: Best average annual return & best sharp ration for all architecture

	ConvMixer	ViT	CNN-TA
Avg. Annual Return	**11.73%**	10.43%	11.09%
Sharpe Ratio	1.10	1.11	**1.57**

ETFs	ConvMixer	ViT	CNN-TA	BaH
XLF	11.48%	**17.57%**	15.14%	8.71%
XLU	2.27%	11.38%	**20.53%**	12.52 %
QQQ	27.10%	**32.67%**	27.03%	29.64%
SPY	11.30%	10.97%	10.83%	**17.14%**
XLP	13.38%	4.58%	12.55%	**14.34%**
EWZ	-5.45%	**2.09%**	-4.72%	-2.83%
EWH	**12.16%**	-5.13%	-0.17%	-3.27%
XLY	**31.22%**	15.81%	25.35%	19.53%
XLE	2.08%	**3.95%**	-6.78%	2.27%
Average	**11.73%**	10.43%	11.09%	10.89%
St. Dev.	11.04%	10.35%	11.84%	**10.26%**
Sharpe Rat.	**1.08**	1.04	0.91	1.07

Table 18: Annual returns of the models that gives best average return

ETFs	ConvMixer	ViT	CNN-TA	BaH
XLF	3.57%	4.17%	**14.66%**	8.71%
XLU	4.18%	**14.55%**	5.37%	12.52 %
QQQ	24.34%	19.54%	7.07%	**29.64%**
SPY	10.18%	6.98%	7.66%	**17.14%**
XLP	9.02%	**18.91%**	1.71%	14.34%
EWZ	-1.95%	-1.37%	**7.45%**	-2.83%
EWH	**6.42%**	-5.06	-1.04%	-3.27%
XLY	**25.15%**	18.12%	10.27%	19.53%
XLE	3.40%	9.65%	**13.70%**	2.27%
Average	9.37%	9.50%	7.43%	**10.89%**
St. Dev.	8.86%	8.53%	**4.81%**	10.26%
Sharpe Rat.	1.10	1.11	**1.57**	1.07

Table 19: Annual returns of the models that gives best sharpe ratio

	ConvMixer	ViT	CNN-TA
Test Accuracy	**62.99%**	62.35%	62.92%
Avg. Annual Return	12.39%	18.80%	**21.95%**
Sharpe Ratio	**2.74**	2.33	2.57

	ConvMixer	ViT	CNN-TA
Test Accuracy	38.79%	**39.01%**	38.64%
Avg. Annual Return	**11.73%**	10.43%	11.09%
Sharpe Ratio	1.10	1.11	**1.57**

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 "Attention Augmented Convolutional Networks" [1], implemented by the Google Brain team under the umbrella of Google AI, presents the augment convolutional networks with a self-attention mechanism. The article titled "Algorithmic Financial Trading with Deep Convolutional Neural Networks: From Time Series to Image Conversion Approach" [2] is about converting time series data in finance into 2D images with the help of technical indicators and making predictions over these images using the Deep Convolutional Neural Network architecture. The article titled "Patches Are All You Need ?" [3] explores whether the success of the Vision Transformer model is the Transformer architecture or patch embedding, and presents the ConvMixer architecture, which uses standard convolutions and patch embedding, as proof. The article named "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" [4] mentions that the Transformer architecture is very successful in natural language processing tasks, but there are not enough applications in the field of computer vision and presents the Vision Transformer model that uses both the Transformer architecture and patch embedding. These articles can be given as examples of related works. Our approach in the senior project is similar to "Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach" [2], but while using Convolutional Neural Network (CNN) in this article, we use the ConvMixer architecture, introduced in 2021, which is thought to outperform CNN according to studies.

Although CNN have dominated computer vision tasks for many years, recent studies has shown that Transformer-based models, most notably the Vision Transformer (ViT), can outperform CNN in some cases [3]. ConvMixer is a highly straightforward model that takes patch embedding from ViT and standard convolutions from CNN. In our senior project, we will compare the newly released ConvMixer architecture, which, as far as we know, has no application in finance before, with Vision Transformer and Convolutional Neural Network models.

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.

Companies or investors can buy their stocks in a optimized way with the 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 has an optimistic social impact, as we believe that when the parameters in the economy are stable and accurate, like our training, the model in our project makes correct predictions, the investors using the model can be happy with the right investments they make. Also, after considering what economic problems we might face in our senior project, we realized that there might be a problem. When things are not going well, that is, if the model we will apply creates an erroneous prediction as a result of situations that occur outside of the learning of the model, we may mislead investors & companies and make a inaccurate investment decision. Additionally, when it is reconsidered in terms of both social and economical impacts, it should not be forgotten that there is always a risk in trading on stocks. There is no such thing as a sure win. We have examined and reported that we will make a certain profit in our own model, but this can have a positive and negative effect both socially and economically depending on the situation. It can be thought that one can easily earn money with the help of an easy model and this is not an ethical thing. However, there is not always a definite gain here. Therefore, there is no ethical problem. When we consider the legal implications of our project, many businesses are actively trading on the stock market using various models; hence, we considered that transaction is lawful around the world.

7.2 Future Work

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

Appendix

You can find the source codes written in this senior project from the links below.
https://github.com/kuantuna/SPDPvCNN

```
'''
CREATING THE IMAGES
'''
nDays = len(indicatorValues[0])
for idx in range(nDays-nIndicators):
    # List, size=n_indicators, contains imageRows of size (n_indicators, 1)
    image = []
    for indicatorValue in indicatorValues:
        # NumPy Array, size=(n_indicators, 1)
        imageRow = indicatorValue[idx:idx+nIndicators][..., np.newaxis]
        image.append(imageRow)
    imageList.append(np.array(image))

'''
CREATING THE LABELS
'''
# Pandas Series, size=n_days-(maxNullVal+nIndicators-1) -> Check this, size is imageList+1, might be a bug.
data_close = data[maxNullVal+nIndicators-1:]["Close"]

# Buy : 0
# Hold: 1
# Sell: 2
for i in range(len(data_close)-1):
    closePriceDifference = data_close.iloc[i+1] - data_close.iloc[i]
    thresholdPrice = threshold * 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

```

Figure 23: Algorithm of Labeling

```python
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

import architectures.helpers.constants as constants

hyperparameters = constants.hyperparameters["cnn_ta"]


def get_ct_model():
    print("Getting the CNN_TA model...")
    model = Sequential()
    model.add(Conv2D(32, hyperparameters["kernel_size"], activation='relu',
                     input_shape=hyperparameters["input_shape"]))
    model.add(Conv2D(64, hyperparameters["kernel_size"], activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(hyperparameters["first_dropout_rate"]))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(hyperparameters["second_dropout_rate"]))
    model.add(Dense(hyperparameters["num_classes"], activation='softmax'))
    model.compile(loss="sparse_categorical_crossentropy",
                  optimizer=keras.optimizers.Adadelta(),
                  metrics=[
                      keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
                      keras.metrics.SparseTopKCategoricalAccuracy(
                          5, name="top5-acc"),
                  ],
                  )
    return model

```

Figure 24: Algorithm of Convolutional Neural Network

```python
import tensorflow_addons as tfa
import architectures.helpers.constants as constants

from tensorflow.keras import layers, regularizers
from tensorflow import keras

hyperparameters = constants.hyperparameters["convmixer"]

''' ConvMixer Implementation'''

def activation_block(x):
    x = layers.Activation("gelu")(x)
    return layers.BatchNormalization()(x)


def conv_stem(x, filters: int, patch_size: int):
    x = layers.Conv2D(filters, kernel_size=patch_size,
                      strides=patch_size)(x)
    # , kernel_regularizer=regularizers.l2(1e-2)
    return activation_block(x)


def conv_mixer_block(x, filters: int, kernel_size: int):
    # Depthwise convolution.
    x0 = x
    x = layers.DepthwiseConv2D(kernel_size=kernel_size, padding="same")(x)
    x = layers.Add()([activation_block(x), x0])  # Residual.

    # Pointwise convolution.
    x = layers.Conv2D(filters, kernel_size=1)(x)
    # ,
    #                   kernel_regularizer=regularizers.l2(1e-2)
    x = activation_block(x)

    return x


def get_conv_mixer_model(
        image_size=hyperparameters["image_size"], filters=hyperparameters["filters"], depth=hyperparameters["depth"],
        kernel_size=hyperparameters["kernel_size"], patch_size=hyperparameters["patch_size"], num_classes=3
):
    """ConvMixer-256/8: https://openreview.net/pdf?id=TVHS5Y4dNvM.
    The hyperparameter values are taken from the paper.
    """
    inputs = keras.Input((image_size, image_size, 1))

    # Extract patch embeddings.
    x = conv_stem(inputs, filters, patch_size)

    # ConvMixer blocks.
    for _ in range(depth):
        x = conv_mixer_block(x, filters, kernel_size)

    # Classification block.
    x = layers.GlobalAvgPool2D()(x)
    outputs = layers.Dense(num_classes, activation="softmax")(x)

    return keras.Model(inputs, outputs)


''' Compiling, Training and Evaluating'''

def compile_model_optimizer(model):
    optimizer = keras.optimizers.Adadelta()

    model.compile(
        optimizer=optimizer,
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"],
    )
    return model

def load_saved_model(path):
    return keras.models.load_model(path, custom_objects={'MyOptimizer': keras.optimizers.Adadelta})


def get_cm_model():
    print("Getting the ConvMixer model...")
    conv_mixer_model = get_conv_mixer_model()
    return compile_model_optimizer(conv_mixer_model)

```

Figure 25: Algorithm of ConvMixer

```python
import tensorflow as tf
import tensorflow_addons as tfa

import architectures.helpers.constants as constants

from tensorflow import keras
from tensorflow.keras import layers

hyperparameters = constants.hyperparameters["vision_transformer"]

"""
Implement multilayer perceptron (MLP)
"""
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x


"""
Implement patch creation as a layer
"""
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches

class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded

def create_vit_classifier():
    inputs = layers.Input(shape=hyperparameters["input_shape"])
    # Augment data.
    # augmented = data_augmentation(inputs)
    # Create patches.
    patches = Patches(hyperparameters["patch_size"])(inputs)
    # Encode patches.
    encoded_patches = PatchEncoder(
        hyperparameters["num_patches"], hyperparameters["projection_dim"])(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(hyperparameters["transformer_layers"]):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=hyperparameters["num_heads"], key_dim=hyperparameters["projection_dim"], dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(
            x3, hidden_units=hyperparameters["transformer_units"], dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=hyperparameters["mlp_head_units"],
                   dropout_rate=0.5)
    # Classify outputs.
    logits = layers.Dense(hyperparameters["num_classes"])(features)
    # Create the Keras model.
    model = keras.Model(inputs=inputs, outputs=logits)
    return model


"""
Compile, train, and evaluate the mode
"""
def compile_model(model):
    optimizer = keras.optimizers.Adadelta()

    model.compile(
        optimizer=optimizer,
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
                 keras.metrics.SparseTopKCategoricalAccuracy(5, name="top5-acc"), ],
    )
    return model

def get_vit_model():
    print("Getting the ViT model...")
    model = create_vit_classifier()
    return compile_model(model)

```

Figure 26: Algorithm of Vision Transformer