



Assignment of bachelor's thesis

Title:	Predicting forex trading signals using methods of image recognition.
Student:	Gleb Fedorov
Supervisor:	Ing. Stanislav Kuznetsov
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

The goal of the Thesis is to use image recognition methods to predict trading signals for buying or selling assets. The student gets trading data of the forex pairs and transforms it into an image using Fast Fourier methods. Then, they prepare a labelling dataset and merge it with images. Finally, they use an image recognition approach to prepare models for predicting trading signals.

1. Prepare a survey in this domain and describe the state-of-the-art methods.
2. Collect the forex assets data Preprocess image and labelling datasets.
3. Prepare image recognition models.
4. Evaluate the models and present the results.

Bachelor's thesis

**PREDICTING FOREX
TRADING SIGNALS
USING METHODS OF
IMAGE
RECOGNITION**

Gleb Fedorov

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Stanislav Kuzetsov
May 12, 2022

Czech Technical University in Prague
Faculty of Information Technology

© 2022 Gleb Fedorov. All rights reserved..

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Fedorov Gleb. *Predicting forex trading signals using methods of image recognition*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Contents

Acknowledgments	vii
Declaration	viii
Abstrakt	ix
List of acronyms	x
Introduction	1
1 Forex preliminaries	5
1.1 Basics	5
1.1.1 Currency pairs	5
1.1.2 Pips	6
1.1.3 Trades	6
1.1.4 Orders	6
1.2 Data	7
2 Preprocessing background	9
2.1 Technical indicators	9
2.1.1 Rate of Change	9
2.1.2 Commodity channel index	10
2.1.3 Relative strength index	10
2.1.4 Ultimate Oscillator	11
2.1.5 Stochastic oscillator	12
2.1.6 Aroon oscillator	12
2.1.7 Chande Momentum Oscillator	13
2.1.8 Absolute price oscillator	13
2.2 Discrete Fourier transform	13
2.3 Filtering	14
2.3.1 Butterworth filter	14
3 Deep learning background	15
3.1 Basics	15
3.1.1 Single-layer perceptron	15
3.1.2 Multilayer perceptron	16

3.1.3	Activation functions	17
3.1.4	Cost function	17
3.2	Optimizers	17
3.2.1	Gradient descent	18
3.2.2	Stochastic gradient descent	18
3.2.3	Adam	19
3.3	Convolutional neural network	19
3.3.1	Convolution operation	20
3.3.2	Convolutional layer	20
3.3.3	Pooling layer	21
3.4	Transformer	22
3.4.1	Self-attention	22
3.5	Transfer learning	23
4	Survey	25
4.1	ImageNet	25
4.2	AlexNet	25
4.2.1	Going deeper	26
4.3	ResNet	27
4.4	DenseNet	27
4.5	Vision Transformer	27
4.6	ConvNeXt	29
5	Implementation	31
5.1	Used technologies	31
5.2	Preprocessing	32
5.2.1	Technical indicators	32
5.2.2	Labeling	32
5.2.3	Filtering	34
5.2.4	Fourier extrapolation	34
5.2.5	Generating images	35
5.2.6	Summary	39
5.3	Modeling and evaluation	39
	Conclusion	43
	Contents of the attached media	49

List of Figures

1.1	A candlestick chart of EUR/USD for a hundred of days	7
3.1	A diagram of a basic MLP	16
3.2	Visualization of a convolution operation.	20
3.3	A convolution layer with multiple filters	21
3.4	Illustration of max pooling	21
3.5	Transformer blocks	23
4.1	AlexNet architecture	26
4.2	Inception module from GoogLeNet	26
4.3	A single residual block	27
4.4	DenseNet architecture	28
4.5	ViT architecture	28
4.6	ResNet and ConvNext blocks comparison	29
5.1	An example of FFT extrapolation	35
5.2	ROC sample images	36
5.3	CCI sample images	36
5.4	RSI sample images	36
5.5	Ultimate oscillator sample images	37
5.6	Stochastic oscillator sample images	37
5.7	Aroon oscillator sample images	37
5.8	Chande Momentum oscillator sample images	38
5.9	Absolute price oscillator sample images	38
5.10	Best test accuracy scores heatmap 8h EUR/USD	40

List of Tables

1.1	Four rows of OHLC data set with 8 hour period from FXCM.	8
5.1	Best test accuracy scores per model 8h EUR/USD	39

List of code listings

5.1	Labeling algorithm in python	33
-----	--	----

I would also like to express my gratitude to my family and friends for supporting me all these years. And I would like to thank my supervisor Ing. Stanislav Kuznetsov, for his mentorship.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 12, 2022

.....

Abstrakt

Algoritmické obchodování je od svého počátku pro mnoho výzkumníků skutečně atraktivní oblastí, která slibuje okamžitý zisk za jakoukoli vynalézavost, kterou do něj vloží. Velmi inspirováni úspěchem rozpoznávání obrazu a hlubokého učení v posledních letech jsme se pokusili zmíněný úspěch využít na Forexu. Připravili jsme přehled současných technik rozpoznávání obrazu a aplikovali je na předpovídání signálů na Forexu. Stejně jako lidé mohou predikovat nadcházející trendy zkoumáním grafů; my jsme natrénovali umělou neuronovou síť, která dokáže předpovídat budoucí cenové trendy z grafů extrapolací vytvořených pomocí Rychlé Fourierovy transformace. Výsledný model dosahuje na testovacích datech výkonnosti 63%.

Klíčová slova Forex, předpovídání časových řad, Fourierova transformace, rozpoznávání obrazu, hluboké učení

Abstract

Algorithmic trading, since its inception, has been a really attractive field for lots of researchers, promising an immediate return for any ingenuity put into it. Greatly inspired by the success of image recognition and deep learning in recent years, we tried to utilize said success in Forex. We prepared a survey of current image recognition techniques and applied them to predict Forex signals. Same as people can conclude upcoming trends by examining graphs; we trained an artificial neural network that can predict future price trends from graphs of extrapolations created via Fast Fourier transform. The final model achieves a performance of 63% on test data

Keywords Forex, time series forecasting, fourier transform, image recognition, deep learning

List of acronyms

EUR	Euro
JPY	Japanese Yen
OHLC	Open High Low Close
ML	Machine Learning
USD	United States Dollar
ROC	Rate Of Change
CCI	Commodity Channel Index
RSI	Relative Strength Index
CMO	Chande Momentum Oscillator
APO	Absolute Price Oscillator
DFT	Discrete Fourier Transform
EMA	Exponential Moving Average
FFT	Fast Fourier Transform
ANN	Artificial Neural Network
MLP	Multilayer Perceptron
SGD	Stochastic Gradient Descent
CNN	Convolutional Neural Networks
ILSVRC	ImageNet Large Scale Visual Recognition Challenge

Introduction

In this section, we will describe our motivation, the problem we are trying to solve, the goals of the thesis, and the rough structure of the work.

Motivation

Forex stands for the foreign exchange market. It is a decentralized global market that allows people to buy and sell different currencies. Forex is the world's largest market with a volume of \$6.6 trillion that determines the foreign exchange rate for every currency. It operates 24h every day except for the weekends, unlike most other markets that are open from 9:30 to 16:00. This also makes it more appealing from the perspective of algorithmic trading.

Forex simultaneously promises an enormous profit and an immense level of risk to any trader. The idea behind Forex is deceptively simple: buy low, sell high (or the opposite of that), so a trader can profit just by predicting a positive or negative change in the upcoming prices of a currency pair. Nevertheless, the actual prediction of the trend is an exceptionally challenging task, and a very select few are able to do it consistently. That is why a tremendous amount of effort has been put into the Forex price analysis and forecasting research since its inception 50 years ago.

The primary source of our inspiration has been the recent successes of deep learning in image recognition. So, in this thesis, we will attempt to utilize said successes in a way more lucrative area of Forex and time series forecasting. For our experiments, we have chosen the currency pair with, by far, the most significant volume: EUR/USD.

Problem statement

Stock price analysis techniques can generally be split into two broad groups: technical and fundamental analysis. The former focuses on prices themselves, believing

that sequences of numbers contain enough information to predict future trends. The latter evaluates stock values by analyzing related economic and financial factors like interest rates, earnings, and Elon Musk's Twitter.

In our thesis, we will attempt Forex trend prediction, focusing exclusively on technical analysis techniques, more specifically machine learning. Furthermore, since image recognition has been the most successful field for ML in the last decade, we will attempt to apply this success to the lucrative field of time series forecasting.

Goals

In this thesis, we are going to aim to achieve the following goals:

- Prepare a survey of state-of-the-art image recognition techniques.
- Collect and preprocess Forex data set into a labeled image data set.
- Prepare image recognition models.
- Evaluate the models and describe the result.

Previous work

Attempts to transform time-series data into images to ease the prediction are not new. Wang (1) achieved competitive performance by encoding time series into images of Gramian Angular Summation/Difference Fields and Markov Transition Fields.

An approach spiritually similar to ours was taken by Tsai (2) by plotting multiple moving averages of the prices and using a convolutional neural network to predict future prices.

Cohen achieved some success in the paper "Visual Forecasting of Time Series with Image-to-Image Regression" (3) by plotting time series directly and training an auto-encoder model on the created images.

The main downside of these approaches is that they do not take advantage of the additional insights gained by using technical indicators widely used for trading by humans.

This thesis is a continuation of the work conducted by Andrey Babushkin (4). We expand on all aspects of the work by using different preprocessing and modeling approaches and focusing on the insight that can be gathered from the images themselves.

Structure

In this work, we are going to roughly follow the CRISP-DM (5) standard with the exception of the deployment phase. So, our thesis is going to be structured as follows:

In the first chapter, we will discuss the business domain of Forex itself, describing the quirks of how Forex trading. Chapter two will cover all the theoretical background necessary for implementing the data preprocessing pipeline. Chapter three will describe the preliminaries necessary to discuss modern models covered in chapter four. As previously stated, chapter four will contain a survey of modern image recognition models. Moreover, in the last chapter, we will describe the data preprocessing pipeline, modeling, and performance of the created models and draw some conclusions.

Chapter 1

Forex preliminaries

This chapter covers basic domain knowledge necessary to discuss Forex trading and signal prediction.

1.1 Basics

In this chapter, we will discuss the basics, necessary to competently tackle the trend prediction problem.

As has already been stated, Forex is a financial market where currencies can be bought and sold. Furthermore, Forex is a decentralized network of traders, where, unlike commodities and shares, Forex trades take place directly between a buyer and a seller, avoiding exchanges.

Forex is, roughly speaking, split into three markets: the spot market, forward Forex market, and future Forex market. Spot markets are the largest segment of Forex, where trades take place instantly at the current price.

Forward market deals with privately-negotiated contracts between a buyer and seller at a future date at a prior specified price. Future contracts are mainly similar to forwards in that they are an agreement to buy or sell some assets at a specific price on a specific date, but unlike forwards, these contracts are marked-to-marked and are traded on the exchange.

Explaining what that means is way out of the scope of this thesis since we will focus exclusively on the spot market, and the other two are mentioned for the sake of completeness.

1.1.1 Currency pairs

Every single currency in the world is traded on Forex. Moreover, all of them can be bought in exchange for another. That is how currency pairs are formed. The most widely traded pairs are EUR/USD and USD/JPY, or Euro to United States Dollar and the United States Dollar to Japanese Yen, respectively. Order within

these pairs is significant. The first currency in a pair is called base currency; the second one is called quote currency. The price of a pair is defined as the ratio of the base currency's price to the quote one.

For example, in the case of the EUR/USD pair, if Euro rises against the dollar, a single Euro is worth more dollars. Therefore the price of the pair rises. Conversely, if the dollar becomes more expensive in regards to Euro, the pair's value will go down.

1.1.2 Pips

Pip or percentage in points is a unit of minimal significant change in a pair's value. The EUR/USD pair price has stayed roughly the same for the last decade, oscillating between 1 and 1.5. Therefore, it is not reasonable to assume that this price will change drastically shortly.

A new measuring unit has been proposed to signify a meaningful change in the price of a pair to address the issue. Pip is simply a change in some decimal place of a pair's price. For most commonly traded currency pairs, like EUR/USD, one pip is a change in the fourth decimal place, except for the ones that involve the Japanese yen. For them, a single pip is a unit of change in the third decimal place.

1.1.3 Trades

So how are the actual trades made? When making a transaction, a trader has two options: they can either go long, i.e., buy the base currency with the expectation that the pair price will rise in value, or they can do the opposite and go short, expecting the pair value to decrease.

This whole principle is relatively intuitive. However, there is one additional nuance in this system: traders have an opportunity to use leverage in their transactions. Leverage is an ability of a trader to borrow money to use in their transaction while depositing only a fraction of the sum upfront. This deposit is called margin. When traders use leverage, they multiply their profit or loss by the leverage value.

For example, if a trader interested in the EUR/USD pair goes long with an initial amount of \$10,000, and the price goes up/down by a single pip, their profit/loss would total \$1, but if they had used the leverage of 10, their loss or profit would be equal to \$10.

1.1.4 Orders

Forex trades are made via orders. Orders are commands for a broker to make a trade in the market on a trader's behalf. It can be scheduled now or sometime in the future when some condition is met. There are two types of orders: market



■ **Figure 1.1** A candlestick chart of EUR/USD for a hundred of days

orders and pending orders. A market order is used to buy or sell at the broker's price instantly.

Pending orders trigger a transaction when the price reaches some specific threshold. This way, traders can automatically take the profit if their prediction is correct. Similarly, these orders allow traders to cut their losses if the situation is not going their way.

When placing a pending order for a specific currency pair, traders need to specify if they are going long or short, the quantity trader wants to buy or sell, and the expiration date for the order. Typically, traders pick a value in pips that they consider a reasonable profit and an acceptable loss. Then, these values are used as a threshold to exit positions via take-profit or stop-loss orders.

1.2 Data

Historical stock data is most widely available in the OHLC data set. The acronym stands for Open High Low Close. Such data sets contain historical asset price data samples with a set period. Each row in the data set contains the pair's price at the start of the period (Open), the highest price pair has reached during this period (High), the lowest price for the period (Low), and the price at the end of the period (Close). Rather frequently, these data sets also contain a volume column. Volume

is the number of lots traded within a given period.

These types of data sets can be often seen in the form of a rather popular candlestick chart, as in figure 1.1.

For the practical part of this thesis, we used historical prices of the EUR/USD pair provided by FXCM. Example rows from the data set can be seen in the table 1.1.

timestamp	Open	High	Low	Close	Volume
2021-12-14 06:00:00	1.12931	1.130855	1.13239	1.12663	66376
2021-12-14 14:00:00	1.12839	1.12589	1.13089	1.12538	53322
2021-12-14 22:00:00	1.12639	1.126915	1.127275	1.125395	17900
2021-12-15 06:00:00	1.12713	1.127365	1.127730	1.12559	56045

■ **Table 1.1** Four rows of OHLC data set with 8 hour period from FXCM.

Preprocessing background

In this chapter we describe all the necessary theory, used to implement the preprocessing pipeline described in the chapter 5 of the thesis.

2.1 Technical indicators

As has already been stated in the introduction, this thesis is focused solely on technical analysis. Hayes provides a great definition: “Technical analysis is a trading discipline employed to evaluate investments and identify trading opportunities by analyzing statistical trends gathered from trading activity, such as price movement and volume. Unlike fundamental analysis, which attempts to evaluate a security’s value based on business results such as sales and earnings, technical analysis focuses on the study of price and volume.” (6)

Specifically, this section covers technical indicators. Chen describes technical indicators rather comprehensively: “Technical indicators are heuristic or pattern-based signals produced by the price, volume, and/or open interest of a security or contract used by traders who follow technical analysis. By analyzing historical data, technical analysts use indicators to predict future price movements.” (7)

2.1.1 Rate of Change

Technical indicators can be conventionally split into several categories. One of them is momentum indicators. “Momentum is the difference between two prices taken over a fixed interval” (8, p. 370). Perhaps the simplest one of them is ROC (Rate of Change). ROC is a pure momentum expressed in percentage. The following formula defines it:

$$ROC_t = \frac{close_t - close_{t-p}}{close_{t-p}} \cdot 100, \quad (2.1)$$

where $close_t$ is an asset's close price at index t at which ROC is calculated and p is a parameter that represents the period, or the amount of ticks previous and the current price.

As is seen from the formula, ROC measures the percentage of change between two prices at the end of two periods n ticks apart. As for its usefulness, Kaufman states: "Momentum can be used as a trend indicator by selling when the momentum value crosses downward through the horizontal line at zero and buying when it crosses above the zero line." (8, p. 373)

2.1.2 Commodity channel index

CCI (Commodity Channel Index) is one of the investors' most popular technical indicators. Donald Lambert initially proposed it in 1980. Schlossberg states that "the prime focus of the CCI was to measure the deviation of the price of the tradable from its statistical average." (9, p. 91).

CCI at index t for some period p is given by equation 2.2.

$$CCI_t = TP_t - \frac{SMATP_{p,t}}{0.015} \times MD_{p,t}, \quad (2.2)$$

where TP_t is the typical price, defined by equation 2.3, $SMATP_{p,t}$ is simple moving average of the typical price given by equation 2.4 and $MD_{p,t}$ is mean deviation of the typical price 2.5. And t is some index from the data set.

$$TP_t = \frac{high_t + low_t + close_t}{3} \quad (2.3)$$

$$SMATP_{p,t} = \frac{1}{p} \sum_{i=0}^p TP_{t-i} \quad (2.4)$$

$$MD_{p,t} = \frac{1}{p} \sum_{i=t}^{t-p} |SMATP_{p,t} - TP_i| \quad (2.5)$$

2.1.3 Relative strength index

RSI (Relative Strength Index) is also a momentum indicator. J. Welles Wilder Jr. first introduced it in his book "New Concepts in Technical Trading Systems". It is an indicator that oscillates between the values of 0 and 100. High and low values of RSI are supposed to signify overbought or oversold states of the asset, respectively. What to consider "high" and "low" is arbitrary, but conventionally, thresholds of 70 and 30 are used. Calculation of RSI is a multi-step process described in the section 6 of the book (10, pp. 63-71). Computation of RSI with period p at some index t , greater than p , goes as follows.

1. Step is to calculate up and down closes U_t and D_t via equations 2.6 and 2.7

$$U_t = \max(0, \text{close}_t - \text{close}_{t-1}) \quad (2.6)$$

$$D_t = -\min(0, \text{close}_t - \text{close}_{t-1}) \quad (2.7)$$

2. Calculate the RS (Relative Strength factor) according to the equation 2.8

$$\text{RS}_{p,t} = \frac{\bar{U}_{M,t}^p}{\bar{D}_{M,t}^p}, \quad (2.8)$$

where $\bar{U}_{M,t}$ and $\bar{D}_{M,t}$ are smoothed moving averages of the up and down closes defined recursively as

$$\bar{U}_{M,t}^p = \frac{(p-1)\bar{U}_{M,t-1}^p + y_t}{p} \quad (2.9)$$

$$\bar{D}_{M,t}^p = \frac{(p-1)\bar{D}_{M,t-1}^p + y_t}{p} \quad (2.10)$$

Initial values of smoothed moving averages $\bar{D}_{M,p+1}^p$ and $\bar{D}_{M,p+1}^p$ are given by

$$\bar{U}_{M,p+1}^p = \frac{1}{p} \sum_{i=0}^p U_{t-i} \quad (2.11)$$

$$\bar{D}_{M,p+1}^p = \frac{1}{p} \sum_{i=0}^p D_{t-i} \quad (2.12)$$

3. Convert the RS_t into RSI_t via equation 2.13

$$\text{RSI}_{p,t} = 100 - \frac{100}{1 + \text{RS}_{p,t}} \quad (2.13)$$

2.1.4 Ultimate Oscillator

The Ultimate Oscillator is a technical indicator that is quite similar to RSI. It also signifies that an asset is overbought or oversold when it crosses an upper or lower threshold, respectively. It was first created by Larry Williams in 1976 and published in 1985. The core advantage of Ultimate Oscillator over the other indicators is that it considers momentum over three different time frames, typically 7, 14, and 28 ticks. Computation of the Ultimate Oscillator described in the original article (11), for time frame parameters p_1 , p_2 , p_3 and some index t , greater than the largest of parameters, is defined as

$$\text{UltOsc}_t = 100 \cdot \frac{4 \cdot \text{avg}_{p_1,t} + 2 \cdot \text{avg}_{p_2,t} + \text{avg}_{p_3,t}}{4 + 2 + 1}, \quad (2.14)$$

where

$$bp_t = close_t - \min(low_t, close_{t-1}) \quad (2.15)$$

$$tr_t = \max(high_t, close_{t-1}) - \min(low_t, close_{t-1}) \quad (2.16)$$

$$avg_{p,t} = \sum_{i=0}^p \frac{bp_{t-i}}{tr_{t-i}} \quad (2.17)$$

2.1.5 Stochastic oscillator

Stochastic oscillator is also a momentum indicator developed by George Lane in the late 1950s. In its principle, it compares the prices of an asset to the range of its prices over some time (12). Value of Stochastic oscillator at tick t with fast period p and slow period n is computed the following way:

$$L_{p,t} = \min(close_{t-1}, close_{t-2}, \dots, close_{t-p}) \quad (2.18)$$

$$H_{p,t} = \max(close_{t-1}, close_{t-2}, \dots, close_{t-p}) \quad (2.19)$$

$$\%K_{p,t} = \frac{close_t - L_{p,t}}{H_{p,t} - L_{p,t}} \times 100 \quad (2.20)$$

$$\%D_n = \frac{1}{n} \sum_{i=0}^n \%K_{p,i}, \quad (2.21)$$

where $\%K$ is the value of the fast stochastic oscillator, and $\%D$ is the value of the slow stochastic oscillator. In the provided formula, a slow stochastic oscillator is a simple average of n last $\%K$ s, but an exponential average is sometimes used instead.

2.1.6 Aroon oscillator

Aroon indicator was proposed by Tushar Chande. It is a technical indicator that signifies the strength of a trend. There are two kinds of Aroon indicators: high and low. The Aroon-Up indicator measures the number of ticks since the last high for some period of time. Aroon-Down works analogously with the lows.

The Aroon oscillator is simply the difference between the values of Aroon-Up and Aroon-Down indicators. Therefore, it can be used to gauge the strength of the current trend and the likelihood of its change (13). Here is how it is calculated for some period p at index t greater than p :

$$AroonUp_{p,t} = \frac{p - \text{Ticks since p-tick high}}{p} \times 100 \quad (2.22)$$

$$AroonDown_{p,t} = \frac{p - \text{Ticks since p-tick low}}{p} \times 100 \quad (2.23)$$

$$AroonOsc_{p,t} = AroonUp_{p,t} - AroonDown_{p,t} \quad (2.24)$$

2.1.7 Chande Momentum Oscillator

Change momentum oscillator (CMO) is another oscillator proposed by Tushar Chande in his book “The New Technical Trader” (14, pp. 93-94). It is a momentum oscillator that is supposed to mitigate the weaknesses present in RSI and other momentum indicators. Calculation of CMO for two period p_1 and p_2 at some index t , greater than p_1 and p_2 , is given

$$\text{CMO}_{p_1, p_2, t} = 100 \times \frac{\sum_{i=0}^{p_1} U_{t-i} - \sum_{i=0}^{p_2} D_{t-i}}{\sum_{i=0}^{p_1} U_{t-i} + \sum_{i=0}^{p_2} D_{t-i}}, \quad (2.25)$$

where U_t and D_t are up and down closes defined in equations 2.6 and 2.7.

This indicator is very similar to RSI, but its values are not smoothed, and CMO oscillates between +100 and -100.

2.1.8 Absolute price oscillator

Absolute price oscillator is a relatively simple technical indicator. It is just a difference between an asset’s two close EMAs (Exponential Moving Average). APO crossing above zero is supposed to signify an upward trend, and analogously, crossing below zero means a downward trend. It can be computed trivially for some two periods P and T as

$$\text{APO}_{p_1, p_2, t} = S_{p_1, t} - S_{p_2, t} \quad (2.26)$$

where $S_{p, t}$ is an EMA of the close prices with some period p at index x and a smoothing factor α between 1 and 0, is given by

$$S_{p, x} = \begin{cases} \text{close}_1 & \text{if } x = 1, \\ \alpha \times \text{close}_x + (1 - \alpha)S_{p, x} & \text{if } x > 1. \end{cases} \quad (2.27)$$

2.2 Discrete Fourier transform

Fourier analysis is a significant scientific achievement widely used in engineering, mathematics, physics, and finance. In addition, it is essential to modern technology in fields like signal processing. Fourier analysis is based on the research of famous french mathematician Jean Baptiste Fourier, whose name it inherited.

This short introduction to DFT is condensed from the chapter 10 of the book “Introduction to biomedical engineering” (15, pp. 576-579) by John Enderle.

Any finite discrete signal can be decomposed into a sum of sine or cosine waves. Discrete Fourier Transform is a linear transformation, that computes a set of coefficients necessary for all the simple waves to add up to the original signal. It essentially converts a series of equidistant signal samples from the time domain to the frequency domain. Given an input sequence $x(n)$ DFT is defined as

$$X(m) = \sum_{k=0}^{N-1} x(k) e^{-j \frac{2\pi mk}{N}}; m = 0, 1, \dots, \frac{N}{2}, \quad (2.28)$$

where $x(k)$ represents signal samples, k is the discrete time variable and N is the number of samples of $x(k)$ and m is an index.

And the inverse of the DFT that converts a signal from the frequency domain back to the time domain is defined as

$$x(k) = \frac{1}{N} \sum_{m=0}^{N-1} X(m) e^{j \frac{2\pi mk}{N}}; k = 0, 1, \dots, N - 1 \quad (2.29)$$

In practice, instead of using DFT, a more efficient implementation called FFT (Fast Fourier Transform) is used instead. The outputs of FFT and DFT are identical, but FFT possesses a much faster execution speed.

2.3 Filtering

Filter is a concept borrowed from the field of signal processing. The term itself signifies a process that removes unwanted details or noise from a signal.

2.3.1 Butterworth filter

Butterworth filter is a family of filters that belongs to a broader group of linear continuous-time filters. These filters' primary purpose is to remove some frequencies from the input signal. Butterworth filter was proposed by British physicist Stephen Butterworth in his paper "On the Theory of Filter Amplifiers" (16).

The amplitude response of low-pass Butterworth is given by

$$|H(j\omega)| = \frac{1}{1 + (\frac{\omega}{\omega_C})^{2n}}, \quad (2.30)$$

where ω_C is the cut-off frequency, and n is the order of the filter.

Butterworth filter was designed for maximal flatness of frequency response. An in-depth explanation of signal analysis is way outside of the scope of this thesis. The flatness of frequency response can be thought of as the purity of the signal. A more detailed and formal explanation is provided by Mathworks (17).

Deep learning background

This chapter will discuss the theoretical background on which models from the following chapter are based.

3.1 Basics

Before diving deeper into the more advanced architectures like convolutional neural networks and transformers, we will address the basics to lay the necessary foundation to discuss modern neural network architectures.

3.1.1 Single-layer perceptron

Single-layer perceptron is the simplest kind of artificial neural network (ANN), that was developed by Frank Rosenblatt in the 1950s. It takes n inputs $\mathbf{x} = (x_1, \dots, x_n)^T$ and produces a single output \hat{Y} . Single-layer perceptron has n real parameters called weights, denoted $\mathbf{w} = (w_1, \dots, w_n)^T$, one for each input, and an additional real parameter b called bias. Inner potential ξ of a perceptron is given by equation 3.1.

$$\xi = \sum_{i=1}^n w_i x_i + b = \mathbf{w}^T \mathbf{x} + b, \tag{3.1}$$

and the output of the perceptron is defined by equation 3.2.

$$\hat{Y} = f(\xi), \tag{3.2}$$

where f is a activation function, in the simplest case, a step function, defined in equation 3.3, but it can be replaced with any non-linear function (18).

$$f(\xi) = \begin{cases} 1 & \text{if } \xi \geq 0, \\ 0 & \text{if } \xi < 0. \end{cases} \tag{3.3}$$

Single-layer perceptron is not particularly interesting or powerful on its own, as it is famously incapable of implementing an XOR function. However, it is a building block that makes more capable models possible.

3.1.2 Multilayer perceptron

Multilayer perceptron (MLP) is a quantitative extension of the single-layer one. MLP is a model that consists of l layers of neurons, as seen in figure 3.1. Each layer can contain an arbitrary number of neurons. We will denote the size of each layer as n_1, \dots, n_l and the input size of the whole network as n_0 .

An output of j th neuron in i th layer can be represented as a function

$$g_j^{(i)} : \mathbb{R}^{n_{i-1}} \rightarrow \mathbb{R},$$

that receives n_{i-1} inputs from the previous layer. The output of $g_j^{(i)}$ is computed identically to equations 3.1 and 3.2.

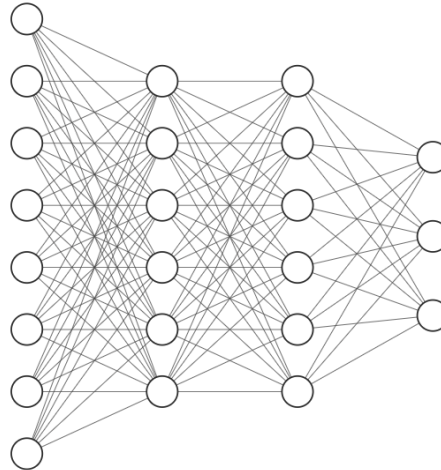
A whole i th layer of the network can be denoted as

$$g^{(i)} : \mathbb{R}^{n_{i-1}} \rightarrow \mathbb{R}^{n_i},$$

, where $g^{(i)} = (g_1^{(i)}, \dots, g_{n_i}^{(i)})^T$. And, of course, the whole neural network can be represented as a function

$$g : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$$

, that is given by composition of all the layers in the network: $g = f^{(l)} \circ f^{(l-1)} \circ \dots \circ f^{(2)} \circ f^{(1)}$ (18).



■ **Figure 3.1** A diagram of a basic MLP

3.1.3 Activation functions

We have already glanced upon activation functions but have not yet given them the attention they deserve. As stated in section 3.1.1 any non-linear function can serve as activation function. They serve a crucial role in neural networks since they introduce non-linearity. A whole MLP without activation functions could be replaced by single matrix multiplication.

Different layers may use different activation functions, but, in practice, the most common activation function for hidden layers (the ones that are neither input nor output) is ReLU 3.4.

$$f(x) = \max(0, x) \quad (3.4)$$

The choice of activation function for the output layer largely depends on the task. For c class classification *softmax* is the most popular option. It is described in the equation 3.5. Moreover, no activation function is used in the last layer for regression tasks.

$$f_i(\boldsymbol{\xi}) = \frac{e^{\xi_i}}{e^{\xi_1} + \dots + e^{\xi_c}}, \quad (3.5)$$

where $\boldsymbol{\xi} = (\xi_1, \dots, \xi_c)^T$ is a vector of inner potentials of c neurons and $f_i(\boldsymbol{\xi})$ is the activation function i th neuron (18).

3.1.4 Cost function

To train a neural network, we need to have some notion of how well the model performs. This is achieved by introducing a cost (often called loss) function.

There are multiple cost functions, each with applications, drawbacks, and advantages. For classification tasks the standard one is *cross-entropy*. For classification of c classes, *cross-entropy* is given by

$$L(Y, \hat{\mathbf{p}}) = - \sum_{j=1}^c \mathbf{1}_{Y=j} \log \hat{p}_j = - \log \hat{p}_Y, \quad (3.6)$$

where $\hat{p}_i = \hat{P}(Y = i | X = x)$ and $\hat{\mathbf{p}} = (\hat{p}_1, \dots, \hat{p}_c)^T$ (18).

3.2 Optimizers

There are several different algorithms for training neural networks called optimizers. An optimizer is an algorithm that updates model parameters during training. While all optimizers diverge in details and often address different nuances of the training process, the core idea is the same. Every optimizer is a modification of the *gradient descent* algorithm.

3.2.1 Gradient descent

Gradient descent is the algorithm that allows neural networks to learn. The goal of gradient descent is to iteratively minimize the average output of the cost function for every item in the training set. To achieve that, we need to compute the vector of partial derivatives of the cost function C with respect to each weight and bias in the network called *gradient*. That is where the algorithm gets its name. *Gradient* is denoted as ∇C as seen in equation 3.7.

$$\nabla C = \left(\frac{\partial C}{\partial w^{(1)}}, \frac{\partial C}{\partial b^{(1)}}, \dots, \frac{\partial C}{\partial w^{(l)}}, \frac{\partial C}{\partial b^{(l)}} \right), \quad (3.7)$$

where $w^{(1)}, \dots, w^{(l)}$ are all the weight vectors and $b^{(1)}, \dots, b^{(l)}$ are all the bias vectors for each layer in the network.

After computing the *gradient*, it is multiplied by *learning rate* and subtracted from the weights and biases in the network, thus slightly nudging the network towards a better solution. *Learning rate*, often denoted as η , is one of the essential hyperparameters of a network. It decides how much the network's weights and biases get updated in each iteration. A network with a too high value of η can skip optimal weight values, and a network with a *learning rate* set too low can train forever. Therefore η must be selected carefully when training a neural network.

3.2.2 Stochastic gradient descent

Neural networks nowadays are trained on big data sets. Therefore precise calculation of the *gradient* can become too computationally intensive even for modern hardware. Stochastic gradient descent (SGD) is a modified gradient descent algorithm designed to mitigate that.

SGD simplifies the task of calculating *gradient* by approximating it. Instead of computing the cost for all entries in the training set, SGD computes *gradient* and updates the model's parameters for entry in the training set separately. The resulting gradients and, thus, parameter updates are less representative of the actual data. However, the error is considered negligible in light of the speed-up achieved by this approach. It was shown that, when we slowly decrease *learning rate*, SGD demonstrates same convergence behavior as the classic *gradient descent* (19).

The mini-batch gradient descent is a middle ground between SGD and the original algorithm. Instead of computing gradient for every entry in the training set, it randomly splits the training data into fixed-size batches and performs parameter updates for each batch. As a result, this approach achieves a more precise gradient approximation than the SGD and is less computationally intensive than the gradient descent algorithm. Conventionally, mini-batch gradient descent is also referred to as SGD.

3.2.3 Adam

Adam is considered the sane default optimizer for most neural networks and types of data sets. Adam was proposed by Diederik in the paper “Adam: A Method for Stochastic Optimization” (20). Adam’s approach is inspired by two other optimizers, Adagrad (21) and RMSprop (RMSprop is an unpublished algorithm, but it is covered by Ruder (19)), and it combines ideas from both of them.

Adam stores an exponentially decaying average of past squared gradients v_t and an exponentially decaying average of past gradients m_t u, both of which are initialized with vectors of zeroes and are updated at each iteration t as defined in equation 3.8.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla C_{t-1} \\ v_t &= \beta_2 m_{t-1} + (1 - \beta_2) (\nabla C_{t-1})^2, \end{aligned} \quad (3.8)$$

where β_1 and β_2 are optimizer hyper parameters.

To combat the bias, introduced by parameters’ initialization with zeroes, the v_t and m_t are corrected as described in equation 3.9.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}, \end{aligned} \quad (3.9)$$

Lastly, both parameters are used to update the network’s parameters according to Adam’s update rule from equation 3.10.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \quad (3.10)$$

where θ_t denotes all parameters of the network at iteration t , η is the learning rate and ϵ is simply a very small number (19).

3.3 Convolutional neural network

Convolutional neural networks (CNN) are a class of artificial neural networks that can be described as a regularized version of multilayer perceptron. CNN’s are comprised of convolutional layers and pooling layers, which we will discuss further in this section, and, typically, some fully connected layers at the end of the network.

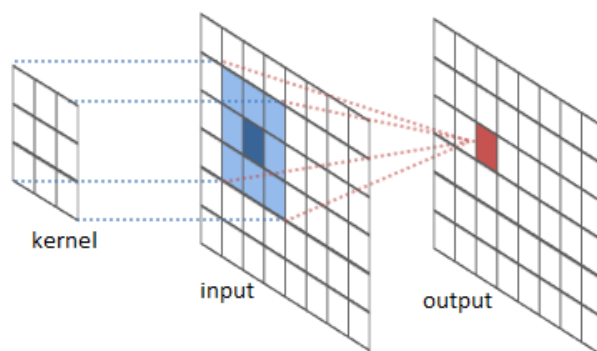
CNN’s are most commonly applied to visual data. Unlike MLPs, CNNs are able to preserve some spatial structure from the input. Therefore they tend to perform significantly better in most image recognition tasks, where this kind of information is crucial.

For example, if we wanted to solve an image classification task with the help of an MLP, we flatten the image to pass it as input to the first layer. As a result,

the network receives a one-dimensional vector of pixel values, thus losing access to all of the spatial structure that was present in the original image.

3.3.1 Convolution operation

Convolution is an operation performed on images with the use of a smaller weights matrix called kernel or *filter*. The kernel is slid systematically over all overlapping kernel-sized patches of the image to produce the output. The element-wise product between each patch and kernel is computed and summed. Values of these sums are the output of the convolution operation. The whole process is better illustrated by figure 3.2, taken from RiverTail (22) documentation. Convolution operation has an important quirk: its output is slightly smaller than the input image.



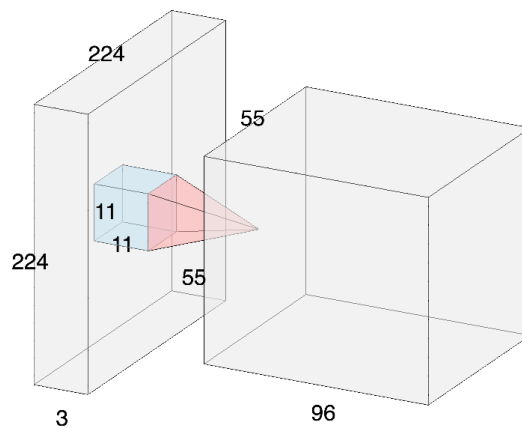
■ **Figure 3.2** Visualization of a convolution operation.

3.3.2 Convolutional layer

As is evident from the name, the convolutional layer employs a convolution operation. Almost always, it employs several of them. Each layer can have multiple *filters*. Each *filter* is a matrix of weights that are optimized during the training process of the network. Every *filter* is used to convolve the input of the layer separately. Results of these convolutions are called *feature maps*. So, a single convolutional layer outputs a volume of *feature maps*: one for each filter it contains, as seen in figure 3.3.

There is a single difference between the convolution operation that we described in section 3.3.1 and the one used inside CNNs: the latter also adds a bias to the weighted sum. After computing the linear transformation that is the convolution, the activation function is applied to each *feature map* analogously to MLPs.

The convolutional layer has three hyper-parameters: a number of filters, filter size, and stride. The number of filters is arbitrary as it defines the depth of the output. Filter size is typically kept between 3×3 and 7×7 . Stride is the step with which filters are moved across the input. It is almost always kept at 1.

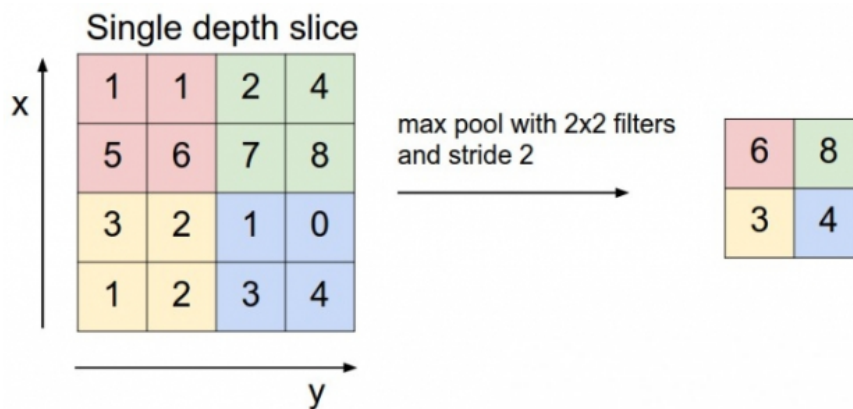


■ **Figure 3.3** A convolution layer with multiple filters

3.3.3 Pooling layer

The primary purpose of the pooling layer is to down-sample feature maps. It addresses a common problem with feature maps being too sensitive to the location of the features in the input (23). Also, it helps the model to generalize by reducing the complexity of the deeper layers.

The most common type of pooling method is *max pooling*. It partitions the input feature map into rectangle patches and only returns the maximum from that patch, as seen in figure 3.4 (24). Another commonly used pooling operation is average pooling. It works similarly to max pooling, but it averages values of the patch instead of picking the maximum.



■ **Figure 3.4** Illustration of max pooling

3.4 Transformer

Transformer is a relatively new neural network architecture introduced in 2017 in the famous paper “Attention is all you need” (25) by a team from Google. The architecture was intended initially for natural language processing applications, but later transformers were shown to be effective in other fields, like image recognition (26).

As the name of the paper suggests, a transformer is an ANN architecture that relies exclusively on the *self-attention*, which we will discuss in the next section, to compute representations of its inputs and output. Another advantage of transformer architecture is that it allows a significantly higher degree of parallelization and thus can be trained significantly faster than previous architectures (25).

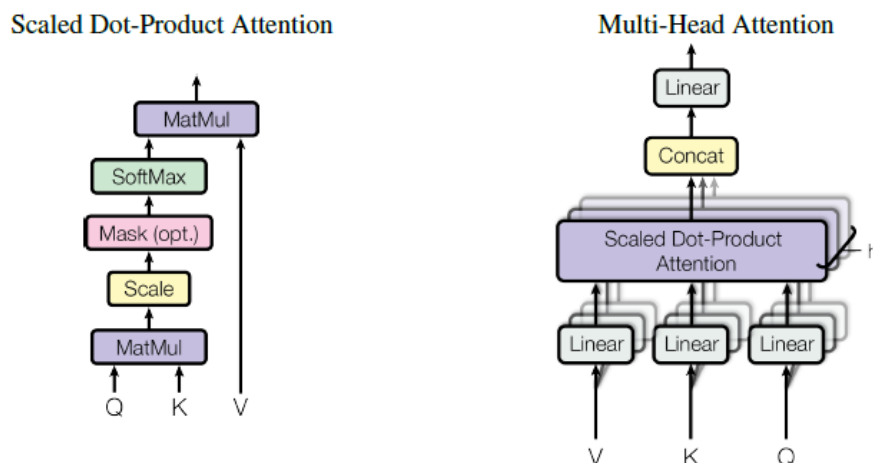
Transformers are structured as *encoder-decoder* models. *Encoder-decoder* is a type of ANN architecture that consists of two parts with unsurprising names: encoder and decoder. Encoder maps a sequence of symbol inputs to a sequence of continuous representations, sometimes referred to as code. The decoder, given the code, produces the output of the model.

Encoder, typically, receives the input in the form of *embeddings*. *Embeddings* are vectors of continuous values that represent discrete values. Embeddings are used to reduce the dimensions of the input. The mapping of input to vectors is learned, allowing us to map similar continuous vectors to discrete values that are semantically similar.

3.4.1 Self-attention

As we have already touched upon, the core of transformers is the *self-attention*. *Self-attention* receives embeddings as input and feeds them into three separate linear layers. These operations produce three distinct vectors, referred to as *key*, *query* and *value*. *Self-attention* then computes dot product of the *keys* and *queries*. The output of the dot product is the similarity matrix of *keys* and *queries* since the dot product is proportional to the cosine of two vectors. The similarity matrix is then scaled down to keep the values of the dot product from exploding. It would hurt the training performance for the next step. A *softmax* is applied to the similarity matrix, and the output of that is multiplied by the value, giving us the output of the whole *self-attention* block. Optionally, the attention head may include a mask. When it is applied to the similarity matrix, all values above the main diagonal are replaced with negative inf. It is done for autoregressive models so that they ignore the inputs they have not seen yet. Overall structure is described more succinctly by the left diagram in figure 3.5 from the paper (25).

This whole computation sounds significantly more complex, than it really is: once we have our *key*, *query* and *value* vectors, the rest can be described with a rather simple equation in matrix form from the original paper (25):



■ **Figure 3.5** Transformer blocks

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (3.11)$$

where V is the *value*, K is the *value*, Q is the *value* query and d_k is the dimension of the *key* vector.

In transformers multiple *self-attention* modules (model, proposed in the original paper uses 8) are stacked together into blocks called *multi-head attention*. In *multi-head attention*, outputs of all self-attention heads are concatenated together and passed through yet another linear layer, as seen on the right diagram in figure 3.5. This allows models to learn multiple connections between the parts of the input.

Why multiple attention heads are needed is better illustrated by an example from natural language processing. Imagine that we want to compute sentiment analysis of restaurant reviews. If the model receives as input a sentence “The food was not terrible”, the transformer has to learn a relationship between the words “not” and “terrible”. This relationship will be drastically different from the connection between “terrible” and “food”. Multiple heads in each attention block allow transformers to learn different relationships separately.


3.5 Transfer learning

Perhaps the most empowering concept discussed in this thesis is transfer learning. It allows us to achieve competitive performance with insufficient training data and significantly less powerful hardware. Transfer learning is a process of taking a model with all of its parameters trained on some data and leveraging it to solve a task on different data.

There are two approaches to transfer learning: using the trained model as a *feature extractor* and *finetuning*. *Finetuning* is the more head-on approach: the

output layer is replaced to fit your specific task, and the whole model is simply trained on the new data. Using model as a *feature extractor* is also rather simple. It is the same procedure as the *finetuning*. However, all model parameters, except for the output layer, are frozen so that they do not change in the training process. The output layer is trained to interpret the representation generated by the original model. *Finetuning* with a lower *learning rate* is often applied afterwards.

Obviously, the more similar the training data used to train the model initially and the one used for the task at hand, the better the model will perform. Nevertheless, since the earlier layers in the model tend to learn simpler, more general features, transfer learning can demonstrate decent results on even dissimilar data, especially on smaller data sets.



Chapter 4

Survey

In this chapter, we will describe modern methods employed in image recognition. We will describe the specific influential models in chronological order, focusing on more general innovations and improvements proposed with these models.

4.1 ImageNet

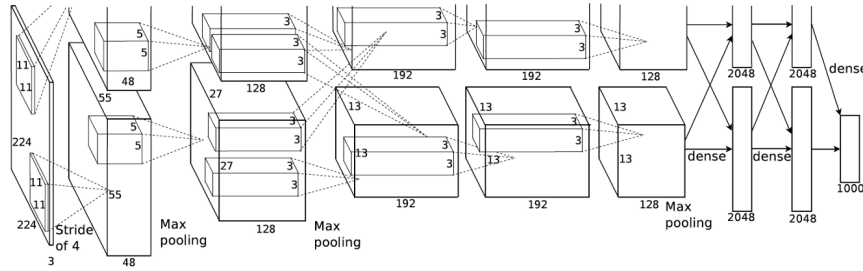
The whole field owes a great deal of its success to the famous ILSVRC (27) competition. ILSVRC stands for ImageNet Large Scale Visual Recognition Challenge. The competition challenges machine learning engineers to train the best model using the ImageNet data set. ImageNet is a gigantic database of images, containing hundreds of thousands of images for every 22,000 categories (15 million images in total). ImageNet has been for a decade and still is the de facto benchmark used to generally evaluate the performance of any image recognition approach and assign it the status of state-of-the-art.

4.2 AlexNet

AlexNet (28), created by Alex Krizhevsky, was the winning model of ILSVRC-2012 competition. It was the first deep learning model to win the competition and attracted much attention to deep learning by greatly outperforming all traditional approaches.

AlexNet is a convolutional neural network with eight layers in total first 5 of which are convolutional and pooling, and the remaining three are fully connected. The architecture is better described by the figure from the original paper 4.1.

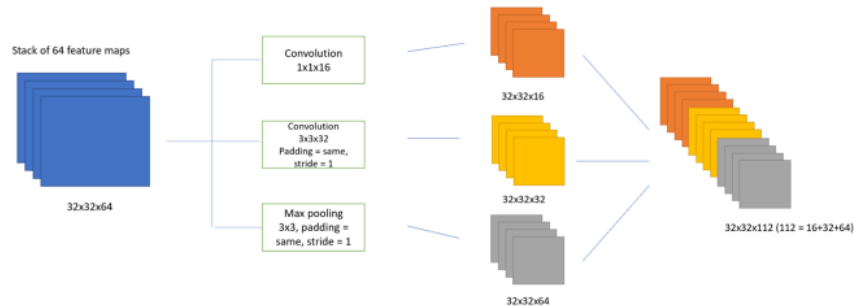
While AlexNet is not as impressive now as it was ten years ago, it set the framework, of stacking 2 to 3 convolutional layers followed by a pooling layer for most CNN architectures onward. It still can show great results on simpler kinds of images.



■ **Figure 4.1** AlexNet architecture

4.2.1 Going deeper

ILSVRC-2014 was dominated by two models: VGG (29) and GoogLeNet (30). Both of these models consolidated the success of AlexNet by becoming significantly deeper and wider. Both models are large: VGG consists of 16 layers, and GoogLeNet has 22, but these models still follow the general framework set by AlexNet.

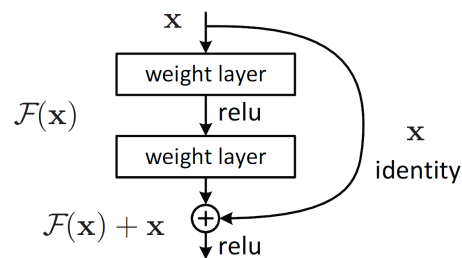


■ **Figure 4.2** Inception module from GoogLeNet

Apart from being significantly larger, GoogLeNet introduced a novelty in the form of the Inception module that allowed running multiple convolution and pooling operations with different filters in parallel, therefore avoiding common trade-offs faced by the prior architectures (31). An Inception module is illustrated in figure 4.2 by Alto (31).

4.3 ResNet

ResNet (32) is a CNN architecture that won 2015 ILSVRC competition. ResNet introduced the concept of *residual learning* that significantly mitigates the vanishing gradient problem that plagued networks with a high number of layers.



■ **Figure 4.3** A single residual block

Residual learning is implemented by the addition of shortcut connections between the layers of the network. These connections simply add inputs of a layer to the inputs of another layer deeper in the model (32). A single such connection can be seen in figure 4.3 from the original paper. This novelty made it possible to train models with hundreds of layers successfully.

While ResNet is outperformed by models discussed further in the thesis on the ImageNet benchmark, it stays the industry standard. As a result, it is often selected as a first model in the experiments or as a baseline for a newly proposed architecture.

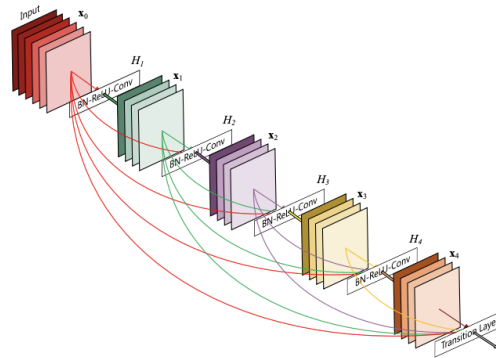
4.4 DenseNet

DenseNet (33) is a successful architecture inspired by ResNet. It takes *residual learning* to the extreme by densely connecting all layers in the neural network, hence DenseNet. In other words, in DenseNet, the input of every hidden layer is formed by concatenating the output of the previous layer with the inputs of all of the layers before it, as seen in figure 4.4 by Huang (33).

DenseNet achieves competitive performance with the drawback of requiring significantly more GPU memory to train.

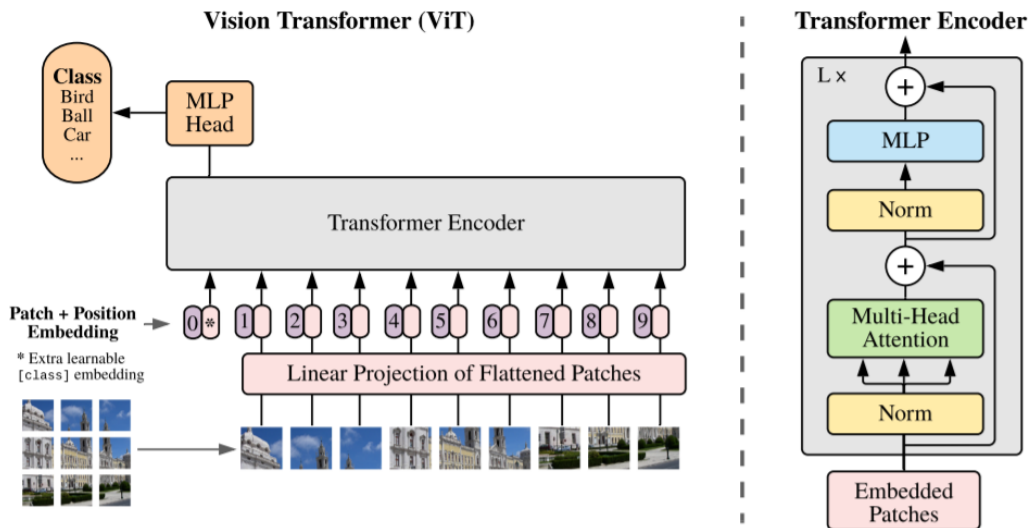
4.5 Vision Transformer

Vision Transformer or ViT is the only transformer model on this list. It was first introduced in the paper “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale” by Alexey Dosovitsky (26). While it is not the first attempt



■ **Figure 4.4** DenseNet architecture

at applying immensely popular transformers to image recognition, it is undoubtedly the first successful one, beating ImageNet’s state-of-the-art performance at the time.



■ **Figure 4.5** ViT architecture

The name of the paper is a reference to the model’s architecture, displayed in the figure 4.5, taken from the paper. The model splits input images into fixed-size patches (e.g., 16×16). These patches are then linearly embedded, and a position embedding is added to them so that the models have an understanding of where these patches are in the image. With an additional learnable embedding prepended to them, the resulting vectors are then fed into the transformer encoder.

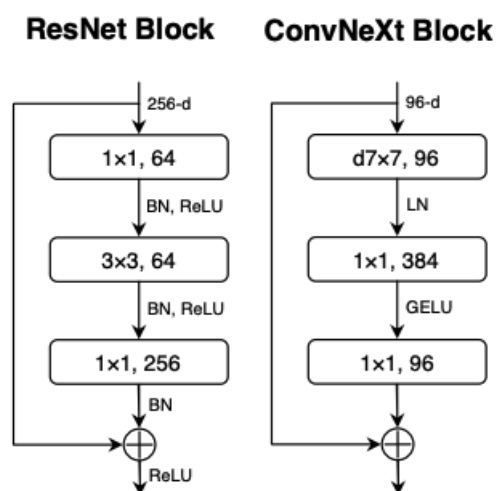
The encoder consists of alternating layers of multiheaded self-attention and multilayer perceptron blocks, both of which we have discussed in the previous chapter, with LayerNorm (34) normalization applied before every block. Finally,

the output of the encoder is passed into a multilayer perceptron that does the classification.

4.6 ConvNeXt

ConvNeXt (35) is a CNN that Takes ResNet as a base and heavily modifies it by applying some modern techniques inspired by transformers. ConvNeXt is partially inspired by the paper “ResNet strikes back: An improved training procedure in timm” (36), which demonstrates that ResNet can achieve much greater performance when trained with the use of more modern techniques.

ConvNext is trained with AdamW (37) optimizer on data augmented with the newer techniques. They also swapped out heavily overlapping 7×7 convolutions with a stride of 2 at the beginning of the network for non-overlapping 4×4 layers with a stride of 4, mimicking the split into patches done by transformers like ViT.



■ **Figure 4.6** ResNet and ConvNext blocks comparison

Figure 4.6 is part of the diagram from the original paper that compares blocks of ResNet and ConvNeXt. As seen in the figure, blocks use larger 7×7 kernels in the convolutional layers. Inverted bottlenecks, also typical in transformers, were introduced to the blocks of the ConvNext. An inverted bottleneck is a sequential pair of layers where the dimension of the first layer is significantly smaller than the dimension of the second one. In the case of ConvNext, it is four times.

Just like transformers, ConvNeXt has fewer activation functions and fewer normalization layers. Also traditionally used in CNNs ReLU activation function and BatchNorm (38) normalization are replaced with GELU (39) activation and

simpler Layer Normalization (34). ConvNeXT also has separate downsampling layers.



Chapter 5

Implementation

In this chapter, we will describe the practical part of the thesis. We will cover all steps of preprocessing the data, generating image data sets, and modeling, and then we will discuss the results.

5.1 Used technologies

The whole practical part of the thesis is entirely written in Python. Python is an interpreted programming language initially created by Guido van Rossum. It has taken off as the de facto standard programming language for most scientific and ML-related purposes in recent years. What sets Python apart from the other languages is an excellent ecosystem of libraries and packages for any computing task, from the web to physical simulations.

In this thesis, we heavily take advantage of Python's ecosystem throughout the implementation. Most work with data is done using the Pandas library. Pandas is an excellent data manipulation package created by Wes McKinnley that allows us to work with large tables of data. All the numeric work, like Fourier transform, is done via the NumPy library. Numpy is a numerical computing library created by Travis Oliphant. Numpy is a very performant package that adds support for n-dimensional arrays and a plethora of mathematical functions and data transformations. To compute the values of technical indicators, we used TA-Lib. TA-Lib is a popular technical analysis library widely used by trading software developers. And finally, to generate the images, we used matplotlib. Matplotlib is an extremely powerful library for creating visualizations in Python, initially created by John D. Hunter.

PyTorch handles the deep learning side of the implementation. PyTorch is a framework for deep learning models. It is a python re-implementation and extension of a Lua package called Torch, hence the name. It is primarily developed by Facebook. Since its release, PyTorch has been quickly adopted by the academic community, and it has become the most used deep learning framework.

5.2 Preprocessing

For the experiments, we have chosen an OHLC EUR/USD data set with 8 hour period, downloaded from FXCM.

5.2.1 Technical indicators

As a first step in the preprocessing pipeline, we will compute values of every technical indicator that we have discussed in the section 2.1 for the whole data set. Most indicators produce NaNs at the beginning of the data set, so we will have to drop those rows, which is not a big issue since those are the least relevant data points in the data set.

Used technical indicator parameter values

- **RSI**: $p = 24$
- **ROC**: $p = 10$
- **Ultimate oscillator**: $p_1 = 7, p_2 = 14, p_3 = 28$
- **Stochastic oscillator**: fast %k period $p = 10$, slow %d period $n = 3$
- **CCI**: $p = 24$
- **APO**: fast period $p_1 = 12$, slow period $p_2 = 26$
- **Aroon oscillator**: $p = 28$
- **Chande oscillator**: $p = 28$

The whole data set is then split into windows of size w (for our experiments, we chose $w = 20$). Each window will be used further down the pipeline to generate a single image for our models.

5.2.2 Labeling

We have to discuss labeling in a separate subsection due to the specificity of the task. As stated in the title of the thesis, we will attempt to predict Forex signals, in other words, an upcoming change in the Forex pair's prices for each window. Nevertheless, what precisely to consider a change and for how many ticks into the future is up for debate.

There are multiple viable ways to mark a window of values as preceding a positive or negative trend. The most obvious way would be to simply consider the value right after the window and decide solely on the change in its value. An

analogous approach is to pick the n th value after the window and label windows based on the change in its value. Finally, a more involved approach is to take n values after the window, fit a line to them, and use its slope as a determining factor. All of the described approaches were used in the previous works related to our topic, but we believe that our approach will lead to better results in a real-world setting.

Our labeling method is based on nuances of the Forex domain, more specifically, how trade orders are placed and executed. Trades and orders are covered in chapter 1. We are going to have three labels: *long*, *short* and *hold*. Both *long* and *short* labels imply a significant positive or negative trend in the upcoming values. On the other hand *hold* means that there is no significant change in the near future prices.

To define what “significant change” is and how many ticks into the future is the “near” future, we will use two hyperparameters: d and l , where d is a threshold in pips, by which currency pair’s value has to change to be considered significant, and l is the number of ticks into the future, that we will consider relevant for a single window.

We are going to label a window (and all images generated from it) as *long* if the first price in the next l ticks that deviates from the last value of the window by more than d pips is greater than the last value of the window. Analogously with *short* labels, but the change has to be negative. If there is not a single price in the next l ticks that changes by more than d pips than the last value of the window, we are going to label the window as *hold*. A sample python implementation of the algorithm is provided in the code listing 5.1.

■ **Code listing 5.1** Labeling algorithm in python

```
import numpy as np

def get_label(close_values, w_end, l, d):
    label = "HOLD"
    diff = close_values[w_end+1:l] - close_values[w_end]
    diff_in_pips = diff * 10000 # 10000 for EUR/USD
    trigger = diff_in_pips[np.argmax(abs(diff_in_pips) > d)]
    if abs(trigger) > d:
        label = "LONG" if trigger > 0 else "SHORT"
    return label
```

The reasoning behind this labeling approach is as follows: when traders commit to a transaction, they also place pending orders that trigger when the price increases or decreases by d pips to take profit, while the situation has not worsened or as a safety net to prevent uncontrollable losses. Therefore, our labeling approach marks windows based on which order will be triggered first.

There are some fees associated with the transaction on all the trading platforms, so the point of the *hold* label is to save traders money on meaningless trades when there are no opportunities for profit.

5.2.3 Filtering

When working with any market data, filtering is an essential part of the preprocessing pipeline, especially in the case of Forex, due to the sheer volume of trades happening simultaneously on the market. Filtering is essential because of large amounts of noise inside Forex data sets that obscure more general trends behind unimportant features. In this work, we are going to remove unimportant signal details using the Butterworth filter that we have covered in section 2.3.1.

Butterworth filter was chosen because it produces a smooth curve with no ripples. These qualities are precious for the next preprocessing step, where we are going to extrapolate indicators into the future, using discrete Fourier transform and for machine learning models in general since it makes input data more similar and helps models to generalize better by preventing *overfitting*.

5.2.4 Fourier extrapolation

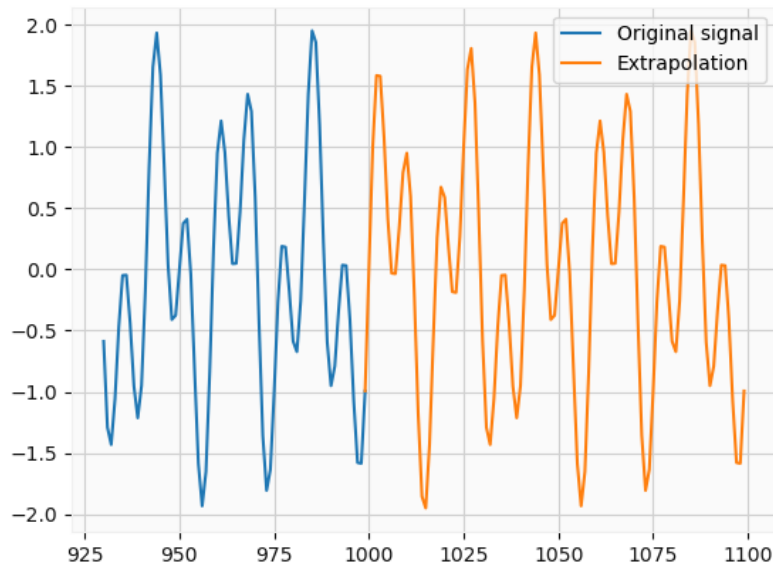
While people do draw some insight from graphs, and some traders successfully use technical indicators on their own, we are going to take it one step further by extrapolation indicators' values into the future using FFT. There are two points behind this approach's intuition.

Firstly, a person, who is experienced in signal processing, can, with a high degree of accuracy, predict if the value of the signal will grow or shrink in the future by examining a graph of the signal's extrapolation done with FFT. We hope that ML models will be able to learn cues contained in the graphs and manage to use them to improve trend prediction.

Secondly, technical indicators are easily interpreted in hindsight, but only some traders are able to create a successful trading strategy using only technical indicators. However, by extrapolating values of the indicators into the future with at least some degree of accuracy, we can use technical indicators as a standalone trading signal, And DFT allows us to do just that.

As has already been covered in section 2.2, Discrete Fourier Transform allows us to decompose signals into a sum of a series of sine or cosine terms. This allows us to extend a series of values by applying FFT to them, forecasting individual components separately, and reassembling the signal by summing together the prolonged components. An example of such extrapolation on simple data can be seen in figure 5.1.

Since Forex prices are not well-behaved by any stretch, we should expect the extrapolation to stay accurate only for a short term, so the extrapolation should only be done for some small number of ticks r .



■ **Figure 5.1** An example of FFT extrapolation

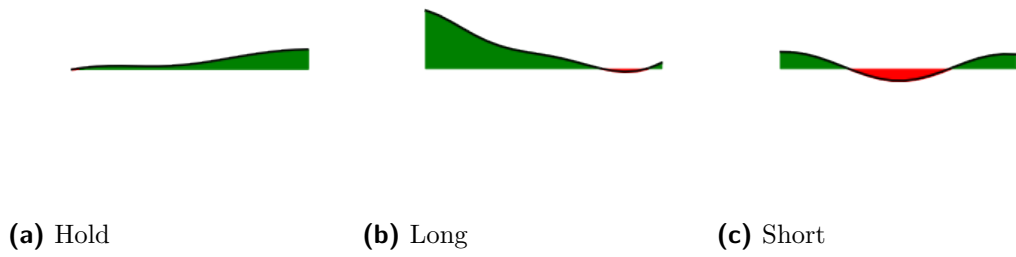
5.2.5 Generating images

All images that we are going to generate can be split into two groups: the ones that have thresholds that hold special meaning and the ones that do not.

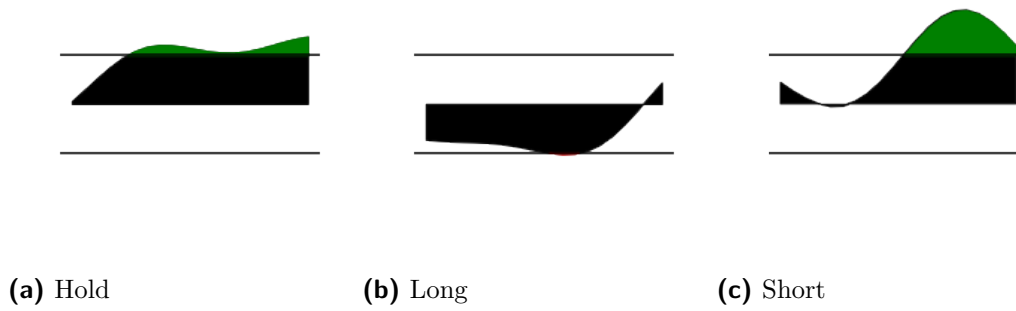
For example, RSI value crossing the upper or lower threshold signifies that an asset is overbought or oversold. We want to make this additional information available to the models. Therefore when the oscillator's value crosses a threshold, we will highlight the area between the curve and the threshold line with a different color, as seen in figure 5.4. For all indicators with the thresholds, we chose 60 and 40 as upper and lower values instead used 30 and 70, which are commonly used for daily trading, because we are working with higher frequency data.

For other technical indicators like ROC, we will simply color the area under the graph green if the value is greater than zero and red if it is less than zero, as seen in figure 5.2.

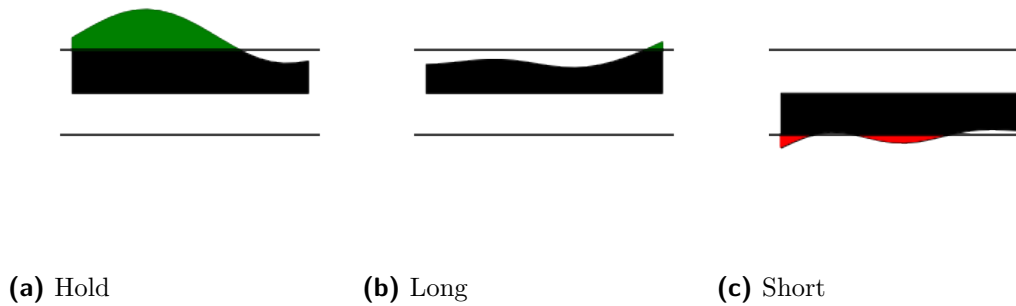
All generated images are 224×224 pixels. Such specific size is dictated by the fact that 224×224 is the lowest resolution that most pre-trained models are able to accept. Samples of generated images for each indicator is displayed in figures 5.2 through 5.9.



■ **Figure 5.2** ROC sample images



■ **Figure 5.3** CCI sample images



■ **Figure 5.4** RSI sample images

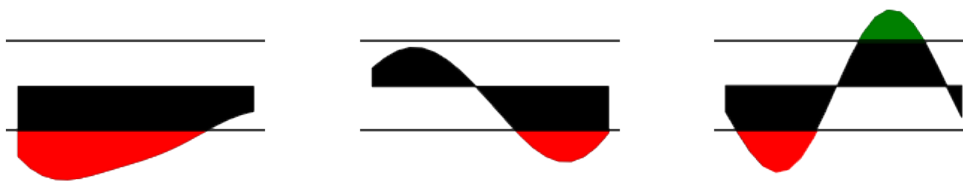


(a) Hold

(b) Long

(c) Short

■ **Figure 5.5** Ultimate oscillator sample images

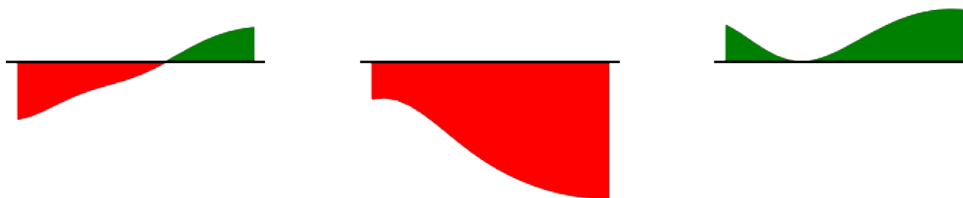


(a) Hold

(b) Long

(c) Short

■ **Figure 5.6** Stochastic oscillator sample images

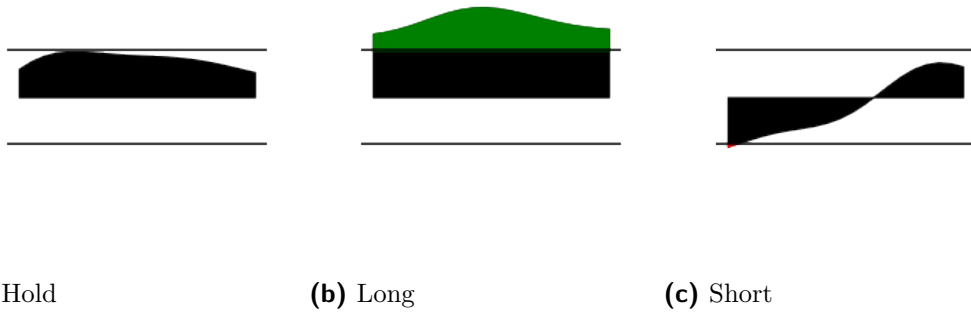


(a) Hold

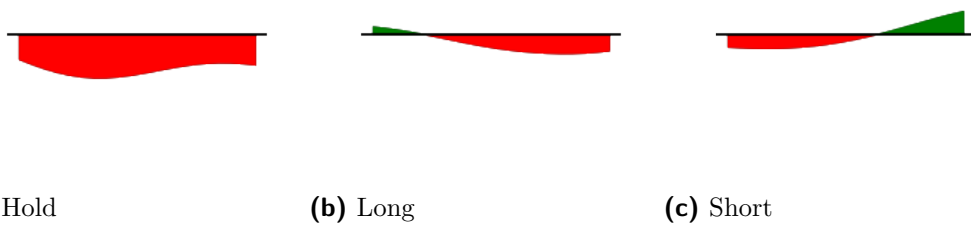
(b) Long

(c) Short

■ **Figure 5.7** Aroon oscillator sample images



■ **Figure 5.8** Chande Momentum oscillator sample images



■ **Figure 5.9** Absolute price oscillator sample images

5.2.6 Summary

Our preprocessing pipeline consists of the following steps:

1. Compute values of technical indicators for the whole data set.
2. Apply Butterworth filter to each indicator.
3. Split data set into windows of size w .
4. Set label for each window. Using d and l .
5. Extrapolate each window r ticks into the future using the FFT.
6. Create images of extrapolation's plots, as covered in the section above.

In our experiments we landed on the following combination of parameter values: $w = 20$, $d = 50$, $l = 20$ and $r = 10$.

5.3 Modeling and evaluation

Instead of coming up with yet another neural network architecture, we are going to experiment on all the models that we have already discussed in the survey 4 chapter of the thesis.

As our performance metric, we are going to simply use *accuracy*. It is a simple, easy-to-understand performance benchmark and the most suitable for our problem domain. Even incorrect predictions of missing trends result in losses for traders in the form of trading platform fees. While those are not as significant as incorrect predictions of opposite trends, we should still avoid them as much as possible.

All models were trained on image data sets generated for each indicator. When splitting data into training, validation, and test sets, we made sure not to shuffle the windows. The model's performance is evaluated based on predicting the future while being trained on the examples from the past.

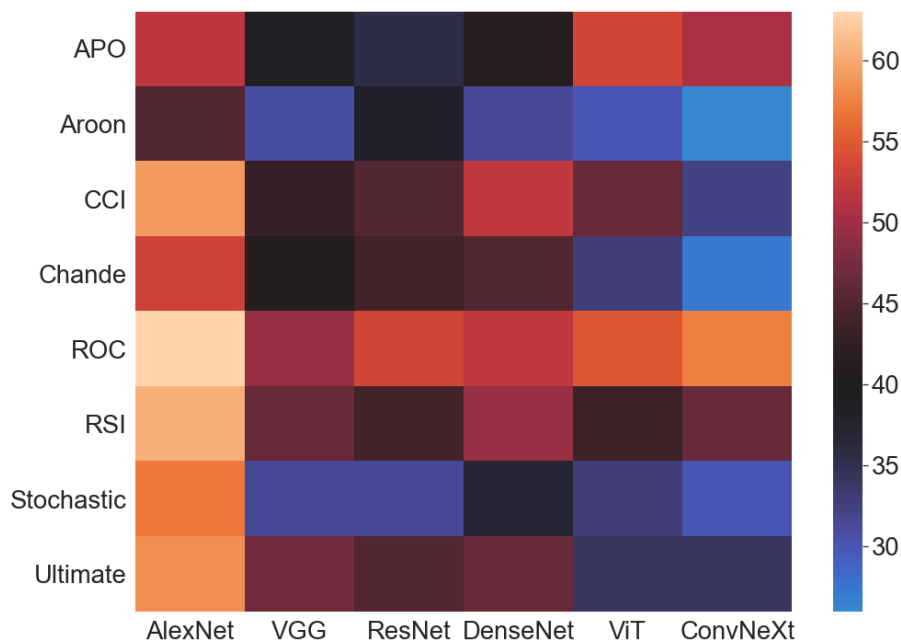
	AlexNet	VGG	ResNet	DenseNet	ViT	ConvNeXt
APO	52.0%	38.6%	35.6%	41.1%	53.4%	50.7%
Aroon	45.1%	31.0%	38.4%	31.5%	30.1%	26.0%
CCI	59.0%	43.0%	45.2%	52.1%	46.6%	32.4%
Chande	53.0%	41.0%	43.8%	45.2%	32.9%	27.4%
ROC	63.0%	49.3%	53.4%	52.1%	54.8%	57.5%
RSI	60.3%	46.6%	43.8%	49.3%	43.6%	46.6%
Stochastic	57.0%	31.5%	31.5%	37.0%	32.9%	30.1%
Ultimate	58.4%	47.0%	45.2%	46.6%	34.2%	34.2%

■ **Table 5.1** Best test accuracy scores per model 8h EUR/USD

Best reached accuracy scores are summarized in table 5.1. To achieve them, AlexNet, VGG, ResNet, and DenseNet were fine-tuned with Adam optimizer and a learning rate $\eta = 10^{-4}$. For more advanced ViT and ConvNext, we opted for AdamW (37), since it is the optimizer that was used to train them in the first place, and a lower learning rate of $\eta = 3 \times 10^{-6}$. All models were trained using *cross-entropy* loss.

The highest accuracy score we achieved is 63% with AlexNet and ROC images. Broader conclusions can be drawn from the heat map of the same table in figure 5.10.

As is evident from the heat map, ROC, the simplest out of all technical indicators, was the most successful across the board, reaching high accuracy scores across all the tested architectures. RSI attained the second-best scores. On the other hand, all models showed subpar performance when trained on images generated from Aroon oscillator.



■ **Figure 5.10** Best test accuracy scores heatmap 8h EUR/USD

The best-performing architecture across all indicators is AlexNet. However, ViT and ConvNeXt displayed terrible results on most indicator data sets, with performance on par with a random classifier.

Overall, results seem to get worse with the complexity of the model. That does make sense due to the following two factors. Firstly, all generated images of the

technical indicator plots are not highly complex. They contain simple geometric shapes in four colors. Therefore an exceedingly deep network might be prone to overfitting.

Secondly, there are only so many 8-hour periods in the year. Moreover, we have less than 20 years' worth of data, which is insufficient to train complex models. This especially hurts transformers, which require more data than convolutional neural networks. The situation is further worsened by the fact that we are not able to apply any data augmentation techniques to artificially increase the size of the data set since it would interfere with the semantics of the generated plots.

Conclusion

In this thesis, we tried to achieve two main goals: surveying currently used image recognition methods and applying them to predict upcoming Forex trends. The first three chapters were dedicated to all the Forex domain and theory preliminaries necessary for chapters 4 and 5. The First goal was fulfilled in chapter 4 of the thesis, where we covered the top neural network architectures of the last decade. To apply models described in chapter 4 to Forex signal prediction, we created a preprocessing pipeline for transforming time-series data into image data sets.

Chapter 5 described the data preprocessing pipeline used to create images. Further, in the same chapter, we described finetuning process of the models from the survey, presented results, and drew some conclusions from the results. The highest accuracy score achieved by us with this approach is 63.0%. This result has been reached using a simpler model and a relatively straightforward technical indicator: AlexNet and ROC. While it might not be an astounding success, Forex asset prices are famously hard to predict, and this result demonstrates that there is at least some merit to the employed approach.

This task allows a tremendous amount of possibilities for improvements. The most obvious one would be to combine models and indicators described in this thesis into an ensemble. Another approach would be to explore event more technical indicators, not focusing solely on momentum and volatility. One could also dive deeper into the more complex indicators to find a more fitting set of hyper-parameters. The way images are generated could also be improved. For example, every image could contain multiple indicators simultaneously, and overlapping their graphs may hold special meaning for future prices. An even more involved approach would be to use a sequence of images for each prediction. This would allow models to capture the movement of the Forex prices and could lead to better results.

The approach showed promising results and, with further development, could be tested in a real-world setting.

Bibliography

1. WANG, Zhiguang; OATES, Tim. Imaging Time-Series to Improve Classification and Imputation. *CoRR*. 2015, vol. abs/1506.00327. Available from arXiv: 1506.00327.
2. TSAI, Yun-Cheng; CHEN, Jun Hao; WANG, Jun-Jie. Predict Forex Trend via Convolutional Neural Networks. *CoRR*. 2018, vol. abs/1801.03018. Available from arXiv: 1801.03018.
3. COHEN, Naftali; SOOD, Srijan; ZENG, Zhen; BALCH, Tucker; VELOSO, Manuela. Visual Forecasting of Time Series with Image-to-Image Regression. *CoRR*. 2020, vol. abs/2011.09052. Available from arXiv: 2011.09052.
4. BABUSHKIN, Andey. *Market signal algorithm based on image recognition*. 2019. Bachelor's Thesis. Czech Technical University in Prague, Faculty of Information Technology,
5. WIRTH, R.; HIPPEL, Jochen. CRISP-DM: Towards a standard process model for data mining. *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*. 2000.
6. HAYES, Adam. *Technical Analysis* [Investopedia] [online]. 2022-03-14 [visited on 2022-03-14]. Available from: <https://www.investopedia.com/terms/t/technicalanalysis.asp>.
7. CHEN, James. *Technical Analysis* [Investopedia] [online]. 2021-09-29 [visited on 2021-09-29]. Available from: <https://www.investopedia.com/terms/t/technicalindicator.asp>.
8. KAUFMAN, Perry J. *Trading Systems and Methods, + Website*. 5th. Wiley Publishing, 2013. ISBN 1118043561.
9. SCHLOSSBERG, Boris. *Technical analysis of the currency market*. Nashville, TN: John Wiley & Sons, 2006. Wiley Trading.
10. WILDER, J.W. *New Concepts in Technical Trading Systems*. Trend Research, 1978. ISBN 9780894590276. Available also from: <https://books.google.cz/books?id=WesJAQAAMAAJ>.

11. WILLIAMS, Larry. The ultimate oscillator. *Technical Analysis of Stocks and Commodities*. 1985, vol. 3, no. 4, pp. 140–141.
12. HAYES, Adam. *Technical Analysis* [Investopedia] [online]. 2021-06-25 [visited on 2021-06-25]. Available from: <https://www.investopedia.com/terms/s/stochasticoscillator.asp>.
13. MITCHEL, Cory. *Technical Analysis* [Investopedia]. 2021-08-24. Available also from: <https://www.investopedia.com/terms/a/aroonoscillator.aspp>.
14. CHANDE, T.S.; KROLL, S. *The New Technical Trader: Boost Your Profit by Plugging Into the Latest Indicators*. Wiley, 1994. Wiley Finance. ISBN 9780471597803. Available also from: <https://books.google.cz/books?id=uPMJAQAAMAAJ>.
15. ENDERLE, John. *Introduction to biomedical engineering*. Academic press, 2012.
16. BUTTERWORTH, Stephen et al. On the theory of filter amplifiers. *Wireless Engineer*. 1930, vol. 7, no. 6, pp. 536–541.
17. THE MATHWORKS, Inc. *What is Frequency Response?* [MathWorks]. Available also from: <https://www.mathworks.com/discovery/frequency-response.html>.
18. KLOUDA, Karel; VAŠATA, Daniel. *Vytěžování znalostí z dat: Neuronové sítě* [online]. 2022 [visited on 2022-03-23]. Available from: <https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-11-cs-handout.pdf>.
19. RUDER, Sebastian. *An overview of gradient descent optimization algorithms*. arXiv, 2016. Available from DOI: 10.48550/ARXIV.1609.04747.
20. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. arXiv, 2014. Available from DOI: 10.48550/ARXIV.1412.6980.
21. DUCHI, John; HAZAN, Elad; SINGER, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*. 2011, vol. 12, no. 7.
22. KUPER, Lindsey. *Bringing Parallelism to the Web with River Trail* [online]. 2016 [visited on 2022-04-24]. Available from: <http://intellabs.github.io/RiverTrail/tutorial/>.
23. BROWNLEE, Jason. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks* [online]. 2019-07-05 [visited on 2021-04-25]. Available from: <https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/>.
24. ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. Understanding of a convolutional neural network. In: *2017 international conference on engineering and technology (ICET)*. 2017, pp. 1–6.

25. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. *Attention Is All You Need*. arXiv, 2017. Available from DOI: 10.48550/ARXIV.1706.03762.
26. DOSOVITSKIY, Alexey; BEYER, Lucas; KOLESNIKOV, Alexander; WEISENBORN, Dirk; ZHAI, Xiaohua; UNTERTHINER, Thomas; DEHGHANI, Mostafa; MINDERER, Matthias; HEIGOLD, Georg; GELLY, Sylvain; USZKOREIT, Jakob; HOULSBY, Neil. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv, 2020. Available from DOI: 10.48550/ARXIV.2010.11929.
27. RUSSAKOVSKY, Olga; DENG, Jia; SU, Hao; KRAUSE, Jonathan; SATHEESH, Sanjeev; MA, Sean; HUANG, Zhiheng; KARPATY, Andrej; KHOSLA, Aditya; BERNSTEIN, Michael; BERG, Alexander C.; FEI-FEI, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*. 2015, vol. 115, no. 3, pp. 211–252. Available from DOI: 10.1007/s11263-015-0816-y.
28. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 2012, vol. 25.
29. SIMONYAN, Karen; ZISSERMAN, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv, 2014. Available from DOI: 10.48550/ARXIV.1409.1556.
30. SZEGEDY, Christian; LIU, Wei; JIA, Yangqing; SERMANET, Pierre; REED, Scott; ANGUELOV, Dragomir; ERHAN, Dumitru; VANHOUCKE, Vincent; RABINOVICH, Andrew. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.
31. ALTO, Valentina. *Understanding the Inception Module in Googlenet*. Available also from: <https://valentinaalto.medium.com/understanding-the-inception-module-in-googlenet-2e1b7c406106>.
32. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. *Deep Residual Learning for Image Recognition*. arXiv, 2015. Available from DOI: 10.48550/ARXIV.1512.03385.
33. HUANG, Gao; LIU, Zhuang; MAATEN, Laurens van der; WEINBERGER, Kilian Q. *Densely Connected Convolutional Networks*. arXiv, 2016. Available from DOI: 10.48550/ARXIV.1608.06993.
34. BA, Jimmy Lei; KIROS, Jamie Ryan; HINTON, Geoffrey E. *Layer Normalization*. arXiv, 2016. Available from DOI: 10.48550/ARXIV.1607.06450.

35. LIU, Zhuang; MAO, Hanzi; WU, Chao-Yuan; FEICHTENHOFER, Christoph; DARRELL, Trevor; XIE, Saining. *A ConvNet for the 2020s*. arXiv, 2022. Available from DOI: 10.48550/ARXIV.2201.03545.
36. WIGHTMAN, Ross; TOUVRON, Hugo; JÉGOU, Hervé. *ResNet strikes back: An improved training procedure in timm*. arXiv, 2021. Available from DOI: 10.48550/ARXIV.2110.00476.
37. LOSHCHILOV, Ilya; HUTTER, Frank. *Decoupled Weight Decay Regularization*. arXiv, 2017. Available from DOI: 10.48550/ARXIV.1711.05101.
38. IOFFE, Sergey; SZEGEDY, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv, 2015. Available from DOI: 10.48550/ARXIV.1502.03167.
39. HENDRYCKS, Dan; GIMPEL, Kevin. *Gaussian Error Linear Units (GELUs)*. arXiv, 2016. Available from DOI: 10.48550/ARXIV.1606.08415.

Contents of the attached media

- README.md description of the media contents
- └─ src
 - └─ impl..... implementation source
 - └─ thesis..... thesis text sources in L^AT_EX
- └─ text..... text of the thesis
 - └─ thesis.pdf..... thesis in PDF format