



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ

Робототехника и комплексная автоматизация (РК)

КАФЕДРА

Системы автоматизированного проектирования (РК6)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:

***«Применение методов машинного обучения (ML) для
решения задач технического анализа при управлении
активами на фондовом рынке»***

Студент РК6-86Б
(Группа)

(Подпись, дата) А.А. Онюшев
(И.О.Фамилия)

Руководитель ВКР

(Подпись, дата) Ф.А. Витюков
(И.О.Фамилия)

Нормоконтролер

(Подпись, дата) С.В. Грошев
(И.О.Фамилия)

2024г.

РЕФЕРАТ

Работа содержит 69 страниц машинописного текста, 31 рисунок, 8 таблиц.
Работа выполнена с использованием 15 источников.

МАШИННОЕ ОБУЧЕНИЕ, АНАЛИЗ ФОНДОВОГО РЫНКА.

Целью работы являлось создание и проведение исследование различных архитектур нейронных сетей для решения задач технического анализа при управлении активами на фондовом рынке.

В данной выпускной квалификационной работе представлены:

- 1) обзор теории устройства нейронных сетей, основанных на архитектурах MLP, CNN, ViT;
- 2) анализ входных данных, поступающих на вход НС;
- 3) описание различных функций, необходимых для данной корректного обучения НС;
- 4) анализ эффективности использования метода ежедневного дообучения;
- 5) проведение тестов различных НС;
- 6) описание метрик, используемых для наглядного сравнения различных НС;
- 7) анализ результатов, полученных по окончанию исследования;
- 8) подбор оптимальных параметров с пояснением.

ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

НС – нейронная сеть.

ДС – датасет.

CNN – (Convolutional Neural Networks) сверточная НС.

MLP – (Multilayer Perceptron) линейный перцептрон.

ViT – (Vision Transformer) зрительный трансформер.

Функция-потерь – функция, показывающая расстояние от предугаданного НС ответа до истинного ответа.

Функция-активации – функция, отвечающая за изменение весов синапсисов.

Тикер – краткое название в биржевой информации котируемых инструментов (акций, облигаций, индексов).

ЕМА – (Exponential Moving Average) экспоненциальная скользящая средняя. Один из показателей, помогающих при техническом анализе.

Stop-Loss – биржевая заявка трейдера, в которой условием исполнения указано достижение цены, которая хуже, чем текущая рыночная.

Take-Profit – биржевая заявка трейдера, которая выставляется заранее, чтобы в случае роста рынка зафиксировать прибыль по бумаге.

MSE – (Mean Squared Error) метрика, используемая для оценки эффективности работы регрессионной модели.

MAE – (Mean Absolute Error) метрика, используемая для оценки эффективности работы регрессионной модели.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	10
1 Разработка архитектуры MLP	11
1.1 Разработка архитектуры.....	11
1.2 Работа с yahoo finance	13
1.3 Кастомный датасет	14
1.4 Настройка и параметризация НС	17
1.5 Обучение НС	17
1.6 Проведение исследования.....	21
1.7 Итоги по MLP	23
2 Разработка архитектуры CNN.....	24
2.1 Устройство CNN	24
2.2 Изображение «в глазах» компьютера.....	26
2.3 Свертка.....	27
2.4 Слой подвыборки (пулинга)	28
2.5 Собственная CNN	29
2.6 Проведение исследования.....	31
2.7 Итоги по CNN	43
3 Разработка архитектуры ViT	45
3.1 Устройство визуальной трансформерной НС.....	46
3.2 Что поступает в трансформер.....	47
3.3 Механизм «Attention».....	48
3.4 Полносвязная нейронная сеть	51
3.5 Блок энкодер.....	52
3.6 Проведение исследования.....	53
3.7 Итоги по ViT	54
4 Метод дообучению.....	56
4.1 Разработка модуля дообучения	56
4.2 Реализация класса создания датасетов	58
4.3 Реализация метода дообучения	59
4.4 Проведение исследования.....	59

4.5	Исследования	60
4.6	Эффективность метода дообучения.....	64
4.7	Закономерности	65
4.8	Оптимальные настройки	65
ЗАКЛЮЧЕНИЕ		66
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		67
ПРИЛОЖЕНИЕ А		69

ВВЕДЕНИЕ

В последние десятилетия развитие технологий дало мощный импульс для применения методов машинного обучения в различных сферах экономики и финансов. Одной из ключевых областей, где эти методы находят своё применение, является технический анализ на фондовом рынке. Компании и индивидуальные инвесторы активно ищут способы повысить эффективность управления активами, используя передовые алгоритмы и аналитические инструменты.

Машинное обучение откроет новые горизонты для предсказания рыночных трендов, автоматизации торговых стратегий и минимизации рисков. Введение данных технологий в процесс управления активами может позволить принимать более обоснованные решения, улучшая как краткосрочные, так и долгосрочные инвестиционные результаты.

Целью данной выпускной квалификационной работы является создание и проведение исследования на нейронных сетях с различными архитектурами. Само исследование заключается в получении наглядных графиков и различных метрик, которые будут указывать на эффективность той или иной настройки нейронной сети. Результаты исследований, в свою очередь, будут помогать принимать решения по оптимизации гиперпараметров НС (нейронная сеть). Работа делится на четыре основные части:

1. Решение задачи технического анализа на архитектуре MLP (линейный перцептрон).
2. Решение задачи технического анализа на архитектуре CNN (сверточная НС).
3. Решение задачи технического анализа на архитектуре ViT (зрительный трансформер).
4. Применение метода дообучения.

1 Разработка архитектуры MLP

Задача данного исследования состоит в изучении различных аспектов линейных нейронных сетей, их структуры, алгоритмов обучения и применения. Будут рассмотрены перцептрон с различным количеством скрытых слоев, а также с различным количеством нейронов в скрытом слое. Также будут исследованы различные методы оптимизации и регуляризации для улучшения производительности нейронных сетей.

В рамках выполнения этого этапа изучим создание перцептрона, пригодного для анализа графика тикеров на бирже и повторения стратегии торговли брокера. Проведем исследование с различными параметрами. Также данное исследование поможет лучше ознакомиться с математической составляющей НС.

Архитектура перцептронов схематично изображена на рисунке 1.

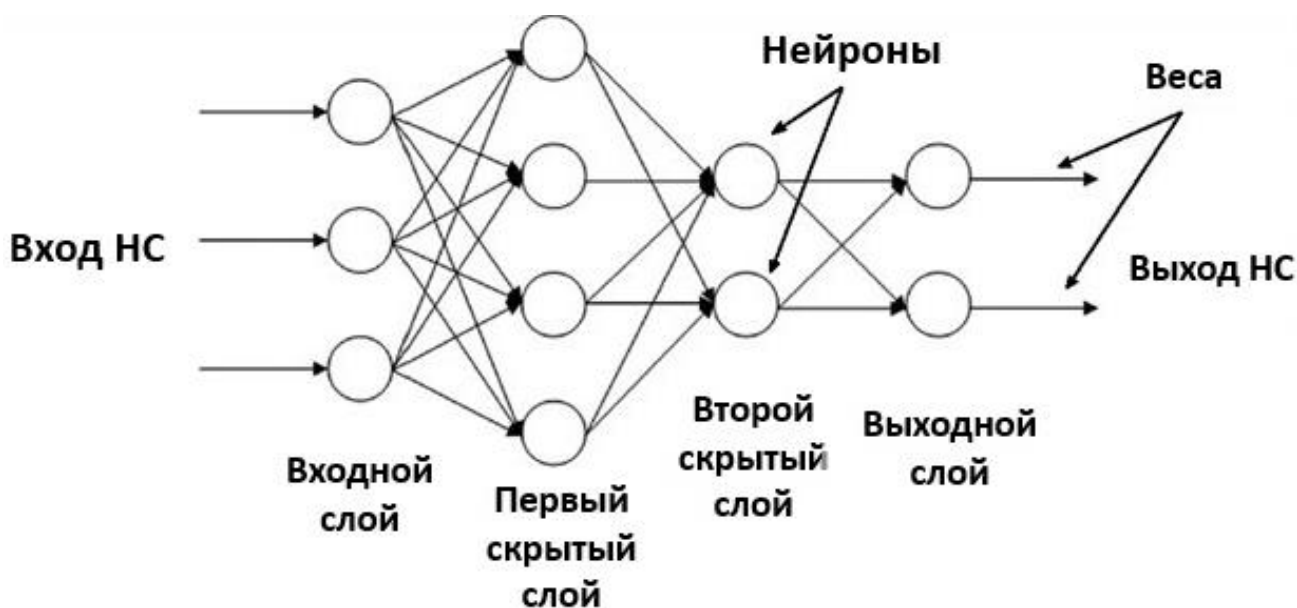


Рисунок 1 – Нейронно-синапсисная архитектура НС

1.1 Разработка архитектуры

При разработке архитектуры нейронной сети требуется учитывать, что на вход должны передаваться данные некоторого размера, которые и будут проанализированы НС. В данной работе на вход НС передается информация о

тикерах в формате: Массив[N] со значениями High, массив[N] со значениями Low, массив[N] со значениями EMA 200 (Exponential Moving Average), где N – количество исследуемых дней от 40 до 2000, а также передается asset – число отвечающее за наличие данного тикера в портфеле.

Проанализировав данные, поданные на вход НС, она возвращает некий результат своего анализа – выходные данные. При разработке архитектуры важно учесть правильность выходных данных. В данной работе выходными данными должны быть два числа в диапазоне от -0.5 до 1.5, означающие биржевые заявки трейдеру – take-profit и stop-loss.

Основное обучение НС происходит в скрытых слоях. Таких слоев может быть огромное количество и каждый из них может содержать абсолютно разное количество нейронов. В данной работе было выбрано использовать 2 скрытых слоя по 100 нейронов в каждом.

Полученная архитектура НС показана на рисунке 2.

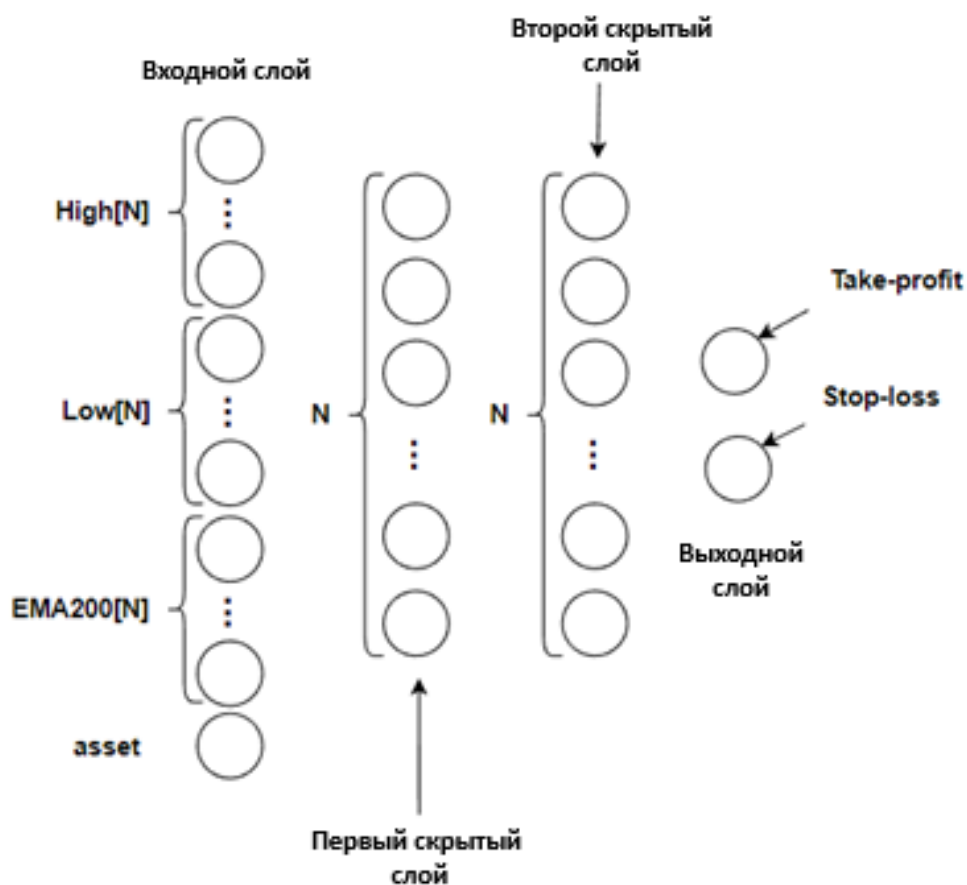


Рисунок 2 – Архитектура исследуемой НС

Реализация данной архитектуры в коде представлена в листинг 1, где в классе *NN_nasdaq* в переменных *linear1*, *linear2*, *linear3* реализованы слои синапсисов между входным и первым скрытым, первым скрытым и вторым скрытым и вторым скрытым и выходным слоями, соответственно.

Листинг 1 – Класс нейронной сети

```

1  class NN_Nasdaq(nn.Module):
2      def __init__(self, input_size, hidden_size, output_size):
3          super().__init__()
4          self.linear1 = nn.Linear(input_size, hidden_size) #
    входной слой
5          self.linear2 = nn.Linear(hidden_size, hidden_size)
6          self.linear3 = nn.Linear(hidden_size, output_size) #
    скрытый слой
7          self.act_func = nn.ReLU() # Задали функцию активации
8          self.flat = nn.Flatten()
9          '''self.model = nn.Sequential(
10              linear1,
11              act_func,
12              linear2
13          )'''
14
15      def forward(self, x):
16          #print(x)
17          out = self.flat(x)
18          out = self.linear1(out)
19          out = self.act_func(out)
20          out = self.linear2(out)
21          out = self.act_func(out)
22          out = self.linear3(out)
23          out = out.reshape((out.shape[0]))
24          return out

```

1.2 Работа с yahoo finance

Для получения необходимой информации о дневных свечах тикера, необходимо обратиться к API yahoo finance, но так как они прекратили его поддержку в 2016 году, воспользуемся модулем *yfinance*.

Чтобы выбрать конкретный тикер, создадим переменную *ticker* (смотреть листинг 2). Чтобы выбрать количество дней информацию по которым мы хотим получить, создадим переменную *days* (листинг 2). Чтобы выбрать количество дней для ЕМА введем переменную *ЕМА_N*.

Листинг 2 – Функция загрузки информации с yahoo finance

```

1  Days = 200 # Кол-во дней в датасете
2  ЕМА_N = 200 # Кол-во дней в ЕМА

```

```

3 num_candle = Days+EMA_N
4
5 ticker = "MSFT" # Названия тикера который отслеживаем
6 hist      =      fin.download(ticker,      period=f'{num_candle}d',
      interval='1d')

```

Функция *fin.download()* (листинг 2) загрузить всю необходимую нам информацию о свечах, а код указанный в листинг 3 создаст столбец с информацией о ЕМА [1].

Листинг 3 – Создание столбца с ЕМА

```

1 hist[f'EMA{EMA_N}'] = hist['Close'].ewm(span=EMA_N,
      adjust=False).mean() # Считаем нужный ЕМА

```

Итоговая таблица будет иметь вид, указанный на таблице 1.

Таблица 1 Итоговый вид датасета

	Date	Open	High	Low	Close	EMA200	Assets	Take_profit	Stop_loss
0	07.12.2021	331,64	335,8	330,1	334,92	334,92			
1	08.12.2021	335,31	335,5	330,8	334,97	334,9205			
2	09.12.2021	334,41	336,49	332,12	333,1	334,9024			
3	10.12.2021	334,98	343	334,79	342,54	334,9784			
4	13.12.2021	340,68	343,79	339,08	339,4	335,0224			
5	14.12.2021	333,22	334,64	324,11	328,34	334,9559			
6	15.12.2021	328,61	335,19	324,5	334,65	334,9529			
7	16.12.2021	335,71	336,76	323,02	324,9	334,8528			
8	17.12.2021	320,88	324,92	317,25	323,8	334,7428			
9	20.12.2021	320,05	322,8	317,57	319,91	334,5953			
10	21.12.2021	323,29	327,73	319,8	327,29	334,5226			
11	22.12.2021	328,3	333,61	325,75	333,2	334,5094			
12	23.12.2021	332,75	336,39	332,73	334,69	334,5112			
13	27.12.2021	335,46	342,48	335,43	342,45	334,5902			
14	28.12.2021	343,15	343,81	340,32	341,25	334,6565			
15	29.12.2021	341,3	344,3	339,68	341,95	334,729			

1.3 Кастомный датасет

Требуемый для исследования датасет, находится вне модуля PyTorch, следовательно, для его корректной загрузки в НС, а также преобразования в тензор, необходимо написать собственный модуль подготавливающий наш датасет под класс Dataset описанный в модуле.

В классе *PreDataLoader* (листинг 4) как раз происходит составление предварительного датасета, который отнормирован и сформирован под требуемые параметры. Основная работа происходит в методе *create_exit_data()*

данного класса (листинг 4). Метод *get_data()* (листинг 5) просто возвращает все значения полученные в методе *create_exit_data()* в формате списка списков списков (необходимая информация о тикере) и значения (требуемый take-profit/stop-loss).

Листинг 4 – Класс предварительного форматирования входных данных

```

1  class PreDataLoader(Dataset):
2      """
3      Class for correct load all data
4      """
5      def __init__(self, data, pred_size=4, label_size=2,
6                  candle_count=50, start=200, stop=350, normalization_pred=True,
7                  normalization_label=False, label_offset=0):
8          """Create DataLoader for your DataSet
9          :Parameters:
10             tickers : str, list
11                 List of tickers to download
12             data: DataFrame
13                 DataFrame of candle parameters
14             pred_size: int
15                 Prediction size(Count of columns with
16 prediction parameters)
17             label_size: int
18                 Label size(Count of columns with label
19 parameters)
20             butch_size: int
21                 Butch size for NN
22             start: int
23                 Number of row that you want to start
24             stop: int
25                 Number of row that you want to finish
26             """
27             self.data = data.copy()
28             self.data2 = data.copy()
29             self.label_offset = label_offset
30             self.pred_size = pred_size
31             self.label_size = label_size
32             self.candle_count = candle_count
33             self.start = start
34             self.stop = stop
35             self.predictions = []
36             self.labels = []
37             self.butches = []
38             self.normalization_pred = normalization_pred
39             self.create_exit_data()
40
41     def create_exit_data(self):
42         prediction = []
43         label = [] if self.label_size>1 else 0
44         for i in range(self.start, self.stop-self.candle_count):
45             for j in range(self.pred_size-1):
46                 for k in range(self.candle_count):
47                     prediction.append(self.data.iloc[:, j][i+k])

```

```

45         prediction.append(self.data.iloc[:, self.pred_size-
46         1][i+self.candle_count-1])
47         for j in range(self.label_size):
48             if (self.label_size == 1):
49                 label = self.data.iloc[:,
50 j+self.pred_size+self.label_offset][i+self.candle_count-1]
51             else:
52                 label.append(self.data.iloc[:,
53 j+self.pred_size+self.label_offset][i+self.candle_count-1])
54
55         if(self.normalization_pred == True):
56             max0 = max(prediction[0:2*self.candle_count])
57             min0 = min(prediction[0:2*self.candle_count])
58             for i in range(len(prediction)-1):
59                 prediction[i] = (prediction[i]-min0)/(max0-
60 min0)
61
62         self.predictions.append(prediction)
63         self.labels.append(label)
64         label = []
65         prediction = []

```

Листинг 5 – Метод возвращающий датасет в требуемом формате

```

1 def get_data(self):
2     exit_data = [[self.predictions[i], self.labels[i]] for i in
3     range(len(self.labels))]
4     return exit_data

```

Класс *NASDAQDataSet* (листинг 6) требуется для использования исследуемого датасета с корректными для НС типами данных. Поскольку НС использует тензоры и модуль PyTorch [2], [3] предоставляет возможность переноса данных на VDDR память GPU, требуется использовать правильные типы данных. Этого можно достичь, унаследовав класс Dataset, который расположен в модуле PyTorch.

Листинг 6 – Класс подготавливающий датасет в требуемом типе данных

```

1 class NASDAQDataSet(Dataset):
2     def __init__(self, data=None):
3         if (data==None):
4             raise Exception("Plz, entry all data")
5         else:
6             self.data=data
7
8     def __len__(self):
9         return len(self.data)
10
11     def __getitem__(self, index):
12         info = torch.tensor(self.data[index][0])
13         label = torch.tensor(self.data[index][1])
14         info = info.to(torch.float32)
15         label = label.to(torch.float32)

```

16	<code>return (info, label)</code>
----	-----------------------------------

1.4 Настройка и параметризация НС

Помимо настройки количества скрытых слоев и количества нейронов в этих слоях, НС требует настройку функции-оптимизации и функции-потерь.

Исследуемая задача более всего похожа на задачу регрессии, основываясь на этом, максимально логичным решением будет использовать функцию `MSELoss`(Mean Squared Error Loss), как функцию-потерь (листинг 7).

Листинг 7 – Добавление функции-потерь `MSE`

1	<code>loss_func = nn.MSELoss()</code>
---	---------------------------------------

Для определения `MSE` используется формула:

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2, \quad (1)$$

где `MSE` – mean squared error (среднеквадратичная ошибка);

`n` – количество входных значений;

Y_i – точная величина;

\hat{Y}_i – предугаданная величина.

После выбора подходящей функции-потерь, необходимо подобрать функцию-оптимизатор. На данный момент одной из самых популярных и эффективных функций-оптимизаторов является `Adam`(adaptive moment), он отлично подойдет и для этой задачи (листинг 8). `Lr` (шаг поиска) достаточно будет указать равным 10^{-3} . Отличный пример сравнения разных функций-оптимизаций приведен на сайте [4].

Листинг 8 – Добавление функции-оптимизатора `Adam()`

1	<code>optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)</code>
---	--

1.5 Обучение НС

Чтобы НС хорошо работала требуется обучить её на данных из всего датасета. Прогоняя эти данные снова и снова мы повышаем вероятность более

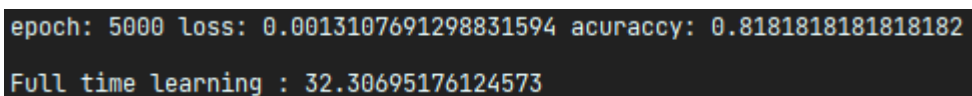
правильного ответа НС. Прогон всего датасета через НС называется – эпоха. Совсем не обязательно в одной эпохе обучать НС всем датасетом за раз, можно разбить его на меньшие части. Такие части называют – батч данных.

Для исследования был составлен пробный датасет, который содержал 91 сущностей для обучения и 22 сущности для тестов. Размер батча равен 50 сущностей. Количество эпох равно 5000. Используемый код для обучения НС (листинг 9).

Листинг 9 – Обучение НС

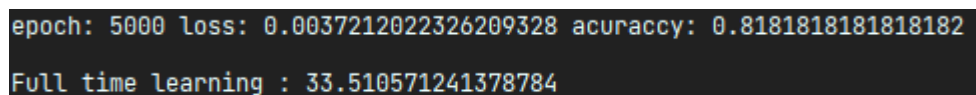
```
1 for epoch in range(epochs):
2     for info, label in dataloader:
3         optimizer.zero_grad() # Обнуляем градиенты, чтобы они не
помешали нам на прогоне новой картинке
4
5         info = info.to(device)
6         label = label.to(device)
7         pred = model(info)
8         loss = loss_func(pred, label) # посчитали ошибку
(значение в label - значение полученное нашим model(img))
9
10        loss.backward() # Прошлись по всему графу вычислений и
посчитали все градиенты для нейронов
11
12        loss_item = loss.item()
13
14        optimizer.step() # Сделали шаг градиентным спуском с
учетом градиента посчитанного loss.backward()
15    print(f'Full time learning : {time.time() - start_time}')
16    path_name = 'model_take_profit_'+device+'.pth' if label_offset ==
0 else 'model_stop_loss_'+device+'.pth'
17    torch.save(model.state_dict(), path_name)
```

Время обучения указано на рисунке 3.



```
epoch: 5000 loss: 0.0013107691298831594 acuraccy: 0.81818181818182
Full time learning : 32.30695176124573
```

(a)



```
epoch: 5000 loss: 0.0037212022326209328 acuraccy: 0.81818181818182
Full time learning : 33.510571241378784
```

(б)

Рисунок 3 – Время обучения НС, а) на CPU, б) на CUDA, соответственно

Ошибка при этом имеет вид, указанный на рисунке 4.

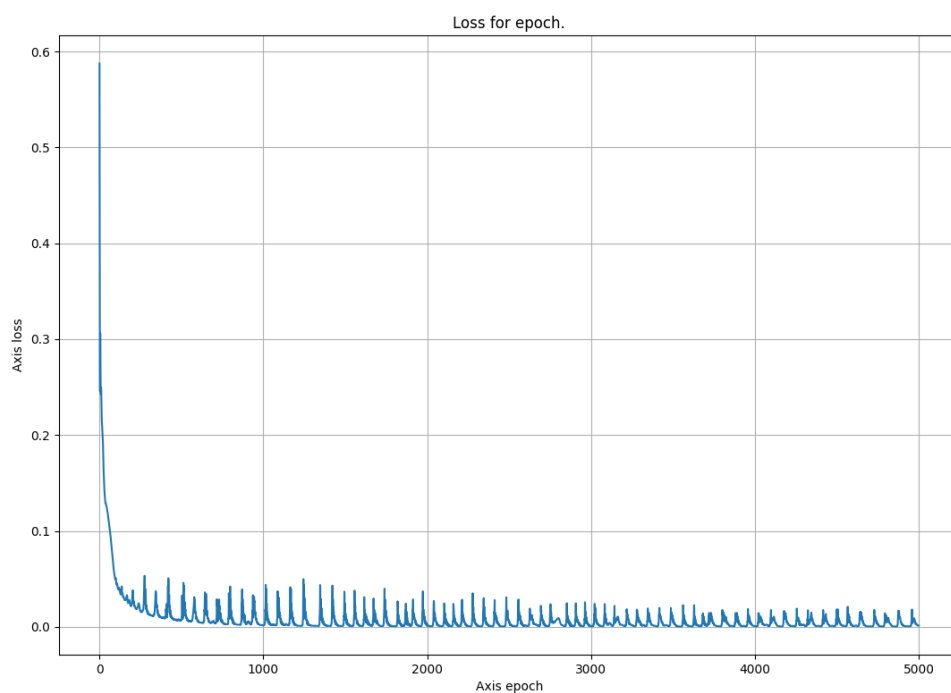


Рисунок 4 – Ошибка при указанных настройках НС

Точность данной НС выглядит, как показано на рисунке 5.

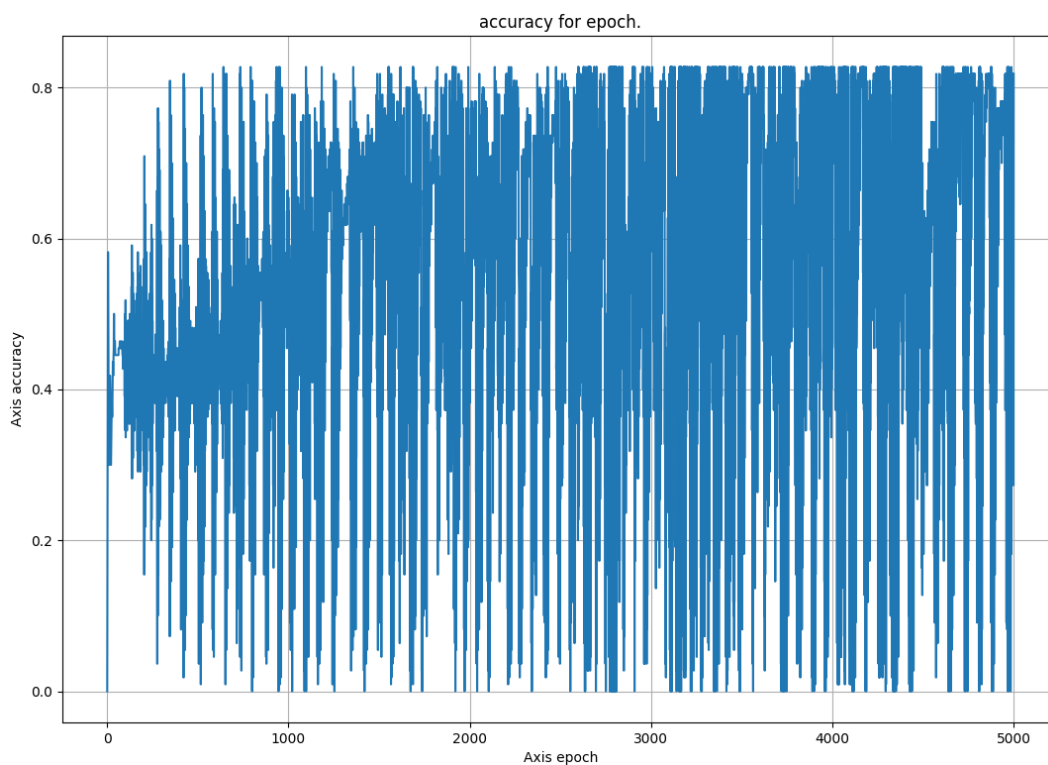


Рисунок 5 – Точность при указанных настройках НС

Ошибка в данной задаче считается, как:

$$|X - X_{NS}|, \quad (2)$$

где X_{NS} – значение, полученное на выходе НС;

X – требуемое значение.

Реализация функции (2) в коде представлена на листинге 10.

Листинг 10 – Функция подсчета ошибки

```
1 def lossing(pred, label):
2     count = 0
3     predict = pred.detach().numpy()
4     lbl = label.detach().numpy()
5     for i in range(len(label)):
6         loss_now = np.abs(lbl[i]-predict[i])
7         count += loss_now
8     return count
```

Точность для задач регрессии сложно посчитать просто сложив все ответы НС, которые идентичны требуемым ответам, поэтому в данной работе точность считается как:

$$\begin{aligned} & \text{if } (X - X_{NS}) < \epsilon : \\ & \quad \text{count} = \text{count} + 1, \end{aligned} \quad (3)$$

где X_{NS} – значение, полученное на выходе НС;

X – требуемое значение;

ϵ – какое-то число, показывающее насколько мала ошибка.

Реализация функции (3) в коде представлена на листинге 11.

Листинг 11 – Функция подсчета точности

```
1 def accuracy(pred, label, epsilon=1):
2     count = 0
3     predict = pred.detach().numpy()
4     lbl = label.detach().numpy()
5     for i in range(len(label)):
6         accuracy_now = (lbl[i]-predict[i])*100
7         if (accuracy_now<epsilon):
8             count+=1
9     return count
```

Также были учтены следующие метрики:

1. Среднее значение величин, которые мы требуем от НС, и которые получаются на выходе из неё.
2. Среднеквадратичное отклонение полученного результата от требуемого.
3. Среднее значение ошибки.
4. Максимальная ошибка.

Получившиеся значения метрик проиллюстрированы на рисунке 6 для этапа тестирования.

```
Average for label_tensor is : 1.052643981846896
Average for result_tensor is : 1.1700253838842565
Standard deviation is : 0.16034090778979193
Error is : 15.232206762676675%
Max error is : 26.913505792617798%
```

Рисунок 6 – Полученные метрики на этапе тестирования

Получившиеся значения метрик проиллюстрированы на рисунке 7 для этапа обучения.

```
Average for label_tensor is : 0.7081414644534771
Average for result_tensor is : 0.7079644048162623
Standard deviation is : 0.034537420637455095
Error is : 4.877192251990224%
Max error is : 603.2340049743652%
```

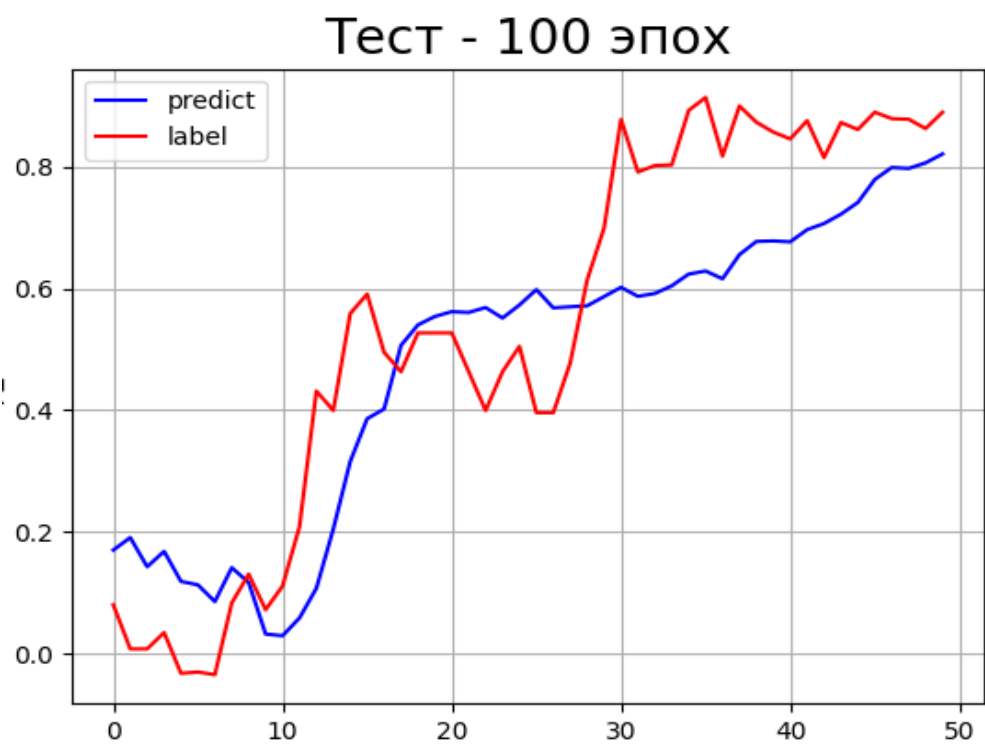
Рисунок 7 – Полученные метрики на этапе обучения НС

1.6 Проведение исследования

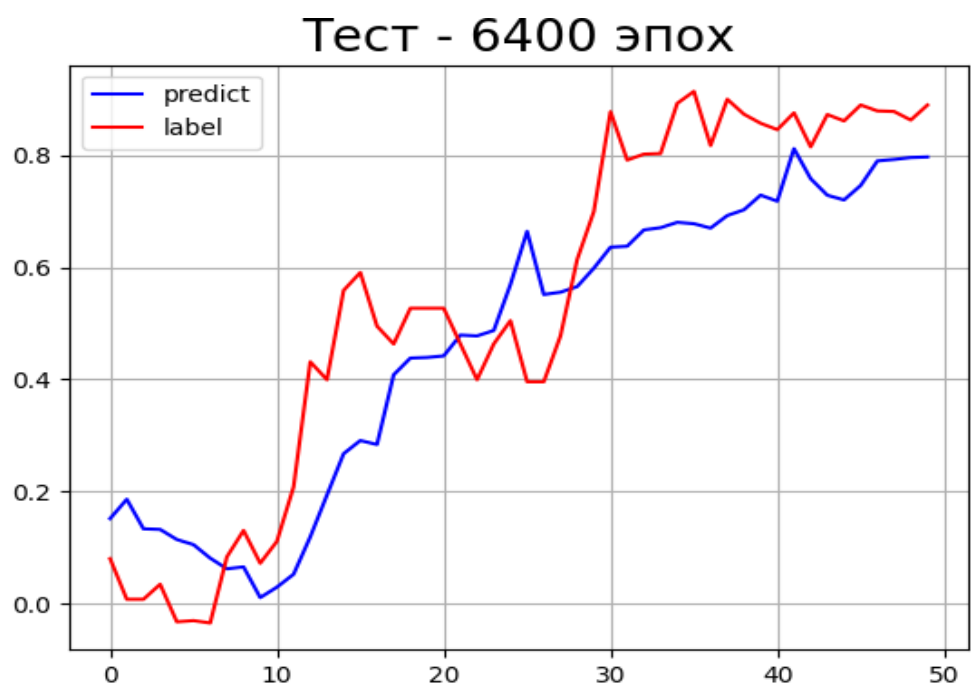
В качестве функций-потерь будем тестировать `MSELoss` и `MAELoss`. Также рассмотрим разное количество эпох (50, 100, 200, 400, 800, 1600, 3200, 6400, 10000). Все предложенные параметры нужны нам, чтобы четко понять как архитектура НС, функция-потерь и количество эпох влияют на итоговый результат.

График красного цвета – требуемый (истинный) результат. График синего цвета – результат, предугаданный НС. По оси X – номера сущностей из тестового датасета. По оси Y – «stop-loss».

На рисунке 8 представлены лучшие графики предсказания.



(a)



(б)

Рисунок 8 – График результата исследования на MLP, а) для 100 эпох, б) для 6400 эпох.

1.7 Итоги по MLP

В ходе выполнения данного этапа прошло ознакомление с написанием искусственного интеллекта на основе НС. Изучены возможности модуля PyTorch языка программирования Python. Разобраны главные математические и биологические основы НС.

Создан линейный перцептрон с возможностью обучения на различных датасетах. Изучены различия в поведении разных функций-оптимизаторов и функций-потерь. Изучены переменные, которые появляются при изменении параметров нейронной сети. Разобрана теоретическая составляющая для сверточных нейронных сетей.

При обучении НС на датасете, составленном по показателям дневных свечей на бирже, можно понять, что линейный перцептрон плохо находит закономерность. Нужен либо датасет большего размера, либо другая структура и архитектура НС, но для данной архитектуры подобранные параметры являются оптимальными.

2 Разработка архитектуры CNN

В настоящее время область сверточных нейронных сетей (CNN) активно исследуется и применяется в различных областях, включая финансовый рынок и биржевую торговлю. CNN позволяют анализировать и обрабатывать большие объемы данных, что идеально подходит для работы с финансовой информацией, которая обычно является многомерной и динамичной.

Использование CNN для анализа биржи и стратегий торговли брокера может быть эффективным способом прогнозирования рыночных движений и принятия обоснованных решений о покупке или продаже активов. Эти модели имеют преимущество в распознавании сложных закономерностей и паттернов на графиках цен, что может помочь в принятии более точных и выгодных решений.

На данном этапе работе рассмотрено, что такое сверточная нейронная сеть и чем она отличается от простого линейного перцептрона. Описана работа по настройке и подбору нужных нам функций-оптимизации, функций-потерь и функций активации, а также размера и количества нейронных слоев и сверточных слоев. Описана работа по составлению датасета на основе данных о тикере MSFT. Также в данной работе были проведены тесты с разными настройками и проанализированы результаты, полученные на выходе. Было проведено сравнение с результатами работы линейного перцептрона при схожих настройках.

2.1 Устройство CNN

Сверточная нейронная сеть (CNN) [5] — это специальная архитектура искусственных нейронных сетей, которая может принимать входное изображение (матрицу данных), присваивать важность (изучаемые веса и смещения) аспектам или объектам изображения (матрицы) и отличать одно от другого. При этом изображения (матрицы), в сравнении с другими архитектурами НС, требуют гораздо меньше предварительной обработки. В

примитивных методах фильтры разрабатываются вручную, но достаточно обученные сети CNN учатся применять эти фильтры/характеристики самостоятельно.

Архитектура CNN аналогична структуре связей нейронов в мозгу человека, учёные черпали вдохновение в организации зрительной коры головного мозга. Отдельные нейроны реагируют на стимулы только в некоторой области поля зрения, также известного как перцептивное поле. Множество перцептивных полей перекрывается, полностью покрывая поле зрения CNN.

Архитектура сверточной нейронной сети схематично изображена на рисунке 9 [6], [7]:

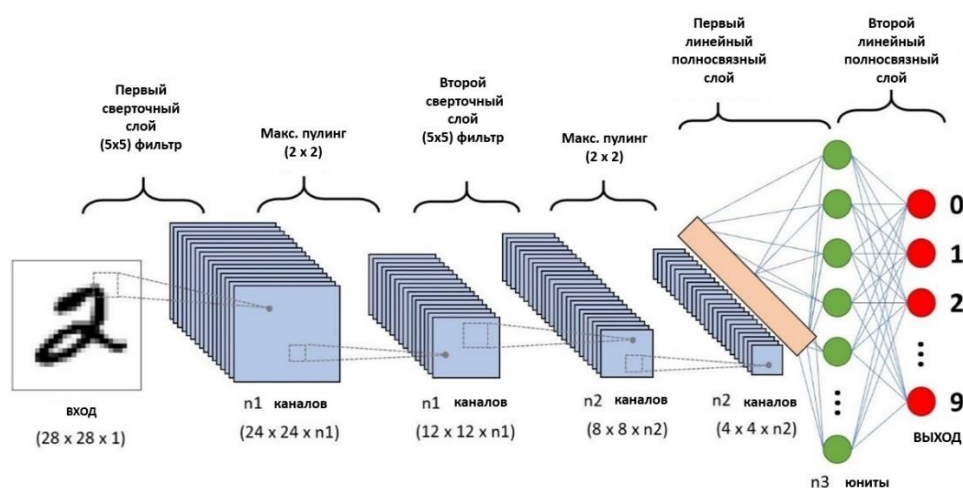


Рисунок 9 – Архитектура сверточной нейронной сети

Главной особенностью свёрточных сетей является то, что они обычно работают именно с матрицами данных, а потому можно выделить особенности, свойственные именно им. Многослойные персептроны работают с векторами, а потому для них нет никакой разницы, находятся ли какие-то точки рядом или на противоположных концах, так как все точки равнозначны и считаются совершенно одинаковым образом. Изображения или же матрицы могут обладать локальной связностью. Например, если речь идёт об изображениях человеческих лиц, то вполне логично ожидать, что точки основных частей лица будут рядом, а не разрозненно располагаться на изображении. Поэтому требовалось найти более эффективные алгоритмы для работы с матрицами данных и ими оказались сверточные сети.

2.2 Изображение «в глазах» компьютера

Изучая CNN, невольно появляется вопрос «Что такое свертка?». Перед тем, как ответить на него, нужно разобраться, как компьютер «видит» какое-то изображение.

На рисунке 10 можно увидеть, как компьютер представляет изображение.

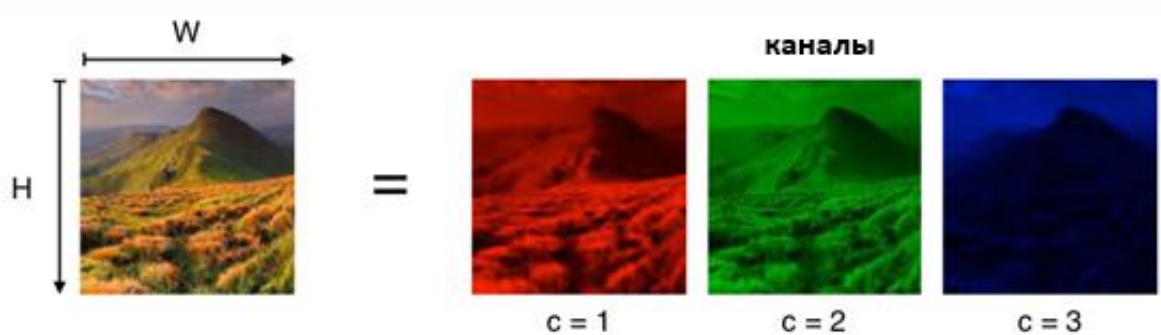


Рисунок 10 – Как компьютер «видит» изображение

Изображения в компьютере представляются в виде пикселей, а каждый пиксель – это значения интенсивности соответствующих цветовых каналов. При этом интенсивность каждого из каналов описывается целым числом от 0 до 255. Чаще всего используются цветные изображения, которые состоят из RGB пикселей – пикселей, содержащих яркости по трём каналам: красному, зелёному и синему. Различные комбинации этих цветов позволяют создать любой из цветов всего спектра. Чтобы хранить информацию о всех пикселях удобнее всего использовать тензор – 3D массив чисел, или, проще говоря, массив матриц чисел.

На рисунке 11 представлено преобразование из изображения в тензор.

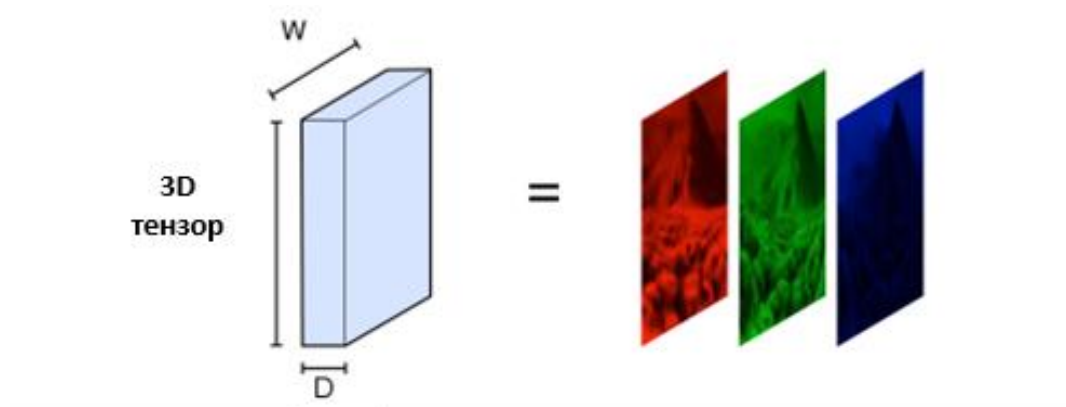


Рисунок 11 – Преобразование из изображения в тензор

На рисунке 12 представлен пример хранения какого-то изображения компьютером.

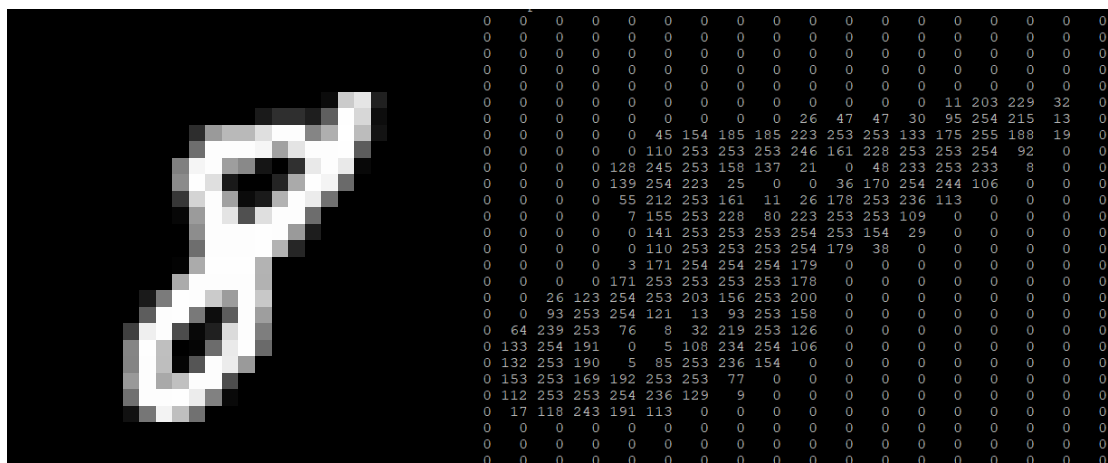


Рисунок 12 – Изображение цифры и его представление в компьютере.

2.3 Свёртка

Слой свёртки, как можно догадаться по названию типа нейронной сети, является самым главным слоем сети. Его основное назначение – выделить признаки на входном изображении и сформировать карту признаков. Карта признаков – это всего лишь очередной тензор (массив матриц), в котором каждая матрица отвечает за какой-нибудь выделенный признак.

Для того, чтобы слой мог выделять признаки, в нём имеются так называемые фильтры (или ядра). Фильтр – квадратная матрица небольшого размера (обычно 3x3 или 5x5), заполненная определенным образом. Чем больше будет таких фильтров – тем больше признаков удастся выделить и тем больше будет глубина каналов у выходного тензора слоя

Данный фильтр поочередно накладывается на изображение начиная с левого верхнего угла и пока не дойдет до правого нижнего угла. На каждом шаге числа в матрице фильтра перемножаются с соответствующими числами на изображении, полученные результаты складываются и записываются в матрицу признака. Лучше понять это можно, посмотрев на рисунок 13.

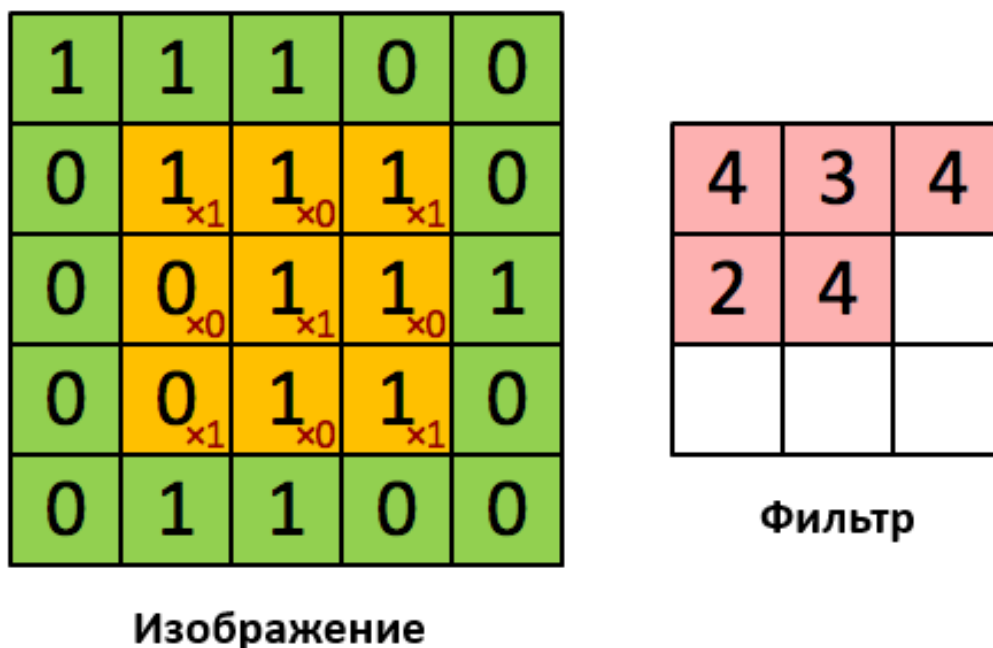


Рисунок 13 – Свертка изображения 5x5x1 с фильтром 3x3x1 для получения признака 3x3x1

На нем желтым цветом изображен фильтр размером 3x3x1, а красные цифры в правом углу – числа матрицы фильтра. Зеленым цветом изображена матрица изображения 3x3x1, а красным полученный признак. Таким образом на 5-ом шаге итерации мы получим:

$$1 * 1 + 0 * 1 + 1 * 1 + 0 * 0 + 1 * 1 + 0 * 1 + 1 * 0 + 0 * 1 + 1 * 1 = 4$$

Полученные в итоге матрицы признаков могут иметь такой же размер, что и исходное изображение, либо размер меньше (как в нашем случае) [8]. Это зависит от заданных размера шага и начального заполнения. В данном репозитории [9] можно найти множество разных GIF-файлов, которые помогут лучше понять, как заполнение и длина шага работают.

2.4Слой подвыборки (пулинга)

Так как сверточные НС используют так же и простую сеть прямого распространения, нам необходимо уменьшить кол-во выходных параметров из слоя свертки и при этом не потерять важную информацию. Для этого используют слои подвыборки.

Данный слой позволяет уменьшить пространство признаков, сохраняя наиболее важную информацию. Существует несколько разных версий слоя пулинга, среди которых максимальный пулинг, средний пулинг и пулинг суммы. Наиболее часто используется именно слой макспулинга.

Примеры различных версий пулинга показаны на рисунке 14:

Исходные значения

6	6	6	6
4	5	5	4
2	4	4	2
2	4	4	2

Макс пулинг

6	6
4	4

Среднеарифм. пулинг

5.25	5.25
3	3

Пулинг суммы

21	21
12	12

Рисунок 14 – Преобразования слоя подвыборки

Действия этого слоя идентичны действиям слоя свертки, только используются другие операции (для макспулинга – берется максимальное число, для среднего пулинга – берется среднее арифметическое от чисел).

2.5 Собственная CNN

Изучив, как устроена сверточная нейронная сеть, можно приступить к написанию собственной.

При разработке архитектуры нейронной сети требуется учитывать, что на вход должны передаваться данные некоторого размера, которые и будут проанализированы НС. В данной работе на вход НС передается информация о тикерах в формате: Массив[N] со значениями High, массив[N] со значениями Low, массив[N] со значениями ЕМА 200, где N – количество исследуемых дней от 40 до 2000, а также передается массив[N] со значениями asset – число отвечающее за наличие данного тикера в портфеле. Все эти 4 массива по N значений образуют матрицу 4×N. Её мы и используем, как входные данные.

Проанализировав данные, поданные на вход НС, она возвращает некий результат своего анализа – выходные данные. При разработке архитектуры важно учесть правильность выходных данных. В данной работе выходными данными является число в диапазоне от -0.5 до 1.5, означающие биржевые заявки трейдеру – take-profit или stop-loss.

Преобразовав матрицу $4 \times N$ в удобный для обычных линейных нейронов формат, используя три слоя свертки с ядрами 2×2 и итоговым выходным количеством сверток 1×64 . Далее с помощью скрытых слоев обычных линейных нейронов анализируем получившиеся свертки. Было использовано 4 скрытых линейных слоя для лучшего анализа.

Полученную архитектуру сети можно увидеть на рисунке 15:

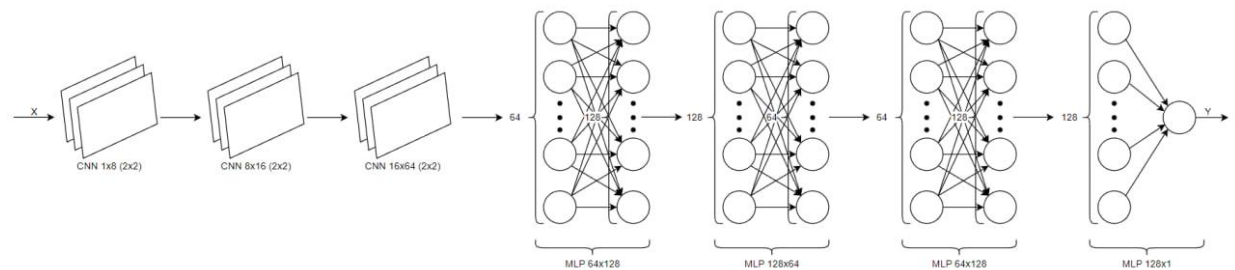


Рисунок 15 – Архитектура НС

Реализация данной архитектуры в коде показана на 12 листинге:

Листинг 12 – Класс нейронной сети

```

1 class ConvNet(nn.Module):
2     def __init__(self):
3         super().__init__()
4
5         self.act = nn.LeakyReLU(0.18) #функция активации
6         self.conv0 = nn.Conv2d(1, 8, 2, stride=1, padding=0)
7         # сверточный слой со сверточным ядром 2 на 2
8         self.conv1 = nn.Conv2d(8, 16, 2, stride=1,
padding=0)
9         self.conv2 = nn.Conv2d(16, 64, 2, stride=1,
padding=0)
10
11         self.adaptivepool = nn.AdaptiveAvgPool2d((1, 1))
12         self.flatten = nn.Flatten()
13         self.linear1 = nn.Linear(64, 128)
14         # линейный слой 64 на 128
15         self.extra_linear_1 = nn.Linear(128, 64)
16         # линейный слой 128 на 64
17         self.extra_linear_2 = nn.Linear(64, 128)
18         # линейный слой 64 на 128

```

```

19         self.linear2 = nn.Linear(128, 1)
20         # линейный слой 128 на 1
21
22
23     def forward(self, x):
24         # функция описывающая одну итерацию обучения НС
25         out = self.conv0(x)
26         out = self.act(out)
27         out = self.conv1(out)
28         out = self.act(out)
29         out = self.conv2(out)
30         out = self.act(out)
31
32         out = self.adaptivepool(out)
33         out = self.flatten(out)
34         out = self.linear1(out)
35         out = self.act(out)
36
37         out = self.extra_linear_1(out)
38         out = self.act(out)
39         out = self.extra_linear_2(out)
40         out = self.act(out)
41
42         out = self.linear2(out)
43         return out

```

2.6 Проведение исследования

Чтобы НС хорошо работала требуется обучить её на данных из всего датасета. Прогнав эти данные повторно множество раз, повысим вероятность более правильного ответа НС. Прогон всего датасета через НС называется – эпоха. Совсем не обязательно в одной эпохе обучать НС всем датасетом за раз, можно разбить его на меньшие части. Такие части называют – батч данных.

Для исследования был составлен пробный датасет, который содержал 200 сущностей для обучения и 50 сущности для тестов. Размер батча равен 25 сущностей. Используемый код для обучения НС на Python:

Сама реализация полного цикла обучения представлена на листинге 13.

Листинг 13 – Код обучающего цикла НС

```

1 for epoch in range(epochs):
2     loss_val = 0
3     acc_val = 0
4     for sample in train_loader:
5         info, lbl = sample['info'], sample['label']
6

```

7	<i># переносим требуемые данные на то устройство, где</i>
8	<i>будем</i>
9	<i>обучать НС (CPU или GPU)</i>
10	<i>info = info.to(device)</i>
11	<i>lbl = lbl.to(device)</i>
12	<i>optimizer.zero_grad()</i>
13	<i>with autocast(use_amp):</i>
14	<i>pred = CNNet(info)</i>
15	<i>loss = loss_fn(pred, lbl)</i>
16	<i># Считаем функцию-потерь</i>
17	<i>scaler.scale(loss).backward()</i>
18	<i>loss_item = loss.item()</i>
19	<i>loss_val += loss_item</i>
20	
21	<i># Обновляем веса</i>
22	<i>scaler.step(optimizer)</i>
23	<i>scaler.update()</i>
24	<i>acc_current = accuracy_v3(pred.cpu().float(),</i>
25	<i>lbl.cpu().float(), epsilon=epsilon)</i>
26	<i>acc_val += acc_current</i>
27	<i>print(f"Epoch : {epoch+1}")</i>
28	<i>print(f"Loss : {loss_val / len(train_loader)}")</i>
29	<i>print(f"Acc : {acc_val /</i>
30	<i>(len(train_loader)*batch_size)})"</i>
	<i>print(f'Full time learning : {time.time() - start_time}')</i>

Исследуемая задача более всего похожа на задачу регрессии, основываясь на этом, максимально логичным решением будет использовать функцию MSELoss (Mean Squared Error Loss) или MAELoss (Mean Absolute Error Loss), как функцию-потерь.

Для определения MSE используется формула:

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2, \quad (4)$$

где MSE – mean squared error (среднеквадратичная ошибка);

n – количество входных значений;

Y_i – точная величина;

\hat{Y}_i – предугаданная величина.

Для определения MAE используется формула:

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}, \quad (5)$$

где MAE – mean absolute error (средняя абсолютная ошибка);

n – количество входных значений;

y_i – точная величина;

\hat{y}_i – предугаданная величина.

После выбора подходящей функции-потерь, необходимо подобрать функцию-оптимизатор. На данный момент одной из самых популярных и эффективных функций-оптимизаторов является Adam (adaptive moment), он отлично подойдет и для этой задачи. Lr (шаг поиска) достаточно будет указать равным 10^{-3} . Отличный пример сравнения разных функций-оптимизаций приведен на данном сайте: [4].

Подобрав все возможные необходимые настройки, можно приступить к тестированию.

Тестирование будем проводить и на CNN архитектуре, и на MLP. В качестве функций-потерь будем тестировать MSELoss и MAELoss. Также рассмотрим разное количество эпох (50, 100, 200, 400, 800, 1600, 3200, 6400, 10000). Все предложенные параметры нужны нам, чтобы четко понять как архитектура НС, функция-потерь и количество эпох влияют на итоговый результат.

Полученные результаты представлены ниже. График красного цвета – требуемый (истинный) результат. График синего цвета – результат, предугаданный НС. По оси X – номера сущностей из тестового датасета. По оси Y – «stop loss».

На рисунке 16 представлены результаты 50 эпох.

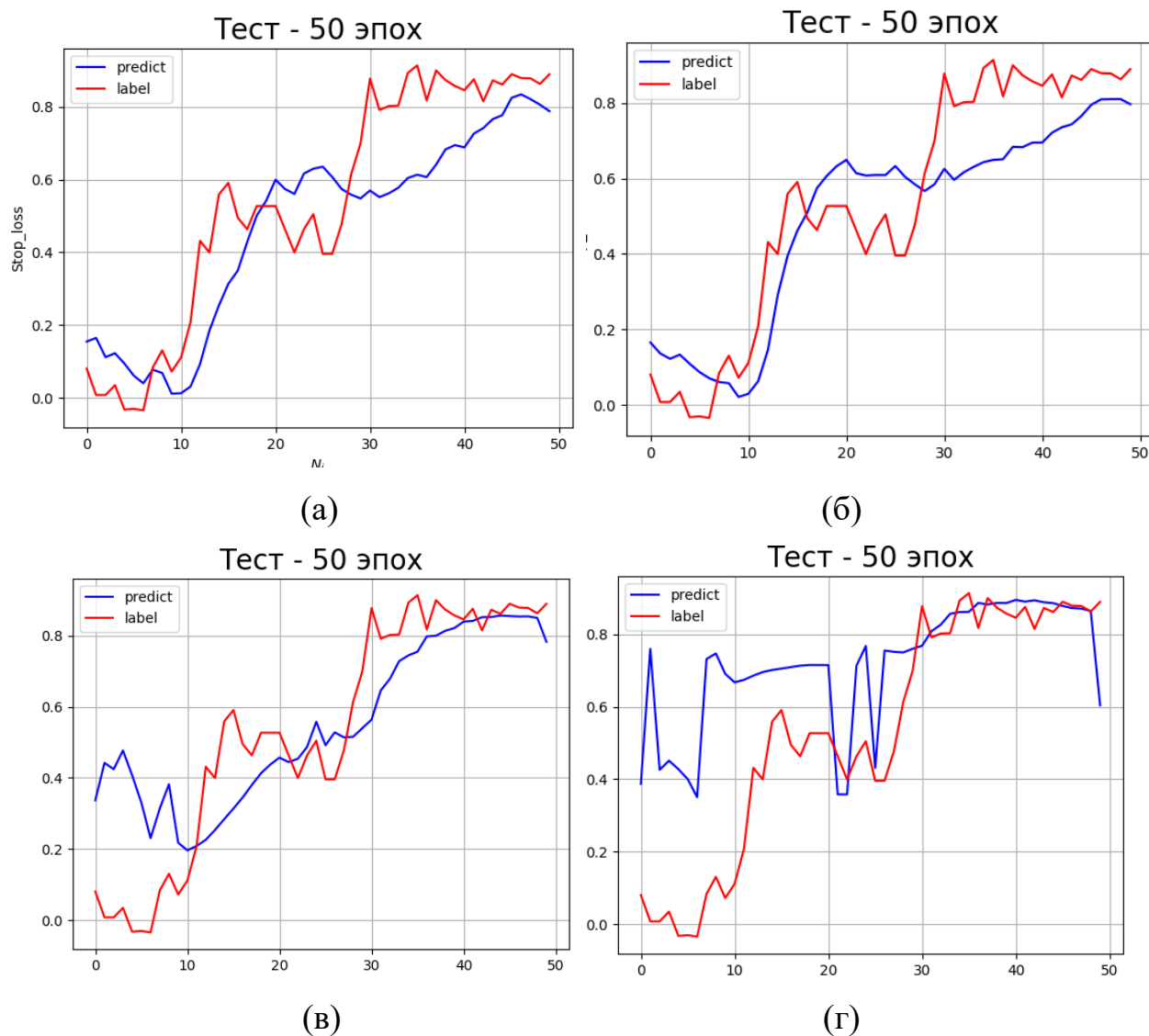


Рисунок 16 – Тест с 50 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 17 представлены результаты 100 эпох.

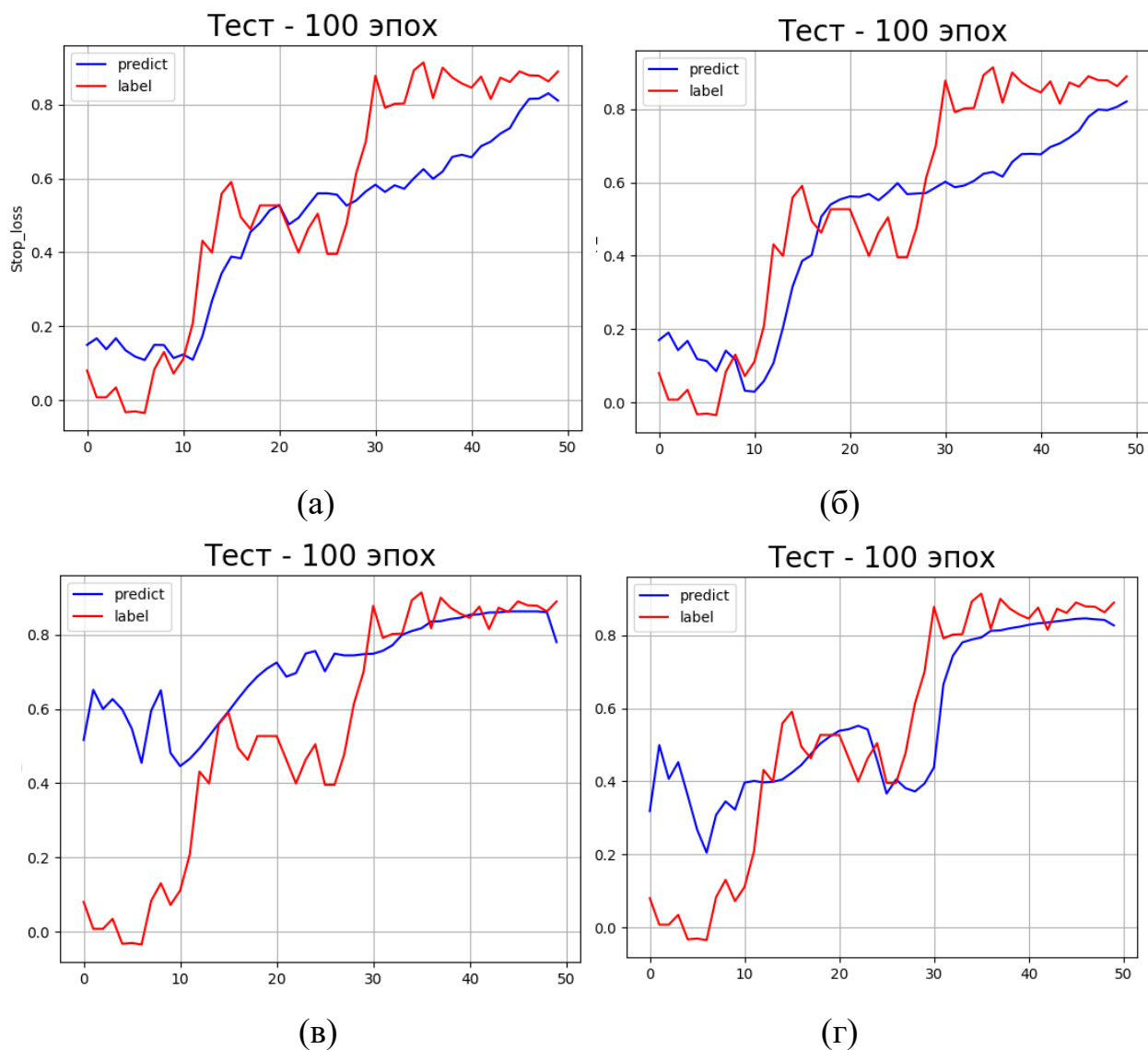


Рисунок 17 – Тест с 100 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 18 представлены результаты 200 эпох.

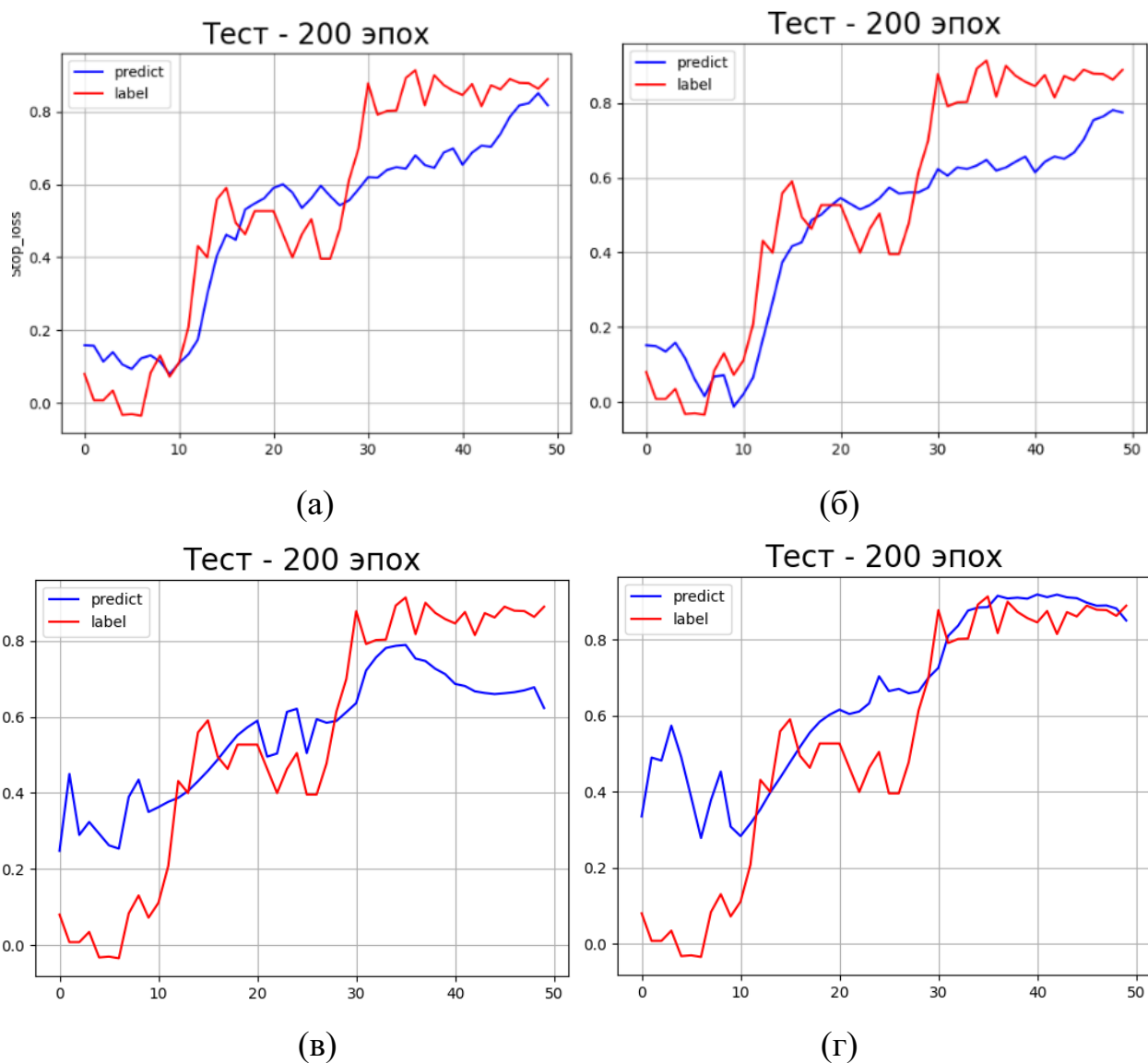
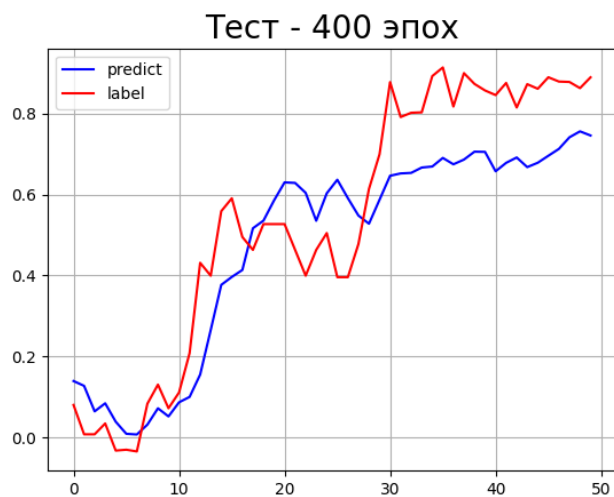
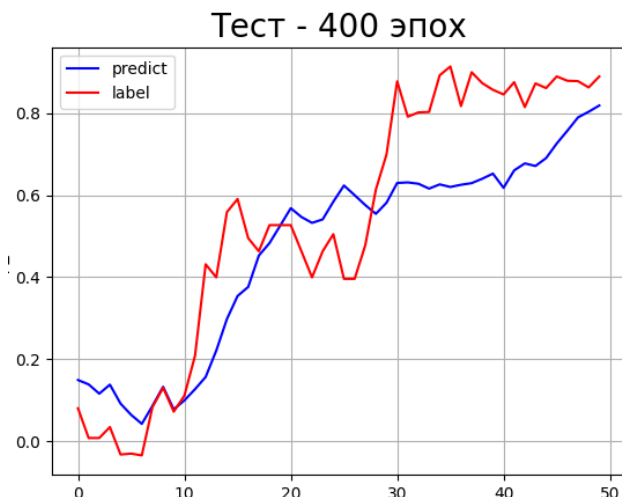


Рисунок 18 – Тест с 200 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

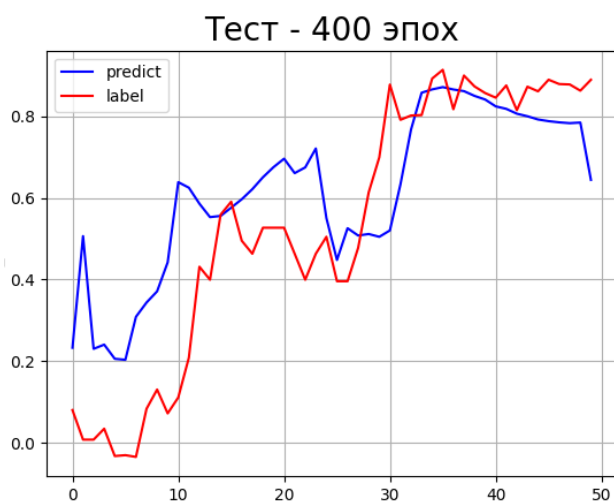
На рисунке 19 представлены результаты 400 эпох.



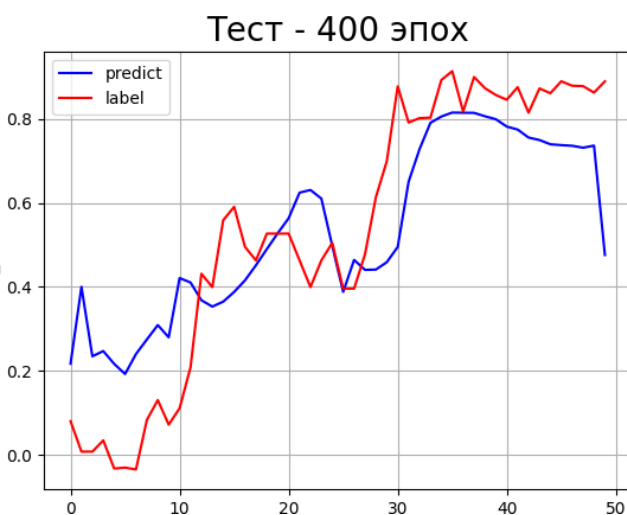
(а)



(б)



(в)



(г)

Рисунок 19 – Тест с 400 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 20 представлены результаты 800 эпох.

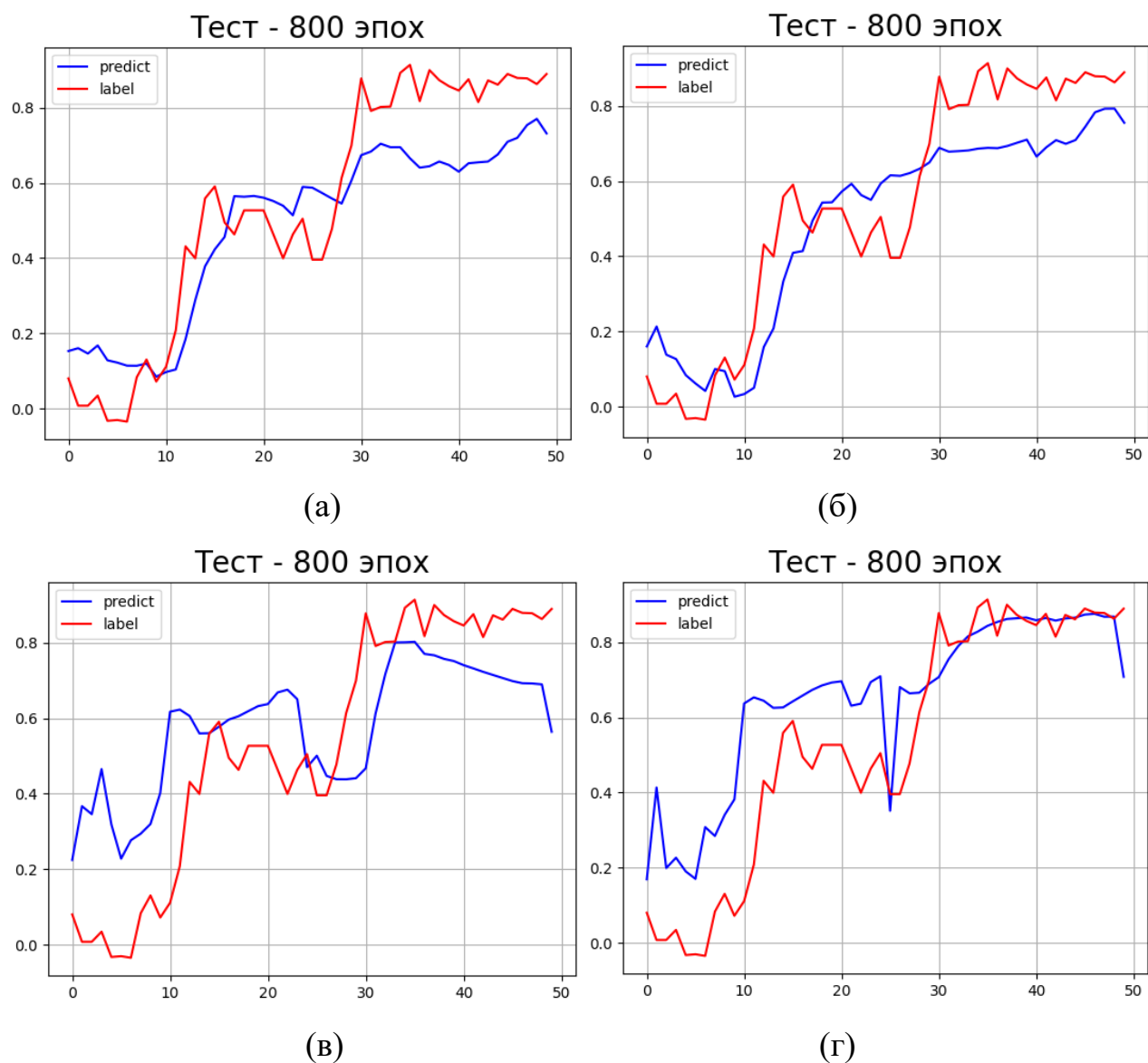


Рисунок 20 – Тест с 800 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 21 представлены результаты 1600 эпох.

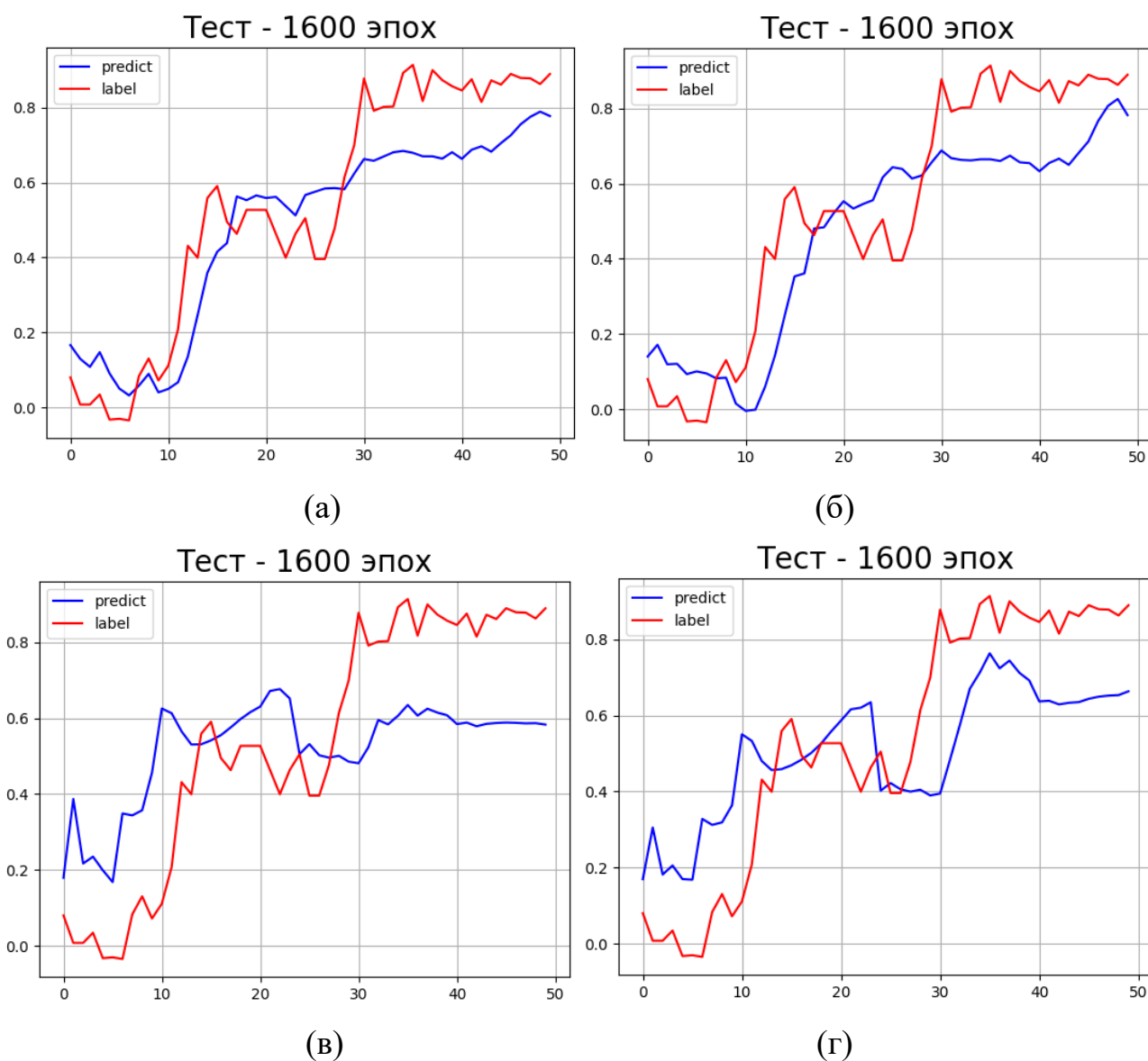


Рисунок 21 – Тест с 1600 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 22 представлены результаты 3200 эпох.

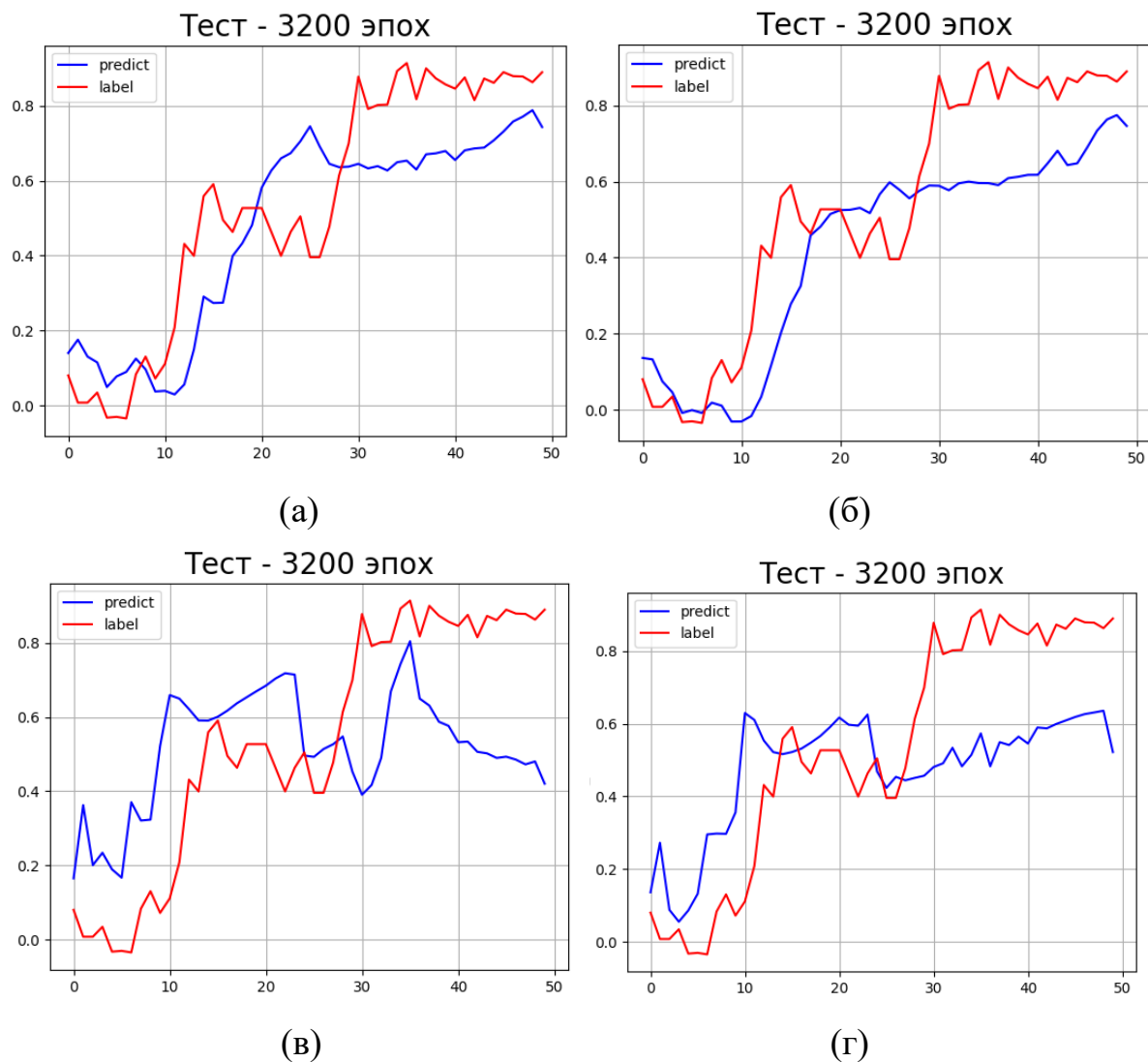


Рисунок 22 – Тест с 3200 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 23 представлены результаты 6400 эпох.

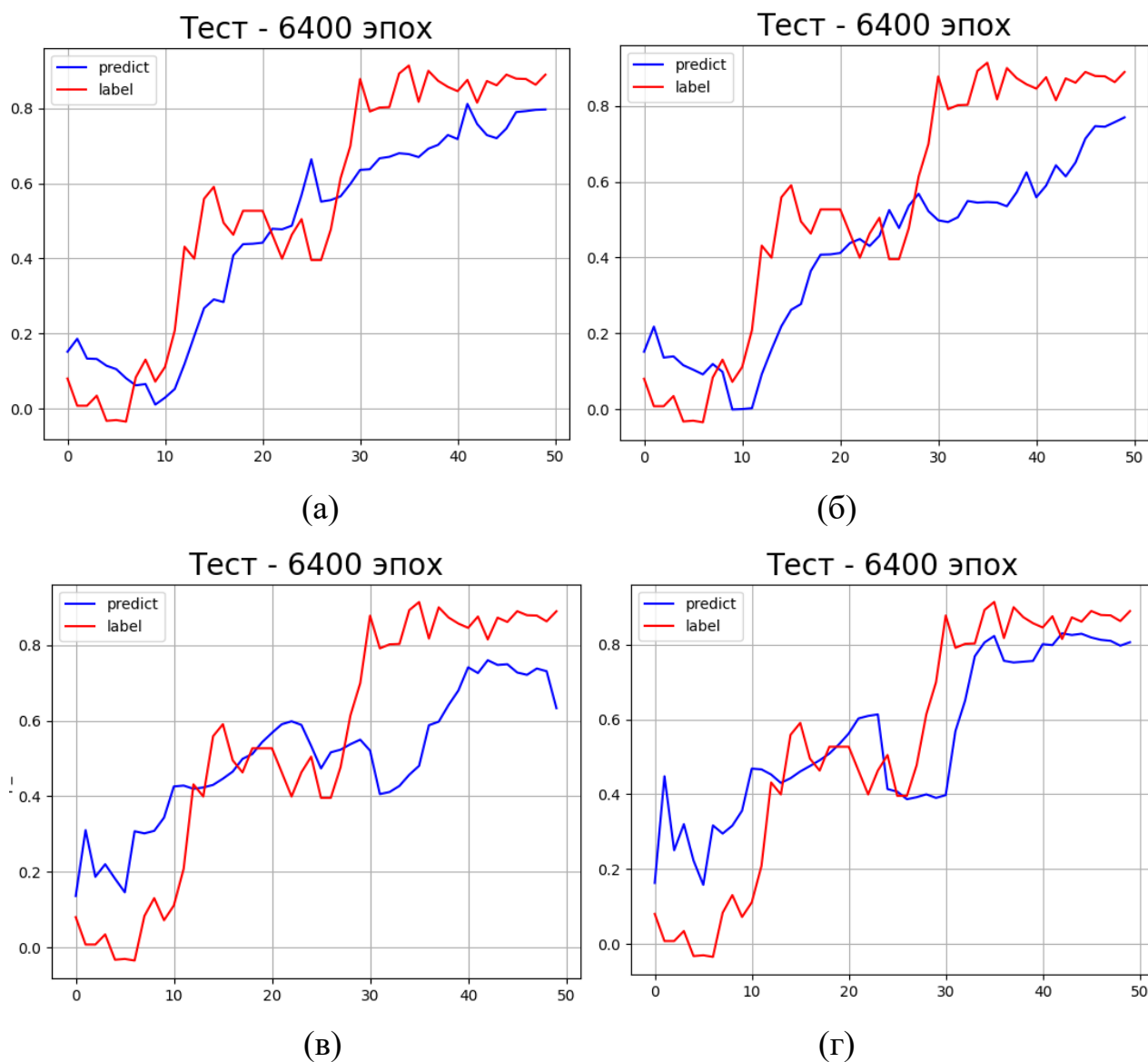


Рисунок 23 – Тест с 6400 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

На рисунке 24 представлены результаты 10000 эпох.

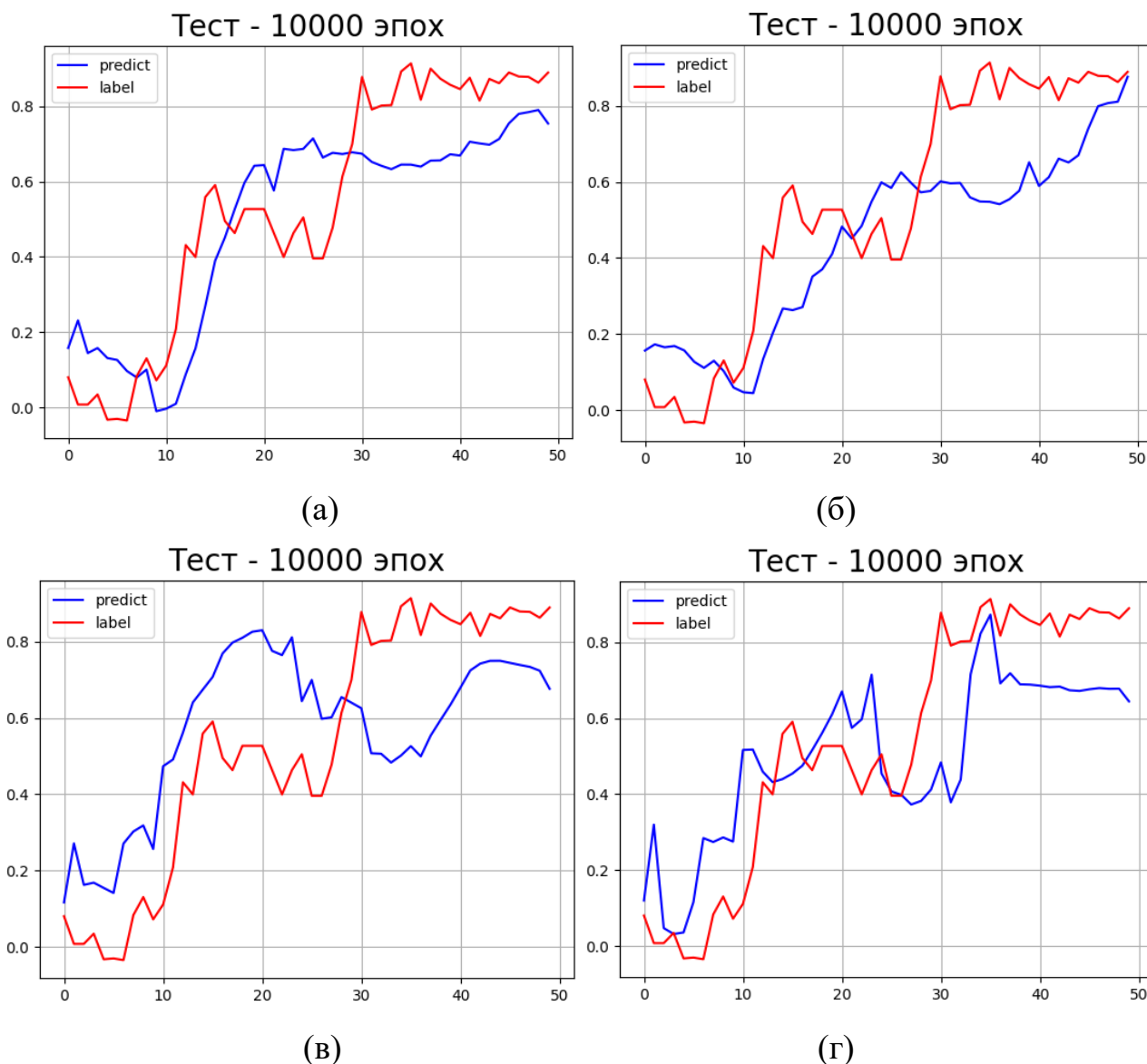


Рисунок 24 – Тест с 10000 эпохами (слева – MSE, справа – MAE; сверху – MLP, Снизу – CNN)

Данные графики описывают результаты «предугадывания» НС в результате работы с тестовым датасетом (этот набор информации НС еще не видела и обучалась она не на нем).

По таблице 2 и 3 можно увидеть, что время обучения НС от количества эпох зависит линейно.

Таблица 2 Зависимость времени обучения НС от количества эпох при использовании MSELoss

MLP	Time	CNN	Time
50	1.00	50	3.484
100	2.00	100	4.673
200	3.586	200	8.708
400	6.720	400	13.0394
800	13.572	800	25.757
1600	27.80	1600	48.983
3200	56.043	3200	98.291
6400	195.29	6400	193.839
10000	180.47	10000	299.83

Таблица 3 Зависимость времени обучения НС от количества эпох при использовании MAELoss

MLP	Time	CNN	Time
50	1.00	50	3.895
100	2.00	100	4.762
200	3.455	200	8.210
400	7.208	400	13.410
800	14.6937	800	25.776
1600	28.3952	1600	51.802
3200	58.797	3200	102.44
6400	117.310	6400	202.842
10000	187.9521	10000	317.748

По остальным метрикам, которые представлены с точными значениями в файле «КР.xlsx», можно понять, что MLP обучается в 1.5 раза быстрее, чем CNN и имеет более точный результат и на этапе обучения, и на этапе тестирования. Также можно увидеть, что отсутствует сильная разница между использованием MSELoss или MAELoss. Обе функции справляются со своей задачей.

2.7 Итоги по CNN

В ходе выполнения данной работы были выполнены следующие задачи:

- 1) Рассмотрено, что такое сверточная нейронная сеть.
- 2) Чем сверточная сеть отличается от простого линейного перцептрона

- 3) Описана работа по настройке и подбору нужных функции-оптимизации, функции-потерь, функции-активации и других настроек архитектуры НС.
- 4) Были проведены анализы работы сверточной НС при разных параметрах.
- 5) Было проведено сравнение с работой линейного перцептрона при разных параметрах.

Результаты экспериментов оказались достаточно «живые» и говорят о том, что сверточные нейронные сети способны анализировать не только изображения, но и большие матрицы данных. Однако биржа является предметом с очень сложными зависимостями и закономерностями и CNN не имеет возможностей точно предугадать следующее поведение биржи.

Оптимизация параметров модели и правильный выбор архитектуры CNN существенно влияют на достижение высокой точности классификации. Также важным аспектом является качество предварительной обработки данных, включая масштабирование, аугментацию и нормализацию, что дополнительно улучшает способность модели к обобщению.

3 Разработка архитектуры ViT

Один из важнейших инструментов машинного обучения — трансформеры. Популярность трансформеров взлетела до небес в связи с появлением больших языковых моделей вроде ChatGPT, GPT-4 и LLama. Эти модели созданы на основе трансформерной архитектуры и демонстрируют отличную производительность в понимании и синтезе естественных языков.

Помимо понимания и синтеза естественных языков, НС с трансформерной архитектурой имеют большой успех и спрос во многих других доменах машинного обучения, таких как распознавание человеческой речи, анализ изображений и видео.

Зрительные трансформеры (ViT) представляют собой одну из последних инноваций среди трансформерных архитектур НС. В отличие от сверточных нейронных сетей (CNN), которые используют свертки для извлечения признаков, ViT разбивают изображение или матрицу на патчи (небольшие куски) и обрабатывают их как последовательности, что позволяет модели эффективно захватывать глобальные взаимосвязи и контекст.

Их способность эффективно работать с временными рядами и большим объемом рыночных данных открывает новые возможности для предсказания ценовых движений и оптимизации стратегий управления активами. Введение зрительных трансформеров в процесс технического анализа может позволить инвесторам и трейдерам более точно оценивать рыночные тренды, минимизировать риски и принимать обоснованные решения, что существенно повышает общий успех стратегий.

На данном этапе работе рассмотрено, что такое трансформерная нейронная сеть и, в частности, визуальный трансформер. Описана работа по настройке и подбору нужных нам функций-оптимизации, функций-потерь и функций активации, а также различных гиперпараметров. Описана работа по коррективке датасета на основе данных о тикере MSFT. Также в данной работе были проведены тесты с разными настройками и проанализированы результаты,

полученные на выходе. Было проведено сравнение с результатами работы сверточной нейронной сети.

3.1 Устройство визуальной трансформерной НС

На рисунке 25 схематично показано внутреннее устройство простейшего визуального трансформера [10]:

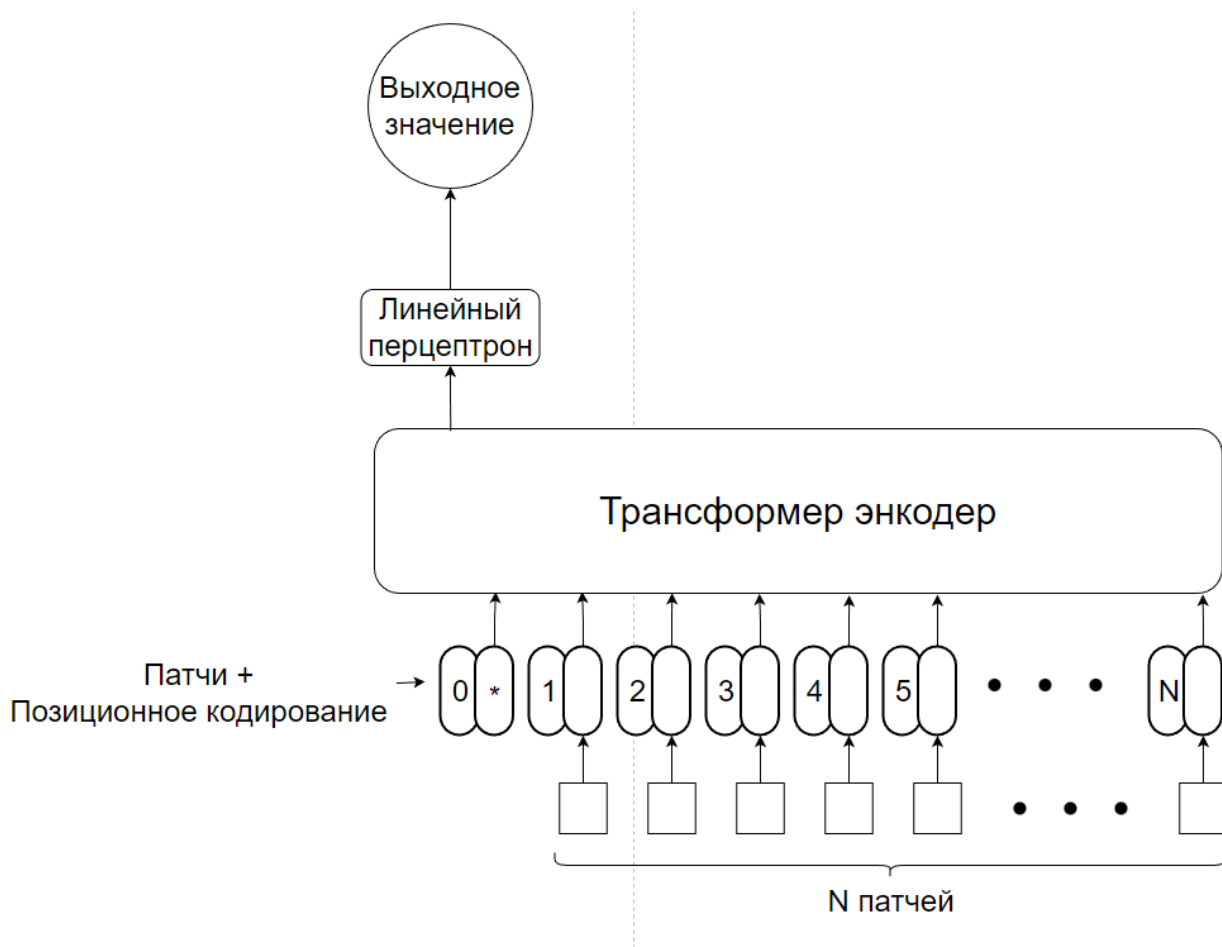


Рисунок 25 – Архитектура ViT НС

Перед реализацией архитектуры трансформера на практике, стоит узнать, что такое трансформер. Трансформер — это такой вид нейросетевой архитектуры, который хорошо подходит для обработки последовательностей данных. Пожалуй, самый популярный пример таких данных это предложение, которое можно считать упорядоченным набором слов. Однако, также часто используются трансформеры для обработки изображений и матриц данных.

Трансформеры создают цифровое представление каждого элемента последовательности, инкапсулируют важную информацию о нём и окружающем

его контексте. Получившиеся представления затем можно передать в другие нейросети, которые воспользуются этой информацией для решения разных задач, в том числе для синтеза и классификации. Создавая такие информативные представления, трансформеры помогают последующим нейросетям лучше понять скрытые паттерны и взаимосвязи во входных данных. И поэтому они лучше синтезируют последовательные и взаимосвязанные результаты.

Главное преимущество трансформеров заключается в их способности обрабатывать длительные зависимости в последовательностях. Кроме того, они очень производительны, могут обрабатывать последовательности параллельно. Это особенно полезно в задачах вроде машинного перевода, анализа настроений и синтеза текста.

Для лучшего понимания, будем рассматривать ViT последовательно по рисунку 25.

3.2 Что поступает в трансформер

Прежде чем подать данные в трансформер, нужно сначала преобразовать их в последовательность векторов (токенов) — набор целых чисел, представляющих входные данные. Будем рассматривать сценарий использования трансформера для конкретно нашей задачи.

Чтобы получить вектор данных, возьмем нашу матрицу значений (окно дневных свечей размером N) и разобьем его на более мелкие кусочки — патчи. В одном патче будет содержаться информация об одной дневной свече. И наш патч будем записывать как вектор.

Получив последовательность целых чисел (вектор), представляющих входные данные, мы можем превратить их в эмбединги — это способ представления информации, облегчающий её обработку алгоритмами машинного обучения. Эмбединги передают смысл токенов в сжатом формате, представляя информацию в виде последовательности чисел. Сначала они синтезируются как случайная последовательность, а значимое представление

формируется во время обучения. Однако у эмбеддингов есть наследственное ограничение: они не учитывают контекст, в котором синтезировались токены.

В зависимости от задачи, при превращении токенов в эмбеддинги нам может потребоваться сохранить порядок токенов. Это особенно важно для нашей задачи, иначе НС может потерять связь между днями. Чтобы этого не допустить, мы применяем к эмбеддингам позиционное кодирование.

Есть разные способы это сделать, но основная идея в том, что у нас есть ещё один набор эмбеддингов, представляющих положение каждого токена во входной последовательности. Этот второй набор комбинируется с эмбеддингами токенов [11]. Такой процесс схематично изображен на рисунке 26.

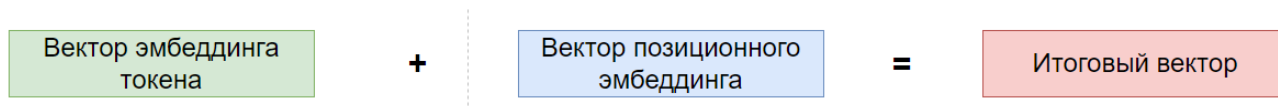


Рисунок 26 – Получаем итоговый вектор эмбеддинга.

Другая сложность в том, что у токенов могут быть разные значения в зависимости от соседних токенов. Чтобы легче было понять, рассмотрим сценарий использования трансформера для обработки естественного языка:

- Андрей не любит арбуз, **он** слишком сладкий.
- Андрей не любит арбуз, **он** любит дыню.

Здесь слово «он» используется в двух абсолютно разных контекстах, поэтому имеют разные значения. В первом предложении слово «он» подразумевает арбуз. Во втором же – Андрей. Трансформер решает эту проблему с помощью механизма «Attention».

3.3 Механизм «Attention»

На рисунке 27 схематично изображено подробное устройство простейшего трансформера [12]:

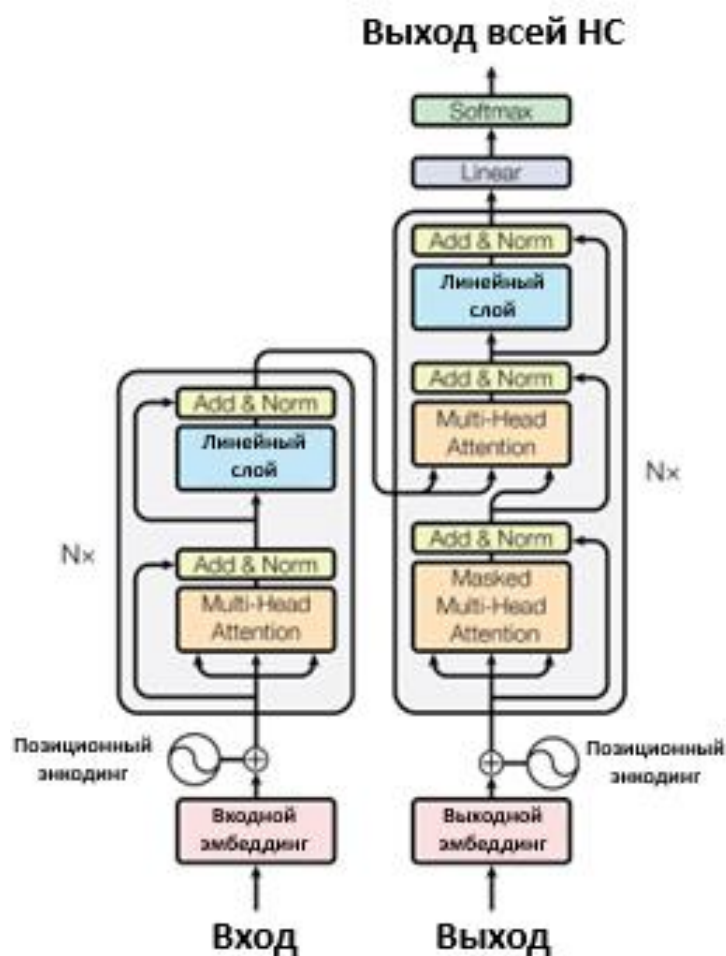


Рисунок 27 – Архитектура трансформерной НС.

Рассмотрим внимательно эту архитектуру и заметим, что большое количество блоков в ней отведены под «Attention».

Пожалуй, самый важный механизм в трансформерной архитектуре — это «Attention» (на рисунке 27 – светло-оранжевые блоки). Он позволяет нейросети понять, какая часть входной последовательности наиболее релевантна задаче. Механизм «Attention» определяет для каждого токена последовательности, какие другие токены необходимы для его понимания в данном контексте. Прежде чем мы перейдем к тому, как это реализовано в трансформере, давайте сначала разберемся, чего пытается добиться механизм «Attention».

Этот механизм можно представить как метод, который заменяет каждый эмбединг токена на эмбединг, содержащий информацию о соседних токенах, вместо использования одинакового эмбединга для каждого токена вне зависимости от контекста. Если бы мы знали, какие токены релевантны

текущему, то узнать его контекст можно с помощью средневзвешенного — или, в общем случае, линейной комбинации — этих эмбедингов.

На практике мы часто параллельно запускаем несколько таких блоков «Attention» (self-attention), чтобы трансформер одновременно обрабатывал разные части входной последовательности — это называют multi-head attention. Идея проста: выходы нескольких независимых блоков self-attention конкатенируются и передаются через линейный слой. Он позволяет модели комбинировать контекстуальную информацию из каждого блока «Attention».

Мы поверхностно рассмотрели, что пытается добиться механизм «Attention». Давайте теперь рассмотрим на листинге 14, как именно это реализовано в коде.

Листинг 14 – Класс слоя «Attention»

```
1 class MultiHeadAttentionLayer(nn.Module):
2     def __init__(self, hid_dim, n_heads, dropout, device):
3         super().__init__()
4         assert hid_dim % n_heads == 0
5         self.hid_dim = hid_dim
6         self.n_heads = n_heads
7         self.head_dim = hid_dim // n_heads
8         self.fc_q = nn.Linear(hid_dim, hid_dim)
9         self.fc_k = nn.Linear(hid_dim, hid_dim)
10        self.fc_v = nn.Linear(hid_dim, hid_dim)
11        self.fc_o = nn.Linear(hid_dim, hid_dim)
12        self.dropout = nn.Dropout(dropout)
13        self.scale =
14        torch.sqrt(torch.FloatTensor([self.head_dim])).to(device)
15
16        def forward(self, query, key, value, mask = None):
17
18            batch_size = query.shape[0]
19
20            #query = [batch size, query len, hid dim]
21            #key = [batch size, key len, hid dim]
22            #value = [batch size, value len, hid dim]
23            Q = self.fc_q(query)
24            K = self.fc_k(key)
25            V = self.fc_v(value)
26            #Q = [batch size, query len, hid dim]
27            #K = [batch size, key len, hid dim]
28            #V = [batch size, value len, hid dim]
29            Q = Q.view(batch_size, -1, self.n_heads, self.head_dim)
30            .permute(0, 2, 1, 3)
31            K = K.view(batch_size, -1, self.n_heads, self.head_dim)
32            .permute(0, 2, 1, 3)
33            V = V.view(batch_size, -1, self.n_heads, self.head_dim)
34            .permute(0, 2, 1, 3)
35            #Q = [batch size, n heads, query len, head dim]
```

```

32         #K = [batch size, n heads, key len, head dim]
33         #V = [batch size, n heads, value len, head dim]
34         energy = torch.matmul(Q, K.permute(0, 1, 3, 2))
        / self.scale
35         #energy = [batch size, n heads, query len, key len]
36         if mask is not None:
37             energy = energy.masked_fill(mask == 0, -1e10)
38         # [batch size, 1, trg len, trg len]
39         attention = torch.softmax(energy, dim = -1)
40         #attention = [batch size, n heads, query len, key len]
41         x = torch.matmul(self.dropout(attention), V)
42         #x = [batch size, n heads, query len, head dim]
43         x = x.permute(0, 2, 1, 3).contiguous()
44         #x = [batch size, query len, n heads, head dim]
45         x = x.view(batch_size, -1, self.hid_dim)
46         #x = [batch size, query len, hid dim]
47         x = self.fc_o(x)
48         #x = [batch size, query len, hid dim]
49         return x, attention

```

3.4 Полносвязная нейронная сеть

После блока «Attention», если посмотреть на рисунок 27, идет блок с полносвязной нейронной сетью (блок голубого цвета). Этот блок включает в себя самую простую нейронную архитектуру MLP (похожую мы рассматривали в первой части данной работы), обрабатывающую представления, полученные на выходе из слоя «Attention».

Этот слой нужен, чтобы среди связей слов в предложении найти самые нужные, найти какие-то закономерности и просто добавляет «свободные» нейроны для «мыслительных» процессов.

Реализация этого блока в коде представлена на листинге 15.

Листинг 15 – Класс полносвязной НС

```

1     class PositionwiseFeedforwardLayer(nn.Module):
2         def __init__(self, hid_dim, pf_dim, dropout):
3             super().__init__()
4
5             self.fc_1 = nn.Linear(hid_dim, pf_dim)
6             self.fc_2 = nn.Linear(pf_dim, hid_dim)
7
8             self.dropout = nn.Dropout(dropout)
9
10        def forward(self, x):
11
12            #x = [batch size, seq len, hid dim]
13
14            x = self.dropout(torch.relu(self.fc_1(x)))
15

```

16	#x = [batch size, seq len, pf dim]
17	
18	x = self.fc_2(x)
19	
20	#x = [batch size, seq len, hid dim]
21	
22	return x

3.5 Блок энкодер

После того, как мы поняли устройство блоков «внимания» и полносвязной НС, можно заметить по рисунку 27, что вместе эти блоки образуют один большой блок под названием «Transformer Encoder».

Именно такой блок нам нужен для нашего ViT и его реализация в коде представлена на листинге 16.

Листинг 16 – Класс энкодера

1	class TransformerEncoderBlock(nn.Module):
2	"""Creates a Transformer Encoder block."""
3	
4	# 2. Initialize the class with hyperparameters from Table 1
	and Table 3
5	def __init__(self,
6	embedding_dim: int = 4, # Hidden size D from
	Table 1 for ViT-Base
7	num_heads: int = 10, # Heads from Table 1 for
	ViT-Base
8	mlp_size: int = 128, # MLP size from Table 1 for
	ViT-Base
9	mlp_dropout: float = 0.1, # Amount of dropout
	for dense layers from Table 3 for ViT-Base
10	attn_dropout: float = 0.1): # Amount of dropout
	for attention layers
11	super().__init__()
12	
13	# 3. Create MSA block (equation 2)
14	self.msa_block=
15	MultiheadSelfAttentionBlock(embedding_dim=embedding_dim,
16	num_heads=num_heads, attn_dropout=attn_dropout)
17	
18	# 4. Create MLP block (equation 3)
19	self.mlp_block = MLPBlock(embedding_dim=embedding_dim,
20	mlp_size=mlp_size,
21	dropout=mlp_dropout)
22	
23	# 5. Create a forward() method
24	def forward(self, x):
25	# 6. Create residual connection for MSA block (add the
	input to the output)
26	x = self.msa_block(x) + x
27	

28	# 7. Create residual connection for MLP block (add the input to the output)
29	x = self.mlp_block(x) + x
30	
31	return x

Таким образом мы получили готовую НС, которая показана на рисунке 32. Можно приступать к тестированию.

3.6 Проведение исследования

Чтобы НС хорошо работала требуется обучить её на данных из всего датасета. Прогоняя эти данные повторно множество раз, мы повышаем вероятность более правильного ответа НС.

Для исследования использован датасет, который содержит 200 сущностей для обучения и 50 сущности для тестов. Размер бача равен 25 сущностей.

В результате проведения прошлых тестов было выявлено, что функции-оптимизаторы MSE и MAE показывают схожую эффективность, поэтому было принято решение использовать для исследования только одну функцию, а именно MAE.

В процессе тестирования изменялись различные гиперпараметры НС, такие как: количество эпох, количество блоков энкодера, количество нейронов в полносвязном блоке, параметры дропаута различных блоков и количество голов.

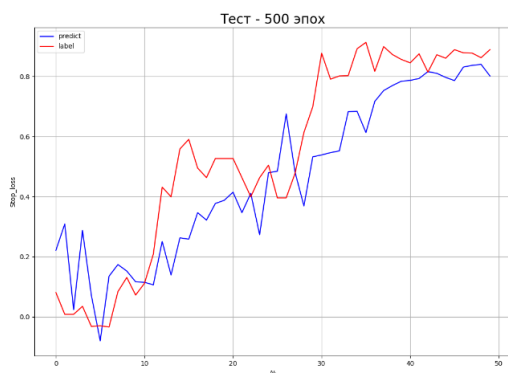
Полные результаты исследования приведены в файле на сайте [13].

На таблице 4 представлена часть результатов, полученных в результате изменения параметра дропаута после блока внимания.

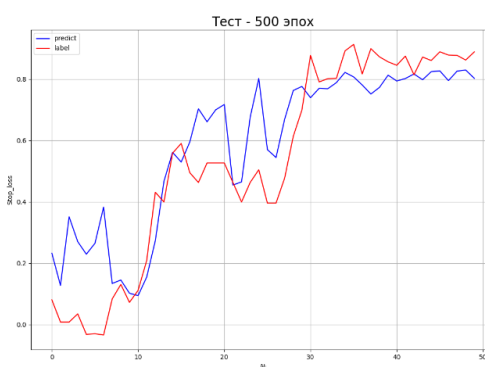
Таблица 4 Результаты исследования на этапе тестирования модели

	Relation	Average_label	Average_prediction	Standard_deviation	Error, %	Max_error, %
DO_add	196.614	0.5352	0.4636	0.1630	30.457	3690.62
	170.80	0.5352	0.3704	0.2938	54.888	2930.36
	414.69	0.5352	0.5169	0.222	41.500	8515.25
	286.04	0.5352	0.587	0.148	27.659	4212.76

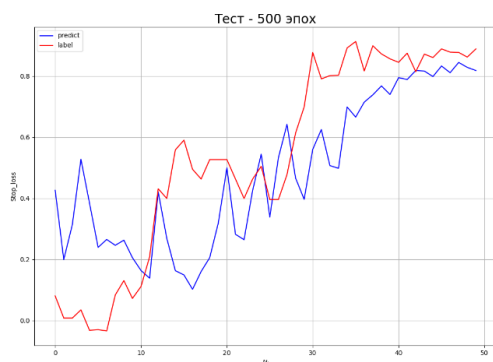
На рисунке 28 изображены графики наилучших результатов, полученных в ходе исследования:



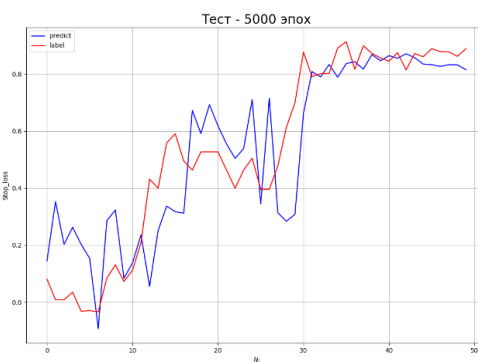
(а)



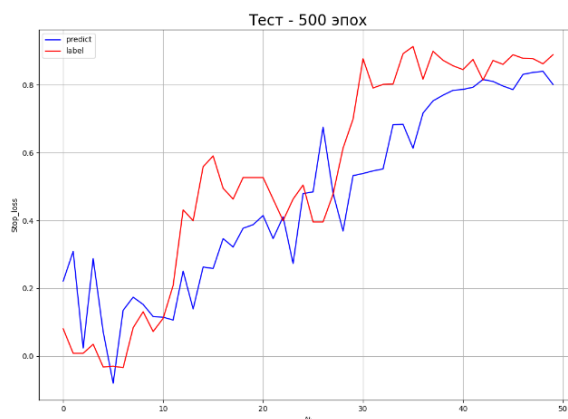
(б)



(в)



(г)



(д)

Рисунок 28 – Графики полученных результатов при различных настройках

График красного цвета – требуемый (истинный) результат. График синего цвета – результат, предугаданный НС. По оси X – номера сущностей из тестового датасета. По оси Y – “stop loss”.

3.7 Итоги по ViT

В ходе выполнения данной работы были выполнены следующие задачи:

- 1) Рассмотрено, что такое трансформерная нейронная сеть;

- 2) Чем трансформер отличается от сверточной НС;
- 3) Были проведены анализы работы трансформерной НС при разных параметрах;
- 4) Было проведено сравнение с работой сверточной НС при разных параметрах.

Результаты экспериментов говорят о том, что трансформерные сети могут обрабатывать не только текст, но и большие матрицы данных. Однако, технический анализ биржевых фондовых котировок является действительно сложно предсказуемым процессом.

Оптимизация параметров модели и правильная настройка гиперпараметров существенно влияют на достижение высокой точности классификации. Также важным аспектом является качество предварительной обработки данных, включая масштабирование, аугментацию и нормализацию, что дополнительно улучшает способность модели к обобщению.

По полученным данным исследования, в сравнение с результатами сверточной НС, можно сказать, что конкретно для данной задачи визуальный трансформер дает больше неточностей (похожих на шум) на итоговом графике. Также само обучение ViT занимает значительно больше времени для достижения схожей с CNN точностью. Возможно, если увеличить размер датасета, получится достигнуть большей точности и эффективности от ViT, но на данный момент результаты CNN кажутся более значительными.

4 Метод дообучению

В современном мире доступ к знаниям обширен и большинство людей обучаются чему-то новому каждый день. Новые знания помогают находить новые пути решения проблемы, помогают принимать решения основываясь на похожих ситуациях в прошлом и просто делают человека многогранным. Нейронные сети берут свое начало в биологии и анатомии человека и имеют внутри себя многие схожие системы и архитектуры. Этапы работы, выполненные до этого, использовали только простой (конечный) метод прямого обучения нейронной сети, на данном же этапе, основываясь на примере реального человека, который получает новые знания каждый день, будет проведено исследование с использованием так называемого цикла дообучений НС. Каждый новый день у брокера появляется новая сущность ДС с информацией о выставленных им take-profit и stop-loss. Этой новой информации и будем дообучать НС.

Задачей данного этапа является разработка и проведение исследования модуля дообучения для CNN и ViT. Также нахождение оптимальных настроек для такого метода.

4.1 Разработка модуля дообучения

Добавление в проект дообучения, предположительно, поможет увеличить точность предугаданных НС значений. Чтобы реализовать такой модуль, нужно разобраться, как будет работать наше дообучение. В данной работе я буду использовать такой вариант, который показан схематично на рисунке 29 [14]:

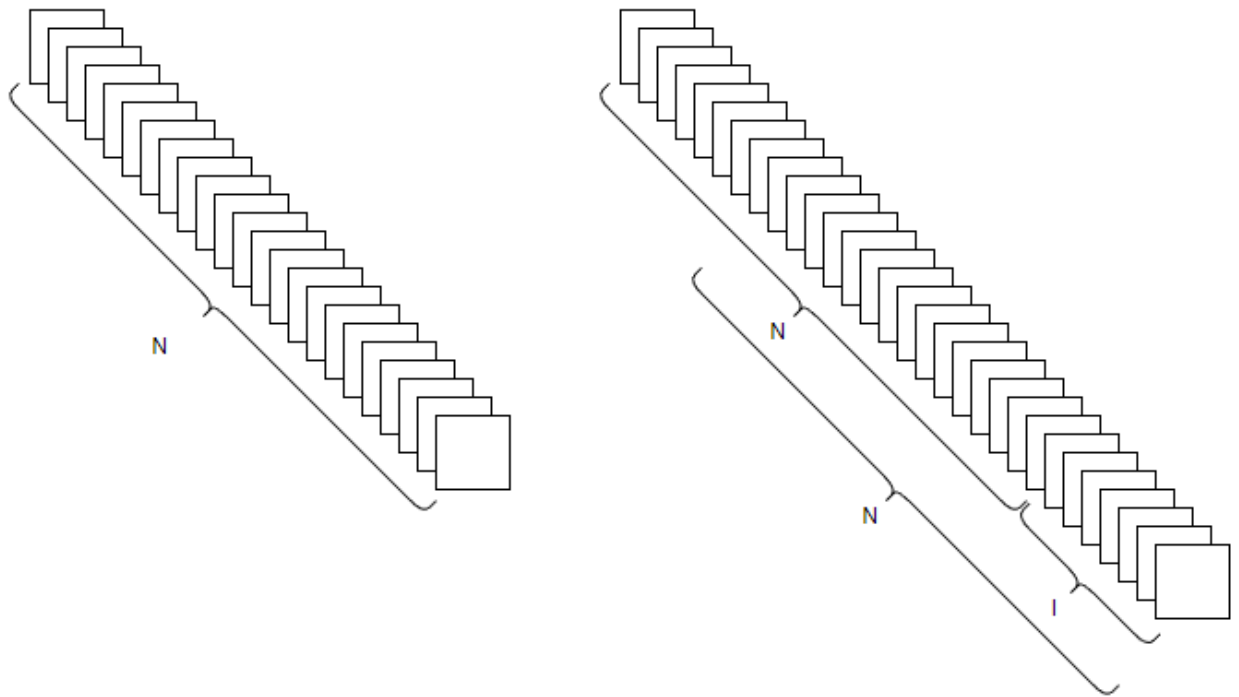


Рисунок 29 – Вариант дообучения.

Пусть у нас имеется ДС состоящий из N сущностей и начинающийся с 0-ой позиции. Обучим нашу НС на этом ДС используя обычное (конечное) обучение. После первого дня использования НС мы получим от брокера ещё одну сущность в наш ДС (того $N+1$). Не будем терять возможность получения нейронной сетью новой информации и дообучим её используя ДС, состоящий также из N сущностей, но уже начинающийся с 1 позиции (напомню, что у нас, на данный момент, всего $N+1$ сущность всего). На i -ый день у нас уже будет $N+I$ сущностей и мы опять дообучим НС ДС из N сущностей но с i -ой позиции и так далее.

Такой метод дообучения позволит нам обучить сперва НС всем данным, что у нас были до начала работы брокера, а далее дообучать актуальными данными на основе уже поведения брокера.

4.2 Реализация класса создания датасетов

Как можно заметить, для такого обучения, нам часто придется создавать ДС, начинающиеся с разных позиций, поэтому удобно будет вынести эту функциональность в отдельный класс, как это реализовано в листинге 17.

Листинг 17 – Класс создания датасетов

```
1 class Dataloader():
2     def __init__(self):
3         options_path = 'config.yml'
4         with open(options_path, 'r') as options_stream:
5             options = yaml.safe_load(options_stream)
6             self.dataset_options = options.get('dataset')
7             data_path = self.dataset_options.get('data_file_name')
8             EMA_N = self.dataset_options.get('EMA')
9             self.data = load_data(data_path, EMA_N).data
10
11     def get_dataloader(self, start: int | None = None, stop: int
12 | None = None, additional: bool = False, test: bool = False) ->
13 torch.utils.data.DataLoader:
14     """
15     Метод создает даталоадер с нужными настройками в
16 зависимости от этапа(обучение/дообучение).\n
17 С нужной начальной позиции и до нужной конечной позиции.
18
19 :param start: С какой позиции
20 :param stop: По какую позицию
21 :param additional: Дообучение (Да/Нет)
22 :param test: Валидация (Да/Нет)
23 :return: loader: Возвращает даталоадер
24 """
25
26     loader_options = self.dataset_options.get('train_loader')
27     if additional is True:
28         loader_options =
29 self.dataset_options.get('additional_loader')
30     if test is True:
31         loader_options =
32 self.dataset_options.get('test_loader')
33
34     if start is None:
35         start = loader_options.get('start')
36     if stop is None:
37         stop = loader_options.get('stop')
38
39     pdl = PreDataLoader(self.data,
40 candle_count=self.dataset_options.get('candle_count'),
41                         start=start, stop=stop,
42 normalization_pred=self.dataset_options.get('normalization'),
43                     vers=self.dataset_options.get('vers'),
44                     lbl=get_label(self.dataset_options.get('label'), False if
45 additional or test else True))
46     ds = DataSet(pdl.batches)
```

36	
37	<pre> loader = torch.utils.data.DataLoader(ds, shuffle=loader_options.get('shuffle'), batch_size=loader_options.get('batch_size'), num_workers=loader_options.get('num_workers'), drop_last=loader_options.get('drop_last')) </pre>
38	<pre> return loader </pre>

В ходе выполнения нашей программы, будет создаваться много ДС, поэтому в данной реализации при инициализации класса (она происходит единоразово при вызове конструктора класса) в параметр *self.data* класса будут записаны все имеющиеся сущности, а уже позже, при вызове функции *get_dataloader()*, из них будут выбраны сущности с нужной позиции и в нужном количестве.

4.3 Реализация метода дообучения

Так как теперь в работе участвует две разных версии обучения – простая (конечная) и дообучения, удобно будет сразу же вынести эти методы в функции.

Функции большого размера, их можно посмотреть в файле по ссылке [15]. Функция простого (конечного) метода обучения названа в файле, как *feedforward()*. Функция дообучения же названа, как *additional_learning()*.

Теперь вся необходимая функциональность у нас готова и можно переходить к экспериментальной части.

4.4 Проведение исследования

Возложим на исследование несколько целей:

- 1) Провести большое количество исследований.
- 2) Доказать эффективность использования метода дообучения.
- 3) Выявить какие-то закономерности.
- 4) Выявить оптимальные настройки.

Так как, при проведении предыдущего исследования на CNN с простым (конечным) методом обучения, функции-ошибки MAE и MSE показали схожие результаты по эффективности, то в данном исследовании будут приведены

примеры только с одной из метрик. Будем использовать MAE – она немного лучше подходит для данного вида задач.

4.5 Исследования

Мы уже выбрали нужную нам реализацию метода дообучения и теперь нам остается только решить с какими настройками количества эпох проводить тесты. Так как в предыдущих работах для простого (конечного) метода обучения CNN были выбраны 10000, 6400, 3200, 1600, 800, 400, 200, 100, 50, то для простоты сравнения будем использовать эти же количества для тестов при использовании простого метода на CNN. Для дообучения сверточной НС же будем использовать другие величины эпох, а именно: 1000, 512, 256, 128, 64, 32, 16, так как дообучение это, возможно, бесконечное количество этапов простого обучения.

Для тестирования метода дообучения на ViT будем использовать: для этапа простого (конечного) обучения – настройки, хорошо показавшие себя на исследовании в третьей части данной работы, а для этапа дообучения будем использовать 25, 50, 75, 100 эпох.

Все полученные исследования представлены в файле на сайте [13]

На таблице 5 представлена часть результатов исследования по CNN с простым (конечным) обучением.

Таблица 5 Часть эксперимента с простым (конечным) обучением CNN

MAE	feedforward	CNN	Time	Relation	Average_I	Average_I	Standard_Error, %	Max_error	
		10000	303.76	266.883	0.5352	0.5137	0.1952	36.47	3012.5
		6400	193.88	285.79	0.5352	0.5673	0.1774	33.1403	3593.116
		3200	100.60	275.59	0.5352	0.4757	0.2286	42.7062	3933.08

На таблице 6 представлена часть результатов исследования по CNN с дообучением, проведенным после простого обучения с таблицы 5.

Таблица 6 Часть эксперимента с дообучением после обучения с таблицы 5

MAE	adittional	CNN	Time	Relation	Average_I	Average_J	Standard_Error, %	Max_error	
		1000	1351.0	206.03	0.5352	0.5003	0.1458	27.236	3192.7
		512	721.89	233.22	0.5352	0.5379	0.1344	25.122	3020.67
		256	368.28	253.41	0.5352	0.5371	0.151	28.122	3717.36
		128	185.44	233.56	0.5352	0.5373	0.154	28.768	3584.97
		64	97.554	253.27	0.5352	0.5434	0.1441	26.919	3575.38
		32	49.222	246.968	0.5352	0.5265	0.1503	28.090	3455.447
		16	28.547	251.83	0.5352	0.5386	0.1512	28.260	3209.02
1000	1375.9	219.95	0.5352	0.4766	0.1603	29.96	3581.503		
512	740.98	218.544	0.5352	0.5295	0.1256	23.47	3464.65		
256	383.83	216.15	0.5352	0.5278	0.1387	25.92	3506.72		
128	190.14	218.57	0.5352	0.5298	0.1411	26.362	3528.90		
64	98.267	229.67	0.5352	0.5266	0.1445	26.998	3598.98		
32	53.72	232.57	0.5352	0.5258	0.1520	28.398	3453.747		
16	29.25	253.47	0.5352	0.5431	0.1493	27.893	3962.455		
1000	1397.9	240.374	0.5352	0.4753	0.1998	37.33	3993.47		
512	741.91	258.23	0.5352	0.5372	0.1802	33.663	4116.83		
256	374.90	243.70	0.5352	0.5145	0.1894	35.385	4267.04		
128	197.00	241.28	0.5352	0.5189	0.1967	36.74	3953.2		
64	95.99	241.89	0.5352	0.5055	0.2005	37.45	3951.2		
32	52.64	242.19	0.5352	0.4969	0.2019	37.725	4194.01		
16	28.091	249.14	0.5352	0.5006	0.207	38.683	4231.87		

На таблице 7 представлена часть результатов исследования по ViT с простым (конечным) обучением.

Таблица 7 Часть эксперимента с простым (конечным) обучением ViT

MAE	test	Relation	Average_label	Average_prediction	Standard_deviation	Error, %	Max_error, %
		307.81	0.5352	0.4646	0.239	44.70	3380.22
		300.01	0.5352	0.4560	0.2206	41.22	5600.21
		447.90	0.5352	0.5856	0.233	43.65	6480.8
		252.72	0.5352	0.4868	0.215	40.233	2230.64

На таблице 8 представлена часть результатов исследования по ViT с дообучением, проведенным после простого обучения с таблицы 7.

Таблица 8 Часть эксперимента с дообучением после обучения с таблицы 7

Дообучение									
MAE	test	Epochs	Time	Relation	Average_label	Average_prediction	Standard_deviation	Error, %	Max_error, %
1)		25	199.18	269.78	0.5352	0.4487	0.243	45.45	3484.40
		50	387.46	289.66	0.5352	0.4469	0.240	44.96	3540.87
		75	572.56	317.09	0.5352	0.4599	0.244	45.685	4743.68
		100	785.48	257.78	0.5352	0.4679	0.2184	40.808	3455.42
		150	1180.3	324.73	0.5352	0.4545	0.233	43.620	6624.47
MAE	test	Epochs	Time	Relation	Average_label	Average_prediction	Standard_deviation	Error, %	Max_error, %
2)		25	208.34	229.32	0.5352	0.4706	0.193	36.055	3105.502
		50	392.09	231.17	0.5352	0.4872	0.181	33.935	2524.46
		75	556.08	294.01	0.5352	0.4938	0.193	36.15	4675.25
		100	789.49	195.34	0.5352	0.5108	0.1580	29.534	2383.88
MAE	test	Epochs	Time	Relation	Average_label	Average_prediction	Standard_deviation	Error, %	Max_error, %
3)		25	195.74	352.38	0.5352	0.5266	0.2336	43.64	5363.40
		50	395.08	433.99	0.5352	0.5022	0.236	44.19	7800.79
		75	604.31	350.91	0.5352	0.5088	0.226	42.35	6469.398
		100	791.19	384.48	0.5352	0.5151	0.230	43.11	6014.38
MAE	test	Epochs	Time	Relation	Average_label	Average_prediction	Standard_deviation	Error, %	Max_error, %
4)		25	200.92	209.21	0.5352	0.4532	0.224	41.95	2374.90
		50	400.91	135.02	0.5352	0.4525	0.1818	33.98	892.17
		75	564.71	199.29	0.5352	0.4268	0.2327	43.48	2262.69
		100	829.45	194.97	0.5352	0.447	0.212	39.68	2617.6

4.6 Эффективность метода дообучения

Как можно увидеть по метрикам или графикам, результаты, полученные после дообучения выглядят более точными, нежели результаты, полученные после простого обучения. Это говорит о том, что использование метода дообучения приводит к увеличению аналитических способностей НС и повышению точности.

На рисунке 30 представлен график для сравнения до дообучения и после.

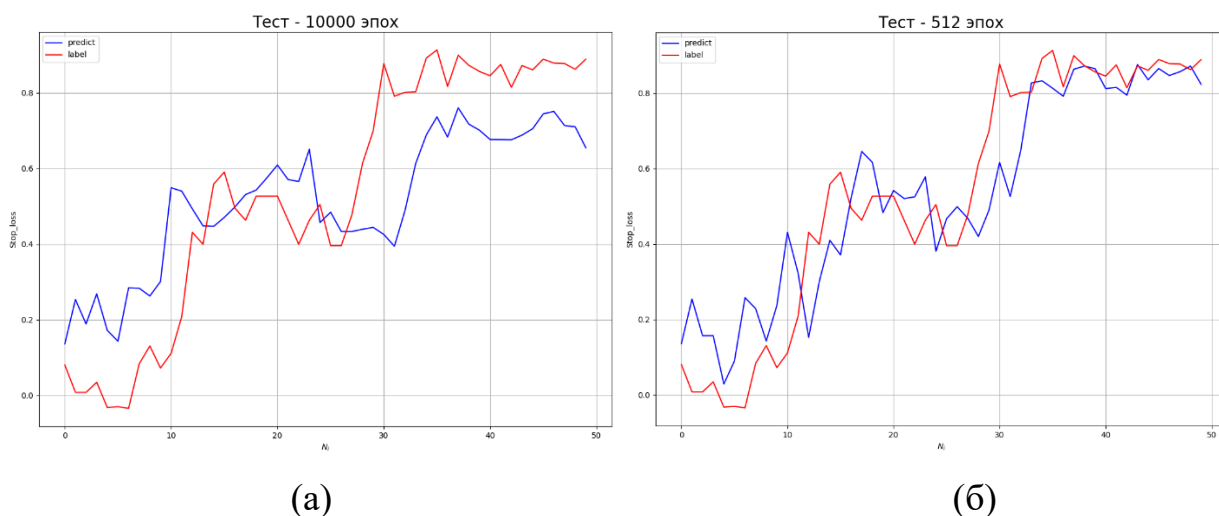


Рисунок 30 – а) график до дообучения, б) после дообучения.

(Сверху подписано количество эпох, использованных на этапе простого обучения и дообучения, соответственно)

На рисунке 31 представлен график для сравнения до дообучения и после.

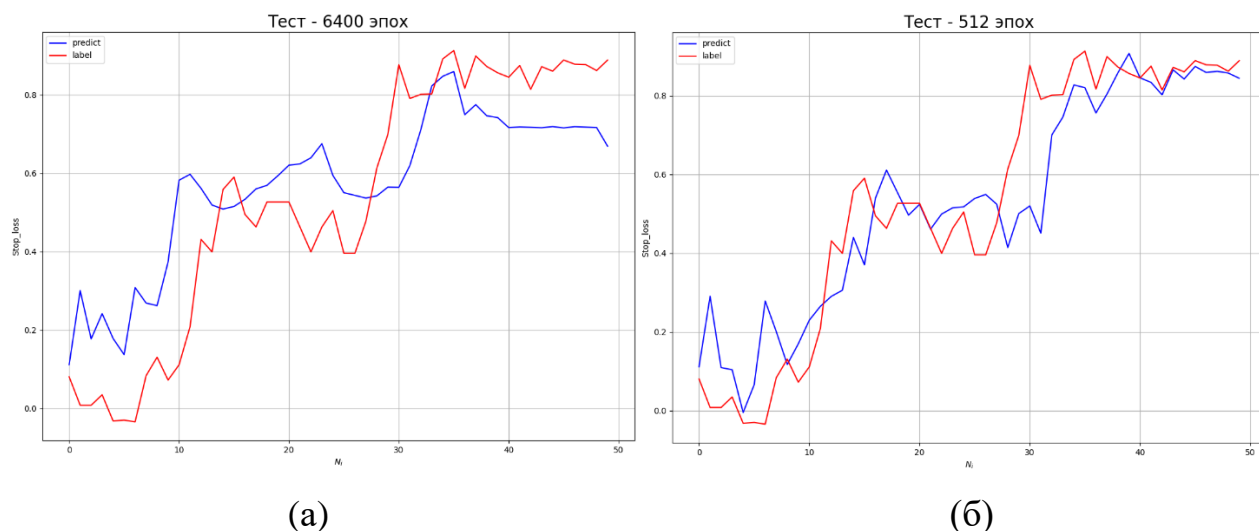


Рисунок 31 – а) график до дообучения, б) после дообучения.

По графикам можно также заметить, что после дообучения, график предсказаний (выделен синим цветом) расположен более близко к графику истинных значений. Следовательно, является более точным.

4.7 Закономерности

При проведении исследования, удалось выяснить, что при малом количестве эпох на этапе обучения в левой части графика появляются большие зашумления, связанные, вероятно, с недостаточным обучением НС. Начиная с 1600 эпох для этапа простого обучения зашумления начинают проходить и появляется более плавный график.

На этапе дообучения достаточно хорошие результаты показывают метрики и графики для 512 эпох. Можно выбрать это число эпох, как оптимальное.

В целом, этап дообучения дает хорошую аппроксимацию истинного графика. Даже при условии, что на этапе простого обучения используют малое количество эпох, этап дообучения сглаживает график предсказаний, тем самым приближая его к истинному графику.

Также удалось выяснить, что этап дообучения позволяет повысить точность примерно на 25%, но при этом придется пожертвовать временем. Это время в масштабах целого дня не сильно большое и им можно пренебречь для такого рода задач.

4.8 Оптимальные настройки

Сложно подобрать оптимальные настройки под задачи похожего типа, но для рассматриваемого случая оптимальными настройками можно назвать:

1. Для этапа простого обучения CNN – 6400 эпох.
2. Для этапа дообучения CNN– 512 эпох.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были изучены различные архитектуры НС, такие как: MLP, CNN, ViT. Была изучена теоретическая часть по корректной реализации собственных датасетов и даталoadеров, используя модули PyTorch. Также была изучена и большая часть инструментария данного модуля. Написаны краткие обзоры полученных теоретических знаний.

Реализованы полномасштабные НС на изученных архитектурах нейронных сетей. Написаны модули для удобного выбора и использования различных функций оптимизации и функций потерь.

Проведено множество исследований на реализованных НС. Сделаны соответствующие выводы по их эффективности. Все результаты этих исследований представлены по ссылке [13].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Pandas Documentation [Электронный ресурс] – URL: <https://pandas.pydata.org/docs/> (дата обращения: 09.02.2024 - 22.03.2024)
2. PyTorch Tutorials [Электронный ресурс] – URL: <https://pytorch.org/tutorials/> (дата обращения: 09.02.2024 - 10.05.2024)
3. PyTorch Documentation [Электронный ресурс] – URL: <https://pytorch.org/docs/stable/index.html> (дата обращения: 09.02.2024 - 10.05.2024)
4. Сайт со сравнением различных функций оптимизации [Электронный ресурс] – URL: <https://emiliendupont.github.io/2018/01/24/optimization-visualization/> (дата обращения: 15.02.2024)
5. «Сверточная нейронная сеть с нуля» [Электронный ресурс] – URL: <https://programforyou.ru/poleznoe/convolutional-network-from-scratch-part-zero-introduction> (дата обращения: 19.02.2024 - 24.03.2024)
6. Блог о сверточной нейронной сети [Электронный ресурс] – URL: <https://habr.com/ru/companies/skillfactory/articles/565232/> (дата обращения: 20.03.2024)
7. CNN from scratch [электронный ресурс] – URL: <https://www.pycodemates.com/2023/07/build-a-cnn-from-scratch-using-python.html> (дата обращения: 23.03.2024)
8. Курс Deep Learning (семестр 1, весна 2023): продвинутый поток [Электронный ресурс] – URL: <https://stepik.org/course/135003/syllabus> (дата обращения: 25.03.2024 - 10.05.2024)
9. Сайт с примером работы сверток [Электронный ресурс] – URL: https://github.com/vdumoulin/conv_arithmetic/tree/master/gif (дата обращения: 11.04.2024)
10. Vision transformer from scratch [электронный ресурс] – URL: <https://debuggercafe.com/vision-transformer-from-scratch/> (дата обращения: 20.04.2024)

11. Хабр статья про трансформер [электронный ресурс] – URL: <https://habr.com/ru/companies/mws/articles/770202/> (дата обращения: 24.04.2024 – 26.04.2024.)
12. Курс Deep Learning (семестр 2, весна 2024) [Электронный ресурс] – URL: <https://stepik.org/course/196142/syllabus> (дата обращения: 25.04.2024 - 25.05.2024)
13. GitHub с результатами всех исследований [Электронный ресурс] – URL: <https://github.com/Relax-FM/Diploma/blob/main/Diploma/excel/DIPLOMA.xlsx> (дата обращения: 09.02.2024 - 10.05.2024)
14. Курс машинного обучения. К.В.Воронцов [Электронный ресурс] – URL: [http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_\(курс_лекций,_К.В.Воронцов\)](http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_(курс_лекций,_К.В.Воронцов)) (дата обращения: 27.04.2024 - 29.04.2024)
15. GitHub с версиями различных функций обучения [Электронный ресурс] – URL: https://github.com/RelaxFM/CNN_VKR/blob/master/Func/learning_version.py (дата обращения: 21.05.2024)

ПРИЛОЖЕНИЕ А. Перечень графических материалов

В перечень графических материалов выпускной квалификационной работы входят:

Лист 1. Архитектура MLP нейронной сети

Лист 2. Графики результатов MLP

Лист 3. Архитектура сверточной нейронной сети

Лист 4. Графики результатов CNN

Лист 5. Графики результатов CNN с методом дообучения

Лист 6. Графики результатов CNN с методом дообучения

Лист 7. Архитектура трансформерной нейронной сети

Лист 8. Графики результатов ViT

Лист 9. Графики результатов ViT с методом дообучения