Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Кафедра алгебры, геометрии и дискретной математики

Направление: Прикладная математика и информатика

**Отчет по технологической практике**

Тема:

“Методы машинного обучения для обработки медицинских сигналов”

**Выполнил**:

студент группы 381903\_3

Манухов В.В.

Подпись

**Научный руководитель**:

директор ин-та ИТММ, зав. кафедрой АГДМ ИИТММ, д.ф.-м.н.

Золотых Н.Ю.

Подпись

Нижний Новгород

2021

Содержание

[Введениe 3](#_Toc106996814)

[Постановка задачи 4](#_Toc106996815)

[Набор данных 5](#_Toc106996816)

[Практическая часть 6](#_Toc106996817)

[Результаты 9](#_Toc106996818)

[Заключение 10](#_Toc106996819)

[Литература 11](#_Toc106996820)

[Электронные источники 12](#_Toc106996821)

[Приложения 13](#_Toc106996822)

# Введениe

Сердечно-сосудистые заболевания являются ведущей причиной высокой смертности во всем мире. Электрокардиография (ЭКГ) – это главный инструмент неинвазивной диагностики сердечно-сосудистых заболеваний. Алгоритмы автоматической интерпретации ЭКГ в качестве систем поддержки диагностики состояний пациентов оказывают значительную помощь врачам. Разработка таких алгоритмов требует больших наборов обучающих данных и четких процедур тестирования. Для решения поставленных задач был предложен клинический набор данных ЭКГ – PTB-XL.

# Постановка задачи

Задачи производственной практики “Научно-исследовательская работа”:

* Изучение области исследуемого объекта (Сигналы ЭКГ, набор данных PTB-XL);
* Первичный анализ и визуализация данных:
  + Вывод таблиц числовых данных;
  + Вывод таблиц категориальных признаков;
  + Построение сигналов ЭКГ;
  + Поиск на графиках взаимосвязей между признаками сигналов ЭКГ пациентов;
* Подготовка данных к работе с алгоритмами машинного обучения;
  + Детектирование выбросов и устранение;
  + Приведение данных к требуемым размерностям и типу данных;
* Создание нескольких моделей машинного обучения на основе сверточных нейронных сетей, которые будет соответствовать свойствам исследуемых объектов;
* Обучение на тренировочной и валидационной выборке, применение на тестовой выборке, анализ ошибки и точности при работе моделей.
* Сравнение работы созданных моделей с работой нескольких известных архитектур сверточных нейронных сетей;

# Набор данных

Данные ЭКГ были аннотированы двумя кардиологами в виде набора данных с несколькими метками, где диагностические метки были дополнительно объединены в надклассы и подклассы. Набор данных охватывает широкий спектр диагностических классов, включая, в частности, большую часть о здоровых людях. Сочетание с метаданными по демографии и дополнительными диагностическими показателями, вероятностями диагноза, вручную аннотированными, свойствами сигнала, а также предлагаемым разделением на тренировочный и тестовый наборы данных дает богатый ресурс для разработки и оценки автоматических алгоритмов интерпретации ЭКГ.

Набор данных PTB-XL – это большой набор данных из 21837 клинических ЭКГ в 12 отведениях от 18885 пациентов длительностью 10 секунд. Этот набор дополняется обширными метаданными по демографическим характеристикам, характеристикам инфаркта, вероятности диагностических заявлений ЭКГ, а также аннотированными свойствами сигналов. PTB-XL - это набор сигналов, собранных в течение почти 7 лет с октября 1989 года по июнь 1996 года. Только в 2019 году был открыт публичный доступ к данным, которые были специально оптимизированы под работу с ними с помощью инструментов машинного обучения.

# Практическая часть

Исследовательская часть проводилась с использованием языка программирования Python и программного интерфейса приложения (API) jupyter notebook.

Для ускоренного анализа работы методов машинного обучения мы использовали облачный сервис Google Colaboratory, который предоставляет доступ к графическому процессору (GPU), значительно ускоряющий процесс обучения глубоких нейронных сетей.

Мы применили следующие пакеты Python:

1. Pandas – это инструмент анализа и обработки данных с открытым исходным кодом, построенный на основе языка программирования Python.
2. NumPy – это проект с открытым исходным кодом, направленный на обеспечение численных вычислений с помощью Python.
3. wfdb – пакет на языке Python для чтения, записи, обработки и построения графиков физиологических сигналов и данных аннотаций. Основные функции ввода-вывода основаны на спецификациях базы данных сигналов (WFDB).
4. ast – помогает приложениям Python обрабатывать деревья грамматики абстрактного синтаксиса Python.
5. scikit-learn (sklearn) – это бесплатная библиотека машинного обучения для языка программирования Python.
6. Tensorflow – это бесплатная библиотека программного обеспечения с открытым исходным кодом для машинного обучения и искусственного интеллекта, где особое внимание уделяется обучению и выводу глубоких нейронных сетей.
7. Keras – это API глубокого обучения, написанный на Python и работающий поверх платформы машинного обучения TensorFlow.
8. Matplotlib – это комплексная библиотека для создания статических, анимированных и интерактивных визуализаций на Python.
9. Seaborn – это библиотека для создания статистической графики на Python. Он построен поверх matplotlib и тесно интегрируется со структурами данных pandas.

Загрузка данных происходит с помощью кода, предоставленного в источнике набора данных PTB-XL. Благодаря этому, мы сразу имеем тренировочный и тестовый выборки, разделенные для достижения наилучшего результата.

Переменные X\_train и X\_test – это 3-х мерные тензоры, которые содержат образцы, представленные 1000 массивами по 12 элементов в каждом. Каждому образцу соответствует метка, содержащаяся в переменной y\_train или y\_test соответственно.

Всего, по условию задачи, работа ведется с 5 классами: NORM, STTC, CD, MI, HYP. Пусть метка, которая содержит несколько классов называется составной. Тогда, так как одному образцу может соответствовать несколько классов одновременно (по одному результату ЭКГ ставится в соответствие несколько заключений), то было решено выполнить разделение таких составных меток на несколько, с соблюдением принадлежности каждой исходному образцу. Для этого была написана функция splitData(), которая разделяет метки с двумя и более классами в составе.

Также, среди меток присутствуют пустые значения (непомеченные образцы). Такие данные было принято удалить из доступных выборок для того, чтобы убрать выбросы и улучшить результат работы модели. С этим справляется написанная функция emptyLabel().

Таким образом, все данные, с которыми будет работать модель машинного обучения, будут фильтроваться с удалением пустых значений и разделением меток на несколько, если они являются составными.

Для улучшения качества работы моделей была применена нормализация данных X\_train и X\_test. Все значения были приведены к числовому диапазону от -1 до 1. Таким образом исключается получения чисел, выходящих за диапазон значений типа данных float64.

Также, были обработаны категориальные атрибуты переменных y\_train и y\_test. При решении задачи классификации значения этих переменных можно представить в виде разряженных меток (для каждого образца имеется только индекс целевого класса) или в виде массивов с указанием целевой вероятности на класс (количество элементов массива соответствует числу классов в задаче классификации). В нашем исследовании использован вариант с разряженными метками.

Следующим шагом стало написание нескольких архитектур нейронных сетей, которые смогут классифицировать различные показатели ЭКГ и с высокой вероятностью ставить в соответствие входному образцу один из 5 классов. Были реализованы простая полносвязная, сверточная одномерная и сверточная двумерная нейронные сети, а также загружены с помощью пакета keras и добавлены к исследованию такие сети как ResNet50, ResNet50V2, ResNet101, VGG16, VGG19. В случае полносвязной сети входные данные имели форму (1000, 12), одномерной и двумерной сверточных сетей – (1000, 12, 1), ResNet50, ResNet50V2, ResNet101, VGG16, VGG19 – (125, 32, 3). Во всех случаях использовалась оптимизация nadam, кроме полносвязной сети, где использовался стандартный стохастический градиентный спуск.

Для оценки качества моделей и сравнения различных алгоритмов применялась метрика accuracy – то есть вычисление того, как часто предсказания модели совпадали с метками.

Проверочными данными (validation data) была выбрана пятая часть от тренировочной выборки.

Для улучшения качества работы модели было реализовано раннее прекращение после 10 эпох ухудшения результата, а также сохранение весов модели при улучшении относительно ошибки на проверочных данных (validation loss), т.е. для оценки на тестовых данных используется модель с весами, показавшая наилучший результат в процессе обучения.

Для каждой модели были построены графики, демонстрирующие изменение точности и ошибки на тренировочных и проверочных данных в ходе обучения, а также проведена оценка работы на тестовых данных.

# Результаты

Сравнивались оценки работы разных моделей на тестовых данных. Все результаты вынесены в таблицу:

|  |  |
| --- | --- |
| **Модель** | **[Ошибка, Точность]** |
| Полносвязная нейронная сеть | [1.5121192932128, 0.32524099946022] |
| Одномерная сверточная нейронная сеть | [0.987737774848, 0.6033559441566] |
| Двумерная сверточная нейронная сеть | [1.0278685092926, 0.6012138724327] |
| ResNet50 | [1.1360713243484, 0.5483756065368] |
| ResNet50V2 | [1.0847526788712, 0.5501606464386] |
| ResNet101 | [1.4233031272888, 0.42234915494919] |
| VGG16 | [1.535159707069, 0.34416279196739] |
| VGG19 | [1.5358237028122, 0.34416279196739] |

Левое значение означает, какую наибольшую ошибку допустила модель при прогнозировании меток на тестовой выборке, а правое точность – процент верно классифицированных образцов из тестовой выборки.

Наилучший по двум значениям результат показала одномерная сверточная нейронная сеть, наибольшую ошибку допустили сети VGG16 и VGG19, а наихудшую точность показала полносвязная нейронная сеть, с почти такой же большой ошибкой. Близкой по обоим показателям к наилучшему результату стала двумерная сверточная сеть, а следом идут ResNet50 и ResNet50V2. Модель на основе ResNet101 показывает заметно ухудшенный результат, по сравнению с ResNet50 и ResNet50V2.

Таким образом можно сделать следующий вывод на основании полученных результатов: с данными в виде сигналов ЭКГ, которые представляют собой показатели (некоторые числовые значения) по 12 отведениями, получаемые в течение 10 секунд, наилучшим образом работает одномерная сверточная нейронная сеть.

# Заключение

Исследование набора данных PTB-XL и создание моделей машинного обучения на его основе является актуальной и сложной задачей, поиск эффективного решения которой приближает человека к созданию автоматизированной поддержки работы врачей по принятию решений об оказании помощи и лечении пациента.

В ходе работы были изучены техники создания моделей сверточных нейронных сетей с применением актуальных версий библиотек машинного обучения, а также применены различные подходы по обработке исходных данных перед их использованием. Работа созданных моделей сверточных нейронных сетей была оценена на тестовых данных в целях получения информации об ошибке на тестовой выборке и точности классификации образцов. Их работа сравнивалась между собой и с известными архитектурами сверточных нейронных сетей. Анализ и сравнение полученных результатов позволяет принимать новые оптимальные решения по настройке гиперпараметров и построению архитектуры модели, а также находить новые пути в представлении и обработке данных перед применением на них моделей машинного обучения.

# Литература

1. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron ISBN: 978-1-492-03264-9;
2. ISBN: 978-1-491-91721-3 Introduction to Machine Learning with Python by Andreas C. Mueller and Sarah Guido;
3. Ш78 Глубокое обучение на Python. — СПб.: Питер, 2018. — 400 с.: ил. — (Серия «Биб- лиотека программиста»). ISBN 978-5-4461-0770-4;
4. Липчак Д. А. Обзор методов автоматической диагностики сердечной аритмии для принятия решений о необходимости проведения дефибрилляции / Д. А. Липчак, А. А. Чупов // Ural Radio Engineering Journal. — 2021. — Vol. 5, No. 4. — P. 380–409;
5. ´Smigiel, S.; Pałczy ´ nski, K.; Ledzi ´ nski, D. ECG Signal Classification Using Deep Learning Techniques Based on the PTB-XL Dataset. Entropy 2021, 23, 1121. <https://doi.org/10.3390/e23091121>;
6. Mehmood, A.; Maqsood, M.; Bashir, M.; Shuyuan, Y. A Deep Siamese Convolution Neural Network for Multi-Class Classification of Alzheimer Disease. Brain Sci. **2020**, 10, 84. <https://doi.org/10.3390/brainsci10020084>
7. Hsieh, C.-H.; Li, Y.-S.; Hwang, B.-J.; Hsiao, C.-H. Detection of Atrial Fibrillation Using 1D Convolutional Neural Network. Sensors **2020**, 20, 2136. <https://doi.org/10.3390/s20072136>

# Электронные источники

1. <https://www.nature.com/articles/s41597-020-0495-6>;
2. <https://physionet.org/content/ptb-xl/1.0.1/>;
3. <https://www.tensorflow.org/>;
4. <https://keras.io/>;

# Приложения

**Подключение основных библиотек и загрузка данных**

## Для Google Colaboratory

In [1]:

*# Библиотека wfdb отсутствует в Google Collab*

**!**pip install wfdb

In [2]:

*# Подключение Google Drive к виртуальной машине.*

**from** google.colab **import** drive drive**.**mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call driv e.mount("/content/drive", force\_remount=True).

*# Копирование данных с Google Drive на локальный диск виртуальной машины.*

In [3]:

**!**cp -r /content/drive/MyDrive/MachineLearningCollab/X\_testCopy.npy .

**!**cp -r /content/drive/MyDrive/MachineLearningCollab/X\_trainCopy.npy .

**!**cp -r /content/drive/MyDrive/MachineLearningCollab/y\_testCopy.npy .

**!**cp -r /content/drive/MyDrive/MachineLearningCollab/y\_trainCopy.npy .

**Подключение библиотек**

In [1]:

*# Подключение библиотек.*

**import** pandas **as** pd **import** numpy **as** np **import** wfdb

**import** ast

**import** pandoc

*#import sklearn*

**from** sklearn.preprocessing **import** OrdinalEncoder, OneHotEncoder

**import** matplotlib.pyplot **as** plt *#import matplotlib.cm as cm* **import** seaborn **as** sns

**import** tensorflow **as** tf

**from** tensorflow **import** keras

**assert** tf**.** version **>=** "2.0"

print("tf. version :", tf**.** version )

print("keras. version ", keras**.** version )

**from** keras **import** layers

**from** keras **import** models

**Первая загрузка данных**

In [2]:

np**.**random**.**seed(42)

**def** load\_raw\_data(df, sampling\_rate, path):

**if** sampling\_rate **==** 100:

data **=** [wfdb**.**rdsamp(path**+**f) **for** f **in** df**.**filename\_lr] *# low rate*

**else**:

data **=** [wfdb**.**rdsamp(path**+**f) **for** f **in** df**.**filename\_hr] *# high rate*

data **=** np**.**array([signal **for** signal, meta **in** data])

**return** data

path **=** 'part/to/ptbxl/' sampling\_rate **=** 100

*# load and convert annotation data*

Y **=** pd**.**read\_csv(path**+**'ptbxl\_database.csv', index\_col **=** 'ecg\_id') Y**.**scp\_codes **=** Y**.**scp\_codes**.**apply(**lambda** x: ast**.**literal\_eval(x))

*# Load raw signal data*

X **=** load\_raw\_data(Y, sampling\_rate, path)

*# load scp\_statements.csv for diagnostic aggregation*

agg\_df **=** pd**.**read\_csv(path**+**'scp\_statements.csv', index\_col **=** 0) agg\_df **=** agg\_df[agg\_df**.**diagnostic **==** 1]

**def** aggregate\_diagnostic(y\_dic): tmp **=** []

**for** key **in** y\_dic**.**keys():

**if** key **in** agg\_df**.**index:

tmp**.**append(agg\_df**.**loc[key]**.**diagnostic\_class)

**return** list(set(tmp))

*# apply diagnistic superclass*

Y['diagnostic\_superclass'] **=** Y**.**scp\_codes**.**apply(aggregate\_diagnostic)

*# Split data into train and test*

test\_fold **=** 10

*# Train*

X\_train **=** X[np**.**where(Y**.**strat\_fold **!=** test\_fold)]

y\_train **=** Y[(Y**.**strat\_fold **!=** test\_fold)]**.**diagnostic\_superclass

*# Test*

X\_test **=** X[np**.**where(Y**.**strat\_fold **==** test\_fold)]

y\_test **=** Y[Y**.**strat\_fold **==** test\_fold]**.**diagnostic\_superclass

**del** X, Y

**Некоторые функции**

In [ ]:

*# Функция, которая разделит списки с 2 и более элементами.*

**def** splitData(X, y):

indexDel **=** [] *# хранит индексы список, которые хранят более 1 элемента (удаляютс* yTmp **=** np**.**zeros((0, 1)) *# хранит элементы (списки с 1 элементом), которые добавл* XTmp **=** np**.**zeros((0, 1000, 12)) *# хранит элементы (списки с 1 элементом), которые* **for** i **in** range(y**.**size):

**if** len(y[i]) **>** 1:

indexDel**.**append(i)

**for** j **in** range(len(y[i])):

yTmp **=** np**.**concatenate((yTmp, np**.**array([y[i][j:j**+**1]]))) XTmp **=** np**.**concatenate((XTmp, np**.**array([X[i]])))

y **=** np**.**delete(y, indexDel, axis **=** 0) X **=** np**.**delete(X, indexDel, axis **=** 0) y **=** y**.**reshape(y**.**size, 1)

**for** i **in** range(y**.**size): y[i] **=** (y[i]**.**tolist()[0])

y **=** np**.**concatenate((y, yTmp)) X **=** np**.**concatenate((X, XTmp)) **return** X, y

*# Поиск пустых значений в массиве меток.*

**def** emptyLabel(y):

*# Найдем ecg\_id строк, где не определен класс.*

ecg\_idEmpty **=** [] *# массив индексов пустых значений по ecg\_id [1, ...]*

ecg\_idEmptyIndex **=** [] *# массив индекс пустых значений по [0, ...]*

sum\_empty **=** 0 *# подсчет пустых значений*

**for** i **in** range(y**.**size):

**if** len(y**.**values[i]) **==** 0:

ecg\_idEmpty**.**append(y**.**index[i]) ecg\_idEmptyIndex**.**append(i)

sum\_empty **+=** 1

*# print(sum\_empty)*

*# print(ecg\_idEmpty)*

*# print(ecg\_idEmptyIndex)*

**return** sum\_empty, ecg\_idEmpty, ecg\_idEmptyIndex

**Анализ данных**

**Первичный анализ**

In [ ]:

*# Вывести конкретное (все, если указана бесконечность в качестве параметра thresho # np.set\_printoptions(threshold=10) # np.inf*

In [ ]:

*# Тип данных переменных y\_train и y\_test, содержащие метки (диагноз).*

print(type(y\_train)) print(type(y\_test)) *# Размерность.*

print(y\_train**.**ndim) print(y\_test**.**ndim)

*# Форма переменных и их размер (количество меток).*

print(y\_train**.**shape, y\_train**.**size) print(y\_test**.**shape, y\_test**.**size)

*# Общее количество элементов (меток).*

print("size:", y\_train**.**size **+** y\_test**.**size)

<class 'pandas.core.series.Series'>

<class 'pandas.core.series.Series'> 1

1

(19634,) 19634

(2203,) 2203

size: 21837

In [ ]:

*# Тип данных переменных X\_train и X\_test, содержащие все объекты (экземпляры, обра*

print(type(X\_test)) print(type(X\_train)) *# Размерность.*

print(X\_train**.**ndim) print(X\_test**.**ndim)

*# Форма переменных и их размер (количество образцов).*

print(X\_train**.**shape, X\_train**.**size) print(X\_test**.**shape, X\_test**.**size)

*# Общее количество элементов (образцов).*

print("size:", X\_train**.**size **+** X\_test**.**size)

<class 'numpy.ndarray'>

<class 'numpy.ndarray'> 3

3

(19634, 1000, 12) 235608000

(2203, 1000, 12) 26436000

size: 262044000

In [ ]:

*# Посмотрим на исходный вид меток.*

print(y\_train)

ecg\_id

1. [NORM]
2. [NORM]
3. [NORM]
4. [NORM]
5. [NORM]

...

21833 [STTC]

21834 [NORM]

21835 [STTC]

21836 [NORM]

21837 [NORM]

Name: diagnostic\_superclass, Length: 19634, dtype: object

In [ ]:

*# Посмотрим на исходный вид показателей.*

print(X\_train)

[[[-1.190e-01 -5.500e-02 6.400e-02 ... -2.600e-02 -3.900e-02 -7.900e-02]

[-1.160e-01 -5.100e-02 6.500e-02 ... -3.100e-02 -3.400e-02 -7.400e-02]

[-1.200e-01 -4.400e-02 7.600e-02 ... -2.800e-02 -2.900e-02 -6.900e-02]

...

[ 6.900e-02 0.000e+00 -6.900e-02 ... 2.400e-02 -4.100e-02 -5.800e-02]

[ 8.600e-02 4.000e-03 -8.100e-02 ... 2.420e-01 -4.600e-02 -9.800e-02]

[ 2.200e-02 -3.100e-02 -5.400e-02 ... 1.430e-01 -3.500e-02 -1.200e-01]]

[[ 4.000e-03 1.380e-01 1.340e-01 ... 1.920e-01 8.300e-02 8.800e-02] [-2.000e-02 1.160e-01 1.360e-01 ... 1.560e-01 5.700e-02 6.300e-02] [-5.300e-02 9.200e-02 1.450e-01 ... 1.070e-01 1.300e-02 2.200e-02]

...

[ 1.210e-01 3.980e-01 2.770e-01 ... -1.065e+00 -4.920e-01 -1.560e-01]

[-3.260e-01 5.700e-02 3.830e-01 ... -2.800e-01 -1.750e-01 -7.100e-02]

[-3.480e-01 -5.600e-02 2.920e-01 ... -3.080e-01 -2.310e-01 -1.450e-01]]

[[-2.900e-02 -7.900e-02 -4.900e-02 ... -1.030e-01 -7.600e-02 -6.600e-02]

[-3.500e-02 -7.000e-02 -3.500e-02 ... -1.040e-01 -7.900e-02 -6.800e-02]

[-5.400e-02 -5.700e-02 -3.000e-03 ... -7.800e-02 -6.600e-02 -5.400e-02]

...

[-2.900e-02 -2.260e-01 -1.980e-01 ... 1.000e-03 2.290e-01 1.800e-02]

[-4.800e-02 -2.660e-01 -2.180e-01 ... -1.000e-03 2.100e-02 -8.000e-03]

[-4.900e-02 -2.880e-01 -2.390e-01 ... 1.000e-03 -1.800e-02 6.000e-03]]

...

[[ 3.800e-02 2.400e-02 -1.400e-02 ... 6.000e-03 1.600e-02 2.700e-02] [ 7.800e-02 6.600e-02 -1.200e-02 ... 1.600e-02 3.100e-02 4.500e-02] [-1.400e-02 -6.000e-03 8.000e-03 ... 9.000e-03 3.000e-02 4.700e-02]

...

[-1.060e-01 -6.200e-02 4.400e-02 ... -2.120e-01 -7.400e-02 5.100e-02]

[-4.500e-02 2.600e-02 7.100e-02 ... -1.510e-01 -5.900e-02 4.800e-02]

[ 4.630e-01 5.300e-01 6.700e-02 ... -1.810e-01 -1.050e-01 3.300e-02]]

[[-5.700e-02 -5.700e-02 0.000e+00 ... -3.500e-02 -3.900e-02 -3.500e-02]

[-4.100e-02 -2.900e-02 1.200e-02 ... -2.300e-02 -2.800e-02 -2.700e-02]

[ 3.000e-03 4.500e-02 4.200e-02 ... -9.000e-03 -1.400e-02 -1.400e-02]

...

[ 3.300e-02 7.000e-02 3.700e-02 ... 2.180e-01 1.010e-01 5.200e-02] [ 2.700e-02 8.200e-02 5.500e-02 ... 2.100e-01 3.350e-01 1.000e-02] [-6.000e-03 5.100e-02 5.700e-02 ... 2.110e-01 3.740e-01 -9.000e-03]]

[[-4.900e-02 -2.500e-02 2.400e-02 ... -4.000e-02 -2.600e-02 -3.100e-02]

[-4.900e-02 -2.900e-02 2.000e-02 ... -3.200e-02 -3.300e-02 -4.300e-02]

[-5.900e-02 -4.200e-02 1.700e-02 ... -4.000e-02 -4.700e-02 -5.100e-02]

...

[ 9.100e-02 1.200e-02 -7.900e-02 ... 6.300e-02 1.600e-02 -1.170e-01]

[ 1.750e-01 2.200e-02 -1.530e-01 ... 8.000e-02 1.800e-02 -1.080e-01]

[ 1.660e-01 -7.000e-03 -1.730e-01 ... 1.060e-01 4.700e-02 -1.030e-01]]]

In [ ]:

*# Содержимое по индексам и значения (тип данных Series, DataFrame пакета Pandas)*

print(y\_train**.**index, '\n') print(y\_train**.**values, '\n') print(y\_train**.**index[0])

print(y\_train**.**values[0])

Int64Index([ 1, 2, 3, 4, 5, 6, 7, 8, 10,

11,

...

21828, 21829, 21830, 21831, 21832, 21833, 21834, 21835, 21836,

21837],

dtype='int64', name='ecg\_id', length=19634)

[list(['NORM']) list(['NORM']) list(['NORM']) ... list(['STTC']) list(['NORM']) list(['NORM'])]

1

['NORM']

In [2]:

*# Посмотрим на содержание y\_train (все данные).*

**for** i **in** range(y\_train**.**size): *# можно вычесть некоторое значение, не большее y\_tra*

print(y\_train**.**values[i])

*# Или так.*

*# for Class in y\_train: # #print(Class)*

**Пропущенные данные**

**Когда мы выделим уникальные значения в y\_train и проведем подсчет каждого, то заметим, что существует пустые значения, которые повторяются 367 раз в тренировочном наборе.**

In [ ]:

*# Создать Ndarray Numpy копию Series Pandas.*

y\_trainNp **=** y\_train**.**to\_numpy() print(type(y\_trainNp))

print(y\_trainNp**.**size)

<class 'numpy.ndarray'> 19634

In [ ]:

*# Выделим уникальные значения y\_train\_np и посчитаем их количество.*

unique, counts **=** np**.**unique(y\_trainNp, return\_counts **= True**) print(unique, '\n', "unique.size: ", unique**.**size)

print(counts)

[list([]) list(['CD']) list(['CD', 'HYP']) list(['CD', 'MI'])

list(['CD', 'MI', 'HYP']) list(['CD', 'MI', 'STTC'])

list(['CD', 'MI', 'STTC', 'HYP']) list(['CD', 'STTC'])

list(['CD', 'STTC', 'HYP']) list(['HYP'])

list(['HYP', 'CD', 'MI', 'STTC']) list(['HYP', 'CD', 'STTC'])

list(['HYP', 'MI', 'STTC']) list(['HYP', 'STTC']) list(['MI'])

list(['MI', 'HYP']) list(['MI', 'STTC']) list(['MI', 'STTC', 'HYP'])

list(['NORM']) list(['NORM', 'CD']) list(['NORM', 'CD', 'MI', 'HYP'])

list(['NORM', 'CD', 'STTC']) list(['NORM', 'HYP']) list(['NORM', 'STTC'])

list(['STTC']) list(['STTC', 'CD', 'MI', 'HYP']) list(['STTC', 'HYP']) list(['STTC', 'MI', 'HYP'])]

unique.size: 28

[ 367 1525 273 1167 112 202 1 434 135 480 14 51 19 149

2282 166 541 1 8170 362 1 5 2 24 2163 127 561 300]

In [ ]:

*# Визуализация с применением seaborn.*

*# Создаем копию.*

y\_trainTmp **=** y\_train

*# Конвертируем list в str.*

**for** i **in** range(y\_trainTmp**.**size):

y\_trainTmp**.**iloc[i] **=** ''**.**join(y\_trainTmp**.**iloc[i]) y\_train\_rI **=** y\_trainTmp**.**reset\_index()

y\_train\_rI\_cat **=** y\_train\_rI[['diagnostic\_superclass']]

colours **=** ['#000099', '#ffff00']

fig, ax **=** plt**.**subplots(figsize **=** (12, 8))

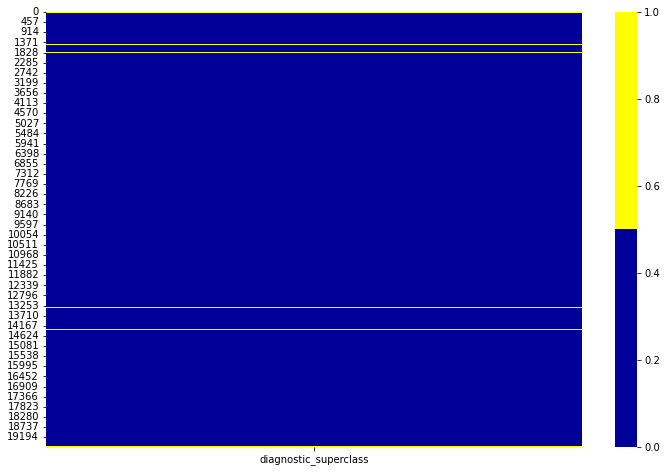
sns**.**heatmap(y\_train\_rI\_cat[:]**.**isin(['']), cmap **=** sns**.**color\_palette(colours))

Out[ ]:

In [ ]:

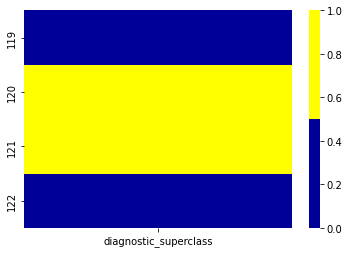
Out[ ]:

<AxesSubplot:>



sns**.**heatmap(y\_train\_rI\_cat[119:123]**.**isin(['']), cmap **=** sns**.**color\_palette(colours))

<AxesSubplot:>



In [ ]:

*# процентное отношение пропущенных данных*

**for** i **in** y\_train\_rI\_cat[0:1000]:

empty **=** np**.**mean(y\_train\_rI\_cat[i]**.**isin([''])) print('{} - {}%'**.**format(i, round(empty**\***100)))

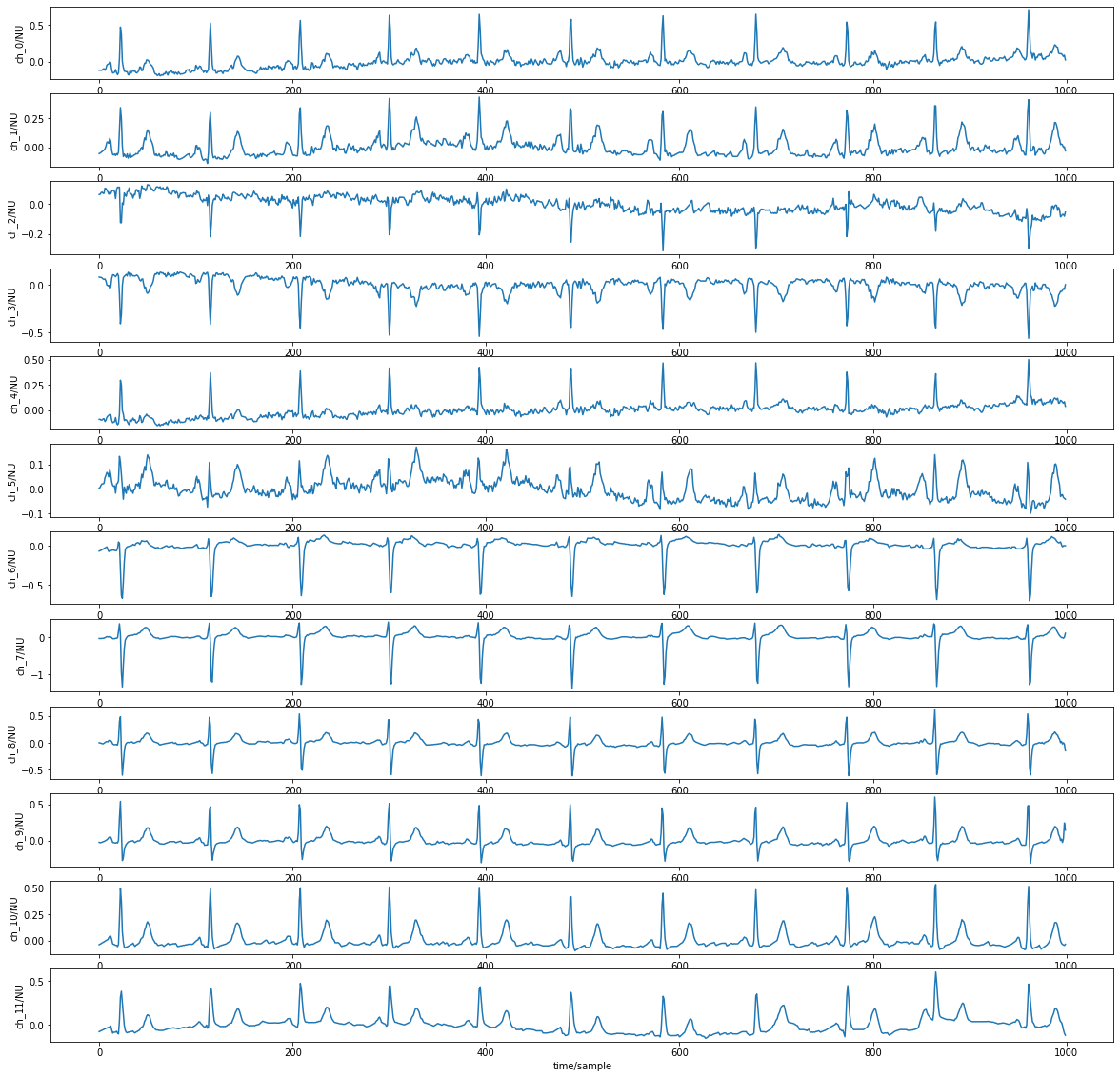
**del** y\_train\_rI\_cat

diagnostic\_superclass - 2%

## Визуализация данных

In [3]:

wfdb**.**plot\_items(signal **=** X\_train[0], figsize **=** (20, 20))



In [4]:

fig, ax **=** plt**.**subplots() *# figure, axes*

fig**.**set\_figwidth(30) fig**.**set\_figheight(20)

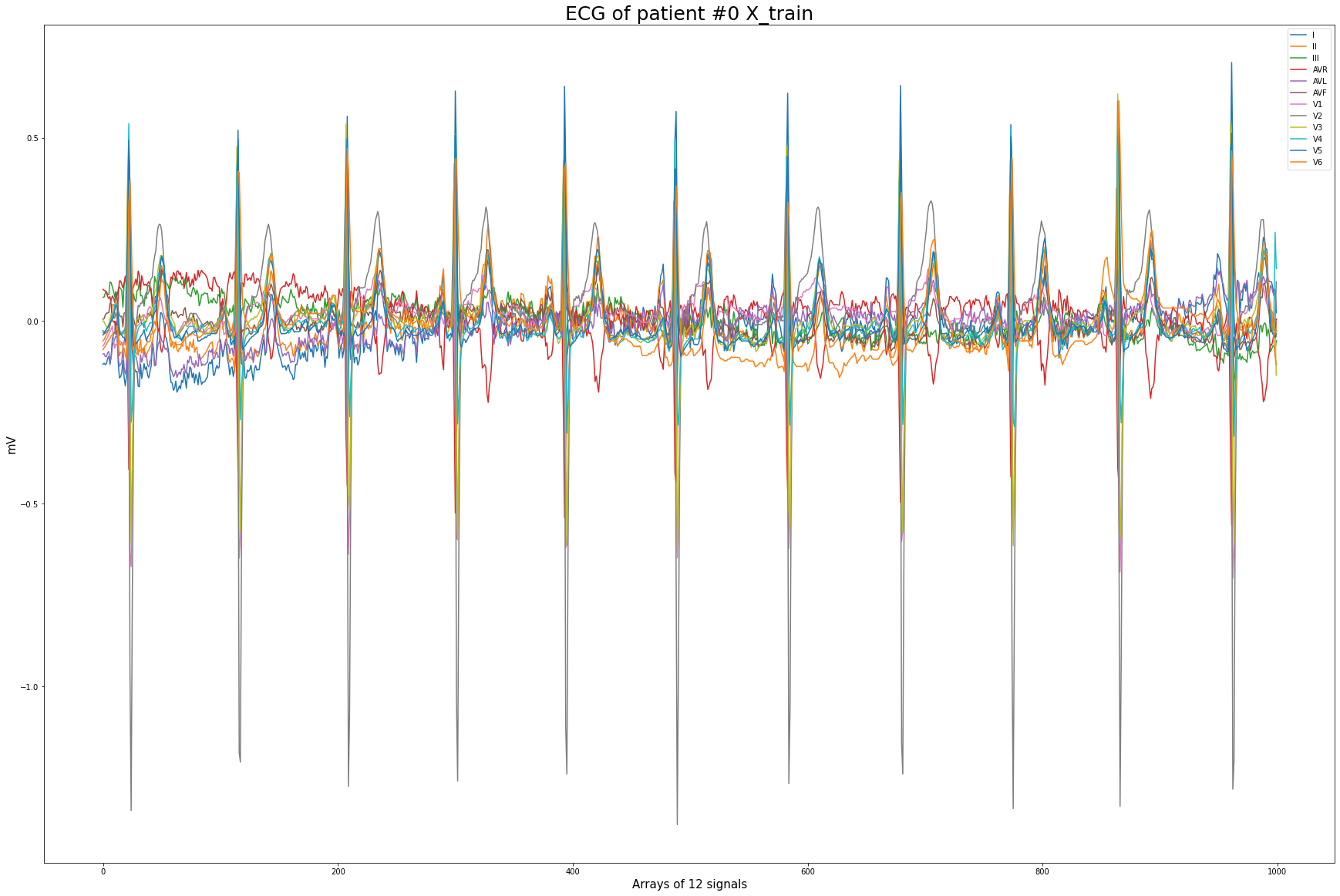
labels **=** ['I', 'II', 'III', 'AVR', 'AVL', 'AVF', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6

ax**.**plot(X\_train[0])

ax**.**set\_xlabel('Arrays of 12 signals', fontsize **=** 15.) ax**.**set\_ylabel('mV', fontsize **=** 15)

ax**.**set\_title('ECG of patient #0 X\_train', fontsize **=** 25.) ax**.**legend(labels) *# label in ax.plot*

**pass**



In [5]:

n\_sig\_min **=** 1

n\_sig\_max **=** 2

n\_arr\_min **=** 0

n\_arr\_max **=** 1000

steps **=** 1000

fig, ax **=** plt**.**subplots() *# figure, axes*

fig**.**set\_figwidth(10) fig**.**set\_figheight(8)

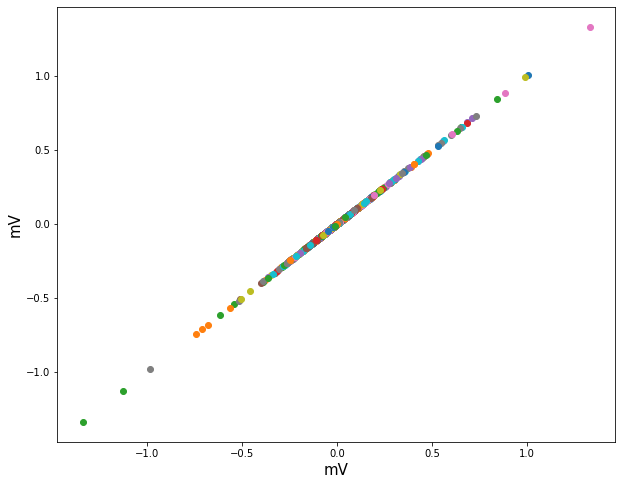
labels **=** np**.**array([i **for** i **in** np**.**arange(0, steps)])

**for** i **in** np**.**arange(n\_arr\_min, n\_arr\_max):

**for** j **in** np**.**arange(n\_sig\_min, n\_sig\_max):

ax**.**scatter(X\_train[i][j][0], X\_train[i][j][0])

ax**.**set\_xlabel('mV', fontsize **=** 15.) ax**.**set\_ylabel('mV', fontsize **=** 15) **pass**



In [6]:

fig, ax **=** plt**.**subplots() *# figure, axes*

fig**.**set\_figwidth(10) fig**.**set\_figheight(8)

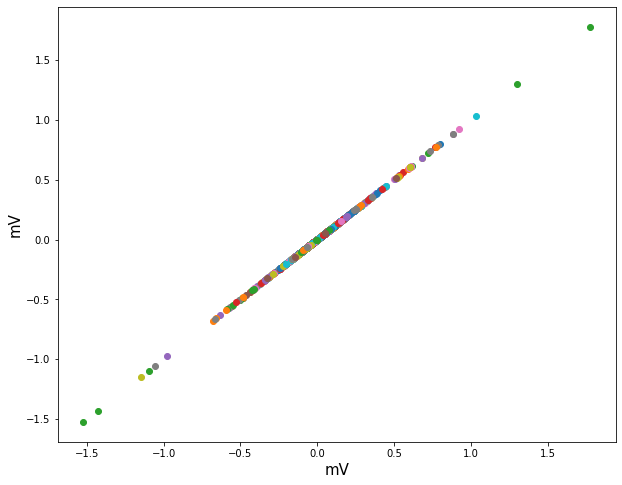
labels **=** np**.**array([i **for** i **in** np**.**arange(0, steps)])

**for** i **in** np**.**arange(n\_arr\_min, n\_arr\_max):

**for** j **in** np**.**arange(n\_sig\_min, n\_sig\_max):

ax**.**scatter(X\_train[i][j][1], X\_train[i][j][1])

ax**.**set\_xlabel('mV', fontsize **=** 15.) ax**.**set\_ylabel('mV', fontsize **=** 15) **pass**



|  |  |  |  |
| --- | --- | --- | --- |
|  | | | Предобработка данных |
| Копирование данных |
| In | [ | ]: | *# Копируем исходный тренировочный и тестовый набор, чтобы не испортить данные.* |
|  |  |  | y\_trainCopy **=** y\_train**.**copy() |
|  |  |  | X\_trainCopy **=** X\_train**.**copy() |
|  |  |  | y\_testCopy **=** y\_test**.**copy() |
|  |  |  | X\_testCopy **=** X\_test**.**copy() |
|  |  |  | Обработка пустых значений |
|  |  |  | Тренировочные данные |
|  |  |  | Обработка |
| In | [ | ]: | *# Выявление с помощью функции emptyLable() всех пустых меток.*  sum\_empty, ecg\_idEmpty, ecg\_idEmptyIndex **=** emptyLabel(y\_trainCopy) |
|  |  |  |  |
| In | [ | ]: | *# Проверка пустых значений.* |
|  |  |  | print("Проверка 1: ", y\_trainCopy**.**get(ecg\_idEmpty[:5])) |
|  |  |  | print("Проверка 2: ", y\_trainCopy**.**values[ecg\_idEmptyIndex[:5]]) |

Проверка 1: ecg\_id

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 17 | | | | [] |
| 18 | | | | [] |
| 20 | | | | [] |
| 23 | | | | [] |
| 34 | | | | [] |
| Name: | | | | diagnostic\_superclass, dtype: object |
|  |  |  | Проверка 2: [list([]) list([]) list([]) list([]) list([])] | |
| In | [ | ]: | *# Удалим соответствующие строки по ecg\_id в X\_trainCopy и в y\_trainCopy.*  y\_trainCopy**.**drop(labels **=** ecg\_idEmpty, axis **=** 0, inplace **= True**) | |
|  |  |  | *#y\_trainCopy.reset\_index(drop = True, inplace = True)* | |
|  |  |  | X\_trainCopy **=** np**.**delete(X\_trainCopy, ecg\_idEmptyIndex, axis **=** 0) | |
|  |  |  |  | |
| In | [ | ]: | *# Проверим размерности наших данных.* | |
|  |  |  | print(y\_trainCopy**.**shape, X\_trainCopy**.**shape) | |
|  |  |  | (19267,) (19267, 1000, 12) | |
| In | [ | ]: | *# Создать Ndarray Numpy копию Series Pandas c исключенными пустыми значениями.* | |
|  |  |  | y\_trainNpCopy **=** y\_trainCopy**.**to\_numpy() | |
|  |  |  | print(y\_trainNpCopy**.**size) | |
|  |  |  | 19267 | |
| In | [ | ]: | *# Разделяем метки с несколькими значениями: например, y[i] = [['NORM'], ['CD'], ['* | |
|  |  |  | X\_trainCopy, y\_trainNpCopy **=** splitData(X\_trainCopy, y\_trainNpCopy) | |
|  |  |  | *# Конвертируем list в str.* | |
|  |  |  | y\_trainCopy **=** [] | |
|  |  |  | **for** i **in** range(y\_trainNpCopy**.**size): | |
|  | | | y\_trainCopy**.**append(''**.**join(y\_trainNpCopy[i]**.**tolist())) y\_trainCopy **=** np**.**array(y\_trainCopy)  *# Удаление старого numpy объекта*  **del** y\_trainNpCopy  *# Сохранить файлы X\_trainCopy и y\_trainCopy, y\_trainNpCopy.*  np**.**save('X\_trainCopy', X\_trainCopy) np**.**save('y\_trainCopy', y\_trainCopy)  *# Размерности и содержимое данных.*  print(y\_trainCopy**.**shape, X\_trainCopy**.**shape, '\n')  *# Все уникальные классы.*  np**.**unique(y\_trainCopy, return\_counts **= True**) | |

Out[ ]:

In [5]:

*# Загрузка данных (не требуется выполнять код выше). y\_trainNpCopy не загружается.*

X\_trainCopy **=** np**.**load('X\_trainCopy.npy') y\_trainCopy **=** np**.**load('y\_trainCopy.npy')

(25025,) (25025, 1000, 12)

(array(['CD', 'HYP', 'MI', 'NORM', 'STTC'], dtype='<U4'), array([4409, 2392, 4933, 8564, 4727], dtype=int64))

Скачать

### Тестовые данные

Обработка

In [ ]:

*# Очистка данных с пустыми метками. # Выявление пустых значений.*

sum\_empty, ecg\_idEmpty, ecg\_idEmptyIndex **=** emptyLabel(y\_testCopy)

*# Удалим соответствующие строки по ecg\_id в X\_trainCopy и y\_trainCopy.*

y\_testCopy**.**drop(labels **=** ecg\_idEmpty, axis **=** 0, inplace **= True**) X\_testCopy **=** np**.**delete(X\_testCopy, ecg\_idEmptyIndex, axis **=** 0)

*# Создать ndarray Numpy копию Series Pandas.*

y\_testNpCopy **=** y\_testCopy**.**to\_numpy() print(y\_testNpCopy**.**size)

*# Разделяем метки с несколькими значениями.*

X\_testCopy, y\_testNpCopy **=** splitData(X\_testCopy, y\_testNpCopy)

*# Конвертируем list в str.*

y\_testCopy **=** []

**for** i **in** range(y\_testNpCopy**.**size):

y\_testCopy**.**append(''**.**join(y\_testNpCopy[i]**.**tolist())) y\_testCopy **=** np**.**array(y\_testCopy)

*# Сохранить файлы X\_testCopy и y\_testCopy.*

np**.**save('X\_testCopy', X\_testCopy) np**.**save('y\_testCopy', y\_testCopy)

*# Проверим размерности и содержимое данных.*

print(y\_testCopy**.**shape, X\_testCopy**.**shape)

2163

(2801,) (2801, 1000, 12)

Скачать

In [6]:

*# Загрузка тестовых файлов.*

X\_testCopy **=** np**.**load('X\_testCopy.npy') y\_testCopy **=** np**.**load('y\_testCopy.npy')

## Нормализация данных

### Тренировочные данные

In [7]:

*# Нормирование от -1 до 1: [-1, 1] значений всех образцов.*

*# Минимальное и максимальное значение в 3D массивe X\_trainCopy. (см. функции np.am*

print(np**.**amax(X\_trainCopy)) print(np**.**amin(X\_trainCopy)) *# По осям.*

*# print(np.amax(X\_trainCopy, axis = 0)) # print(np.amin(X\_trainCopy, axis = 0)) # print(np.amax(X\_trainCopy, axis = 1)) # print(np.amin(X\_trainCopy, axis = 1)) # print(np.amax(X\_trainCopy, axis = 2)) # print(np.amin(X\_trainCopy, axis = 2))*

*# Нахождение максимального по абсолютному значению.*

print(np**.**max(np**.**abs(X\_trainCopy)))

*# print(np.max(np.abs(X\_trainCopy), axis = 0)) # print(np.max(np.abs(X\_trainCopy), axis = 1)) # print(np.max(np.abs(X\_trainCopy), axis = 2))*

X\_trainCopyNorm **=** X\_trainCopy**/**(np**.**amax(np**.**abs(X\_trainCopy))) print(np**.**max(X\_trainCopyNorm))

print(np**.**min(X\_trainCopyNorm))

*# Аналогично.*

*# print(np.amax(X\_trainCopyNorm)) # print(np.amin(X\_trainCopyNorm))*

17.212

-20.032

20.032

0.8592252396166135

-1.0

### Тестовые данные

In [8]:

*# Нормирование от -1 до 1: [-1, 1]*

*# Нахождение максимального по абсолютному значению.*

print(np**.**max(np**.**abs(X\_testCopy)))

*# Нормировка.*

X\_testCopyNorm **=** X\_testCopy**/**(np**.**amax(np**.**abs(X\_trainCopy))) print(np**.**max(X\_testCopyNorm))

print(np**.**min(X\_testCopyNorm))

12.966

0.6472643769968051

-0.569888178913738

## Обработка текстовых и категориальных атрибутов по Geron "homl2RU". 1 вар: стр. 113; 2 вар: стр. 397.

### Тренировочные данные

1. вариант

In [9]:

*# 1 вар.*

*# Разреженные метки (для каждого образца имеется только индекс целевого класса от*

ordinal\_encoder **=** OrdinalEncoder()

y\_trainCopyEncoded **=** ordinal\_encoder**.**fit\_transform(pd**.**DataFrame(y\_trainCopy)) print(y\_trainCopyEncoded[:10])

print(ordinal\_encoder**.**categories\_)

[[3.]

[3.]

[3.]

[3.]

[3.]

[3.]

[3.]

[2.]

[3.]

[3.]]

[array(['CD', 'HYP', 'MI', 'NORM', 'STTC'], dtype=object)]

1. вариант

In [ ]:

*# 2 вар.*

*# Целевая вероятность на класс. (Применяется функция потери "categorical\_crossentr #в метки, представляемые векторами. Чтобы сделать наоборот: np.argmax() с axis = 1* cat\_encoder **=** OneHotEncoder(sparse**=False**)

y\_trainCopyHot **=** cat\_encoder**.**fit\_transform(pd**.**DataFrame(y\_trainCopy)) print(y\_trainCopyHot[:10])

print(cat\_encoder**.**categories\_)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [[0. | 0. | 0. | 1. | 0.] |
| [0. | 0. | 0. | 1. | 0.] |
| [0. | 0. | 0. | 1. | 0.] |
| [0. | 0. | 0. | 1. | 0.] |
| [0. | 0. | 0. | 1. | 0.] |
| [0. | 0. | 0. | 1. | 0.] |
| [0. | 0. | 0. | 1. | 0.] |
| [0. | 0. | 1. | 0. | 0.] |
| [0. | 0. | 0. | 1. | 0.] |
| [0. | 0. | 0. | 1. | 0.]] |

[array(['CD', 'HYP', 'MI', 'NORM', 'STTC'], dtype=object)]

### Тестовые данные

1 вариант

In [10]:

*# Разреженные метки (для каждого образца имеется только индекс целевого класса от*

ordinal\_encoder **=** OrdinalEncoder()

y\_testCopyEncoded **=** ordinal\_encoder**.**fit\_transform(pd**.**DataFrame(y\_testCopy)) y\_testCopyEncoded[:10]

Out[10]:

In [ ]:

*# Целевая вероятность на класс.*

cat\_encoder **=** OneHotEncoder(sparse**=False**)

y\_testCopyHot **=** cat\_encoder**.**fit\_transform(pd**.**DataFrame(y\_testCopy)) y\_testCopyHot[:10]

array([[3.],

[3.],

[3.],

[3.],

[3.],

[2.],

[0.],

[3.],

[3.],

[3.]])

2 вариант

Out[ ]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| array([[0., | 0., | 0., | 1., | 0.], |
| [0., | 0., | 0., | 1., | 0.], |
| [0., | 0., | 0., | 1., | 0.], |
| [0., | 0., | 0., | 1., | 0.], |
| [0., | 0., | 0., | 1., | 0.], |
| [0., | 0., | 1., | 0., | 0.], |
| [1., | 0., | 0., | 0., | 0.], |
| [0., | 0., | 0., | 1., | 0.], |
| [0., | 0., | 0., | 1., | 0.], |
| [0., | 0., | 0., | 1., | 0.]]) |

**Создание, обучение и тестирование разных моделей нейронных сетей, в том числе известные архитектуры**

**Обучение модели будет проходить с использованием тренировочных (train) и проверочных (valid) данных.**

**Оценка работы модели производится с применением тестовых данных (test).**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | Полносвязная нейронная сеть |
| Преобразование данных для работы с сетью |
| In | [ | ]: | X\_train **=** X\_trainCopyNorm  y\_train **=** y\_trainCopyEncoded X\_test **=** X\_testCopyNorm  y\_test **=** y\_testCopyEncoded |
|  |  |  | print(X\_train**.**shape, y\_train**.**shape, X\_test**.**shape, y\_test**.**shape) |
|  |  |  | (25025, 1000, 12) (25025, 1) (2801, 1000, 12) (2801, 1) |
|  |  |  | Создание модели |
| In | [ | ]: | *# Создание обычной полносвязной модели.* |
|  |  |  | model **=** keras**.**models**.**Sequential()  model**.**add(keras**.**layers**.**Flatten(input\_shape **=** [1000,12])) *# входной слой (преобразо* |
|  | | | model**.**add(keras**.**layers**.**Dense(10000, activation**=**"relu")) *# скрытый слой*  model**.**add(keras**.**layers**.**Dense(3000, activation**=**"relu")) model**.**add(keras**.**layers**.**Dense(800, activation**=**"relu")) model**.**add(keras**.**layers**.**Dense(100, activation**=**"relu"))  model**.**add(keras**.**layers**.**Dense(5, activation**=**"softmax")) *# выходной слой*  model**.**summary() |
|  |  |  | Model: "sequential"  Layer (type) Output Shape Param #  ================================================================= |

|  |  |  |  |
| --- | --- | --- | --- |
| flatten (Flatten) | (None, | 12000) | 0 |
| dense (Dense) | (None, | 10000) | 120010000 |
| dense\_1 (Dense) | (None, | 3000) | 30003000 |
| dense\_2 (Dense) | (None, | 800) | 2400800 |
| dense\_3 (Dense) | (None, | 100) | 80100 |
| dense\_4 (Dense) | (None, | 5) | 505 |

=================================================================

Total params: 152,494,405

Trainable params: 152,494,405

Non-trainable params: 0

### Обучение модели

In [ ]:

*# Компиляция модели.*

model**.**compile(loss **=** "sparse\_categorical\_crossentropy", optimizer **=** "sgd",

metrics **=** ["accuracy"])

*# Реализация раннего прекращения.*

checkpoint\_filepath **=** './checkpoint\_Neural1/'

model\_checkpoint\_cb **=** tf**.**keras**.**callbacks**.**ModelCheckpoint( filepath**=**checkpoint\_filepath,

save\_weights\_only**=True**, save\_best\_only**=True**)

early\_stopping\_cb **=** keras**.**callbacks**.**EarlyStopping(

patience**=**10,

restore\_best\_weights**=True**)

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 30, validation\_split **=** 0.2, callbac

*# Сохранение модели.*

model**.**save('Neural1.h5')

*# Откат к наилучшей модели.*

model**.**load\_weights(checkpoint\_filepath)

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch 1/30  626/626 [==============================] - 19s 26ms/step | - loss: | 1.4712 | - accurac |
| y: 0.4194 - val\_loss: 1.9151 - val\_accuracy: 0.0314 Epoch 2/30  626/626 [==============================] - 15s 24ms/step | - loss: | 1.4406 | - accurac |
| y: 0.4199 - val\_loss: 1.8737 - val\_accuracy: 0.0320 |  |  |  |
| Epoch 3/30  626/626 [==============================] - 16s 25ms/step | - loss: | 1.4249 | - accurac |
| y: 0.4242 - val\_loss: 1.8646 - val\_accuracy: 0.0426 Epoch 4/30 |  |  |  |
| 626/626 [==============================] - 16s 25ms/step  y: 0.4353 - val\_loss: 1.8231 - val\_accuracy: 0.0611 Epoch 5/30  626/626 [==============================] - 15s 25ms/step | * loss: * loss: | 1.4102  1.3988 | * accurac * accurac |
| y: 0.4404 - val\_loss: 1.8200 - val\_accuracy: 0.0723 Epoch 6/30  626/626 [==============================] - 16s 26ms/step | - loss: | 1.3905 | - accurac |
| y: 0.4429 - val\_loss: 1.7832 - val\_accuracy: 0.0883 |  |  |  |
| Epoch 7/30  626/626 [==============================] - 16s 25ms/step | - loss: | 1.3829 | - accurac |
| y: 0.4448 - val\_loss: 1.7764 - val\_accuracy: 0.0969 Epoch 8/30 |  |  |  |
| 626/626 [==============================] - 12s 20ms/step  y: 0.4477 - val\_loss: 1.8772 - val\_accuracy: 0.0783 Epoch 9/30  626/626 [==============================] - 13s 21ms/step | * loss: * loss: | 1.3758  1.3682 | * accurac * accurac |
| y: 0.4489 - val\_loss: 1.8730 - val\_accuracy: 0.0825 Epoch 10/30  626/626 [==============================] - 12s 20ms/step | - loss: | 1.3593 | - accurac |
| y: 0.4521 - val\_loss: 1.7819 - val\_accuracy: 0.0977 |  |  |  |
| Epoch 11/30  626/626 [==============================] - 16s 25ms/step | - loss: | 1.3487 | - accurac |
| y: 0.4565 - val\_loss: 1.7629 - val\_accuracy: 0.1061 Epoch 12/30 |  |  |  |
| 626/626 [==============================] - 13s 20ms/step  y: 0.4588 - val\_loss: 1.7880 - val\_accuracy: 0.1033 Epoch 13/30  626/626 [==============================] - 12s 19ms/step | * loss: * loss: | 1.3361  1.3199 | * accurac * accurac |
| y: 0.4635 - val\_loss: 1.8831 - val\_accuracy: 0.0933 Epoch 14/30  626/626 [==============================] - 12s 19ms/step | - loss: | 1.2979 | - accurac |
| y: 0.4702 - val\_loss: 1.7900 - val\_accuracy: 0.1109 |  |  |  |
| Epoch 15/30  626/626 [==============================] - 13s 20ms/step | - loss: | 1.2740 | - accurac |
| y: 0.4802 - val\_loss: 2.0018 - val\_accuracy: 0.0859 Epoch 16/30 |  |  |  |
| 626/626 [==============================] - 15s 24ms/step  y: 0.4917 - val\_loss: 1.5299 - val\_accuracy: 0.2064 Epoch 17/30  626/626 [==============================] - 13s 20ms/step | * loss: * loss: | 1.2446  1.2230 | * accurac * accurac |
| y: 0.5007 - val\_loss: 2.7188 - val\_accuracy: 0.0643 Epoch 18/30  626/626 [==============================] - 13s 20ms/step | - loss: | 1.1974 | - accurac |
| y: 0.5087 - val\_loss: 1.9089 - val\_accuracy: 0.1271 |  |  |  |
| Epoch 19/30  626/626 [==============================] - 13s 20ms/step | - loss: | 1.1646 | - accurac |
| y: 0.5249 - val\_loss: 1.6291 - val\_accuracy: 0.2036 Epoch 20/30 |  |  |  |
| 626/626 [==============================] - 12s 19ms/step  y: 0.5359 - val\_loss: 1.7214 - val\_accuracy: 0.1856 Epoch 21/30  626/626 [==============================] - 13s 20ms/step | * loss: * loss: | 1.1315  1.1102 | * accurac * accurac |
| y: 0.5426 - val\_loss: 1.5362 - val\_accuracy: 0.2226  Epoch 22/30 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 626/626 [==============================] - 12s 20ms/step  y: 0.5603 - val\_loss: 1.5851 - val\_accuracy: 0.2212 | - loss: | 1.0735 | - accurac |
| Epoch 23/30  626/626 [==============================] - 13s 20ms/step | - loss: | 1.0467 | - accurac |
| y: 0.5689 - val\_loss: 1.5615 - val\_accuracy: 0.2324 Epoch 24/30 |  |  |  |
| 626/626 [==============================] - 12s 20ms/step  y: 0.5782 - val\_loss: 2.2638 - val\_accuracy: 0.1389 Epoch 25/30  626/626 [==============================] - 13s 20ms/step | * loss: * loss: | 1.0188  0.9898 | * accurac * accurac |
| y: 0.5932 - val\_loss: 1.8561 - val\_accuracy: 0.1924 Epoch 26/30  626/626 [==============================] - 13s 21ms/step | - loss: | 0.9651 | - accurac |
| y: 0.6049 - val\_loss: 1.5684 - val\_accuracy: 0.2671 |  |  |  |

<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fea201feed0>

|  |  |  |
| --- | --- | --- |
| Out[ | ]: |  |
|  |  | Оценка работы модели |
| In [ | ]: | score **=** model**.**evaluate(X\_test, y\_test) score |

Out[ ]:

In [ ]:

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy'] loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

epochs **=** range(1, len(acc) **+** 1)

plt**.**plot(epochs, acc, 'bo', label**=**'Training acc')

plt**.**plot(epochs, val\_acc, 'b', label**=**'Validation acc') plt**.**title('Training and validation accuracy')

plt**.**legend()

plt**.**figure()

plt**.**plot(epochs, loss, 'bo', label**=**'Training loss')

plt**.**plot(epochs, val\_loss, 'b', label**=**'Validation loss') plt**.**title('Training and validation loss')

plt**.**legend()

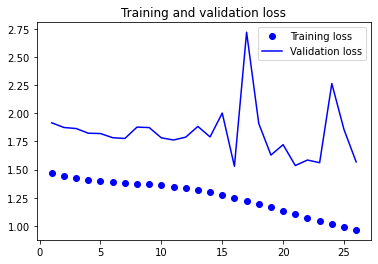
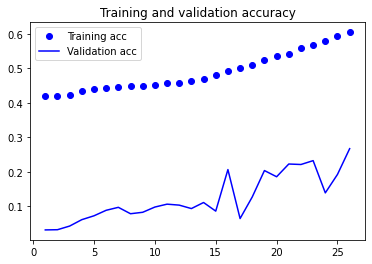
plt**.**show()

88/88 [==============================] - 1s 8ms/step - loss: 1.5121 - accuracy: 0.

3252

[1.5121192932128906, 0.32524099946022034]

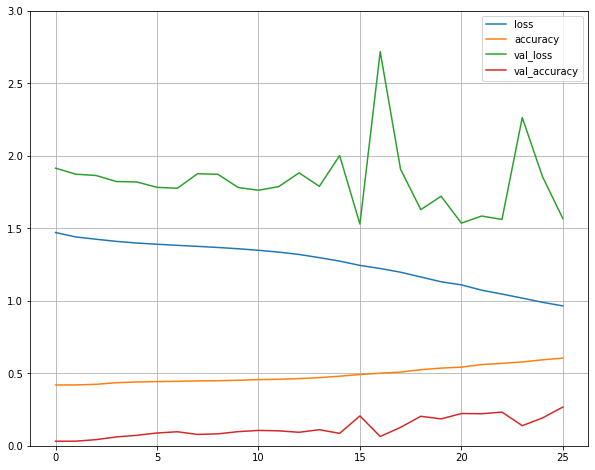
### Графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения



In [ ]:

pd**.**DataFrame(history**.**history)**.**plot(figsize **=** (10, 8)) plt**.**grid()

plt**.**gca()**.**set\_ylim(0, 3) plt**.**show()



Одномерная сверточная нейроная сеть

### Преобразование данных для работы с сетью

In [ ]:

X\_train **=** X\_trainCopyNorm

y\_train **=** y\_trainCopyEncoded X\_test **=** X\_testCopyNorm

y\_test **=** y\_testCopyEncoded

print(X\_train**.**shape, y\_train**.**shape, X\_test**.**shape, y\_test**.**shape)

(25025, 1000, 12) (25025, 1) (2801, 1000, 12) (2801, 1)

### Создание модели

In [ ]:

*# \_3*

model **=** keras**.**Sequential([

keras**.**layers**.**Conv1D(filters **=** 8, kernel\_size **=** 16, padding **=** 'same', strides **=** 2 keras**.**layers**.**MaxPooling1D(pool\_size**=**8, strides **=** 4, padding **=** 'same'),

keras**.**layers**.**Conv1D(filters **=** 12, kernel\_size **=** 12, activation**=**'relu', padding **=**

keras**.**layers**.**MaxPooling1D(pool\_size**=**4, strides **=** 2, padding **=** 'same'),

keras**.**layers**.**Conv1D(filters **=** 32, kernel\_size **=** 9, padding **=** 'same', strides **=** 1 keras**.**layers**.**MaxPooling1D(pool\_size**=**5, strides **=** 2,padding **=** 'same'),

keras**.**layers**.**Conv1D(filters **=** 64, kernel\_size **=** 7, padding **=** 'same', strides **=** 1 keras**.**layers**.**MaxPooling1D(pool\_size**=**4, strides **=** 2, padding **=** 'same'),

keras**.**layers**.**Conv1D(filters **=** 64, kernel\_size **=** 5, padding **=** 'same', strides **=** 1

keras**.**layers**.**MaxPooling1D(pool\_size**=**2, strides **=** 2, padding **=** 'same'),

keras**.**layers**.**Conv1D(filters **=** 64, kernel\_size **=** 3, padding **=** 'same', strides **=** 1 keras**.**layers**.**MaxPooling1D(pool\_size**=**2, strides **=** 2, padding **=** 'same'),

keras**.**layers**.**Conv1D(filters **=** 72, kernel\_size **=** 3, padding **=** 'same', strides **=** 1

keras**.**layers**.**MaxPooling1D(pool\_size**=**2, strides **=** 2, padding **=** 'same'), keras**.**layers**.**Flatten(),

keras**.**layers**.**Dense(216, activation**=**'relu'), keras**.**layers**.**Dropout(0.1),

keras**.**layers**.**Dense(5, activation**=**'softmax')

])

model**.**summary()

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

conv1d (Conv1D) (None, 500, 8) 1544

max\_pooling1d (MaxPooling1D (None, 125, 8) 0

)

conv1d\_1 (Conv1D) (None, 63, 12) 1164

max\_pooling1d\_1 (MaxPooling (None, 32, 12) 0

1D)

conv1d\_2 (Conv1D) (None, 32, 32) 3488

max\_pooling1d\_2 (MaxPooling (None, 16, 32) 0

1D)

conv1d\_3 (Conv1D) (None, 16, 64) 14400

max\_pooling1d\_3 (MaxPooling (None, 8, 64) 0

1D)

conv1d\_4 (Conv1D) (None, 8, 64) 20544

max\_pooling1d\_4 (MaxPooling (None, 4, 64) 0

1D)

conv1d\_5 (Conv1D) (None, 4, 64) 12352

max\_pooling1d\_5 (MaxPooling (None, 2, 64) 0

1D)

|  |  |  |  |
| --- | --- | --- | --- |
| conv1d\_6 (Conv1D) | (None, | 2, 72) | 13896 |
| max\_pooling1d\_6 (MaxPooling 1D) | (None, 1, 72) | | 0 |
| flatten (Flatten) | (None, 72) | | 0 |
| dense (Dense) | (None, 216) | | 15768 |
| dropout (Dropout) | (None, 216) | | 0 |
| dense\_1 (Dense) | (None, 5) | | 1085 |

=================================================================

Total params: 84,241

Trainable params: 84,241

Non-trainable params: 0

### Обучение модели

In [ ]:

*# Компиляция модели*

model**.**compile(optimizer**=**'nadam',

loss **=** 'sparse\_categorical\_crossentropy',

metrics **=** ['accuracy'])

*# Реализация раннего прекращения.*

checkpoint\_filepath **=** './checkpoint\_1D/'

model\_checkpoint\_cb **=** tf**.**keras**.**callbacks**.**ModelCheckpoint( filepath**=**checkpoint\_filepath,

save\_weights\_only**=True**, save\_best\_only**=True**)

early\_stopping\_cb **=** keras**.**callbacks**.**EarlyStopping( patience**=**10,

restore\_best\_weights**=True**)

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, batch\_size**=**32, epochs**=**30, validation\_split**=**0

*# Сохранение модели.*

model**.**save('1D.h5')

*# Откат к наилучшей модели.*

model**.**load\_weights(checkpoint\_filepath)

Epoch 1/30

626/626 [==============================] - 19s 11ms/step - loss: 1.1929 - accurac

y: 0.5162 - val\_loss: 1.5038 - val\_accuracy: 0.2571 Epoch 2/30

626/626 [==============================] - 6s 10ms/step - loss: 1.0655 - accuracy:

0.5673 - val\_loss: 1.4665 - val\_accuracy: 0.2921 Epoch 3/30

626/626 [==============================] - 6s 9ms/step - loss: 1.0096 - accuracy:

0.6101 - val\_loss: 1.6917 - val\_accuracy: 0.3253 Epoch 4/30

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 626/626 [==============================]  0.6326 - val\_loss: 1.4382 - val\_accuracy: Epoch 5/30  626/626 [==============================] | * 6s 10ms/step   0.3648   * 6s 10ms/step | * loss: * loss: | 0.9707  0.9263 | * accuracy: * accuracy: |
| 0.6476 - val\_loss: 1.3989 - val\_accuracy: Epoch 6/30  626/626 [==============================] | 0.3724  - 6s 10ms/step | - loss: | 0.8961 | - accuracy: |
| 0.6550 - val\_loss: 1.3796 - val\_accuracy: | 0.3868 |  |  |  |
| Epoch 7/30 |  |  |  |  |

626/626 [==============================] - 6s 9ms/step - loss: 0.8815 - accuracy:

0.6625 - val\_loss: 1.4598 - val\_accuracy: 0.3782 Epoch 8/30

626/626 [==============================] - 6s 10ms/step - loss: 0.8686 - accuracy:

0.6661 - val\_loss: 1.3629 - val\_accuracy: 0.3894 Epoch 9/30

626/626 [==============================] - 6s 9ms/step - loss: 0.8592 - accuracy:

0.6717 - val\_loss: 1.4430 - val\_accuracy: 0.3766 Epoch 10/30

626/626 [==============================] - 6s 10ms/step - loss: 0.8511 - accuracy:

0.6730 - val\_loss: 1.4274 - val\_accuracy: 0.3876 Epoch 11/30

626/626 [==============================] - 7s 11ms/step - loss: 0.8461 - accuracy:

0.6759 - val\_loss: 1.3894 - val\_accuracy: 0.3872 Epoch 12/30

626/626 [==============================] - 6s 10ms/step - loss: 0.8383 - accuracy:

0.6769 - val\_loss: 1.4198 - val\_accuracy: 0.3808 Epoch 13/30

626/626 [==============================] - 6s 9ms/step - loss: 0.8326 - accuracy:

0.6810 - val\_loss: 1.3771 - val\_accuracy: 0.3890 Epoch 14/30

626/626 [==============================] - 7s 11ms/step - loss: 0.8239 - accuracy:

0.6822 - val\_loss: 1.3857 - val\_accuracy: 0.3874 Epoch 15/30

626/626 [==============================] - 6s 10ms/step - loss: 0.8179 - accuracy:

0.6842 - val\_loss: 1.4132 - val\_accuracy: 0.3932 Epoch 16/30

626/626 [==============================] - 6s 9ms/step - loss: 0.8117 - accuracy:

0.6847 - val\_loss: 1.4073 - val\_accuracy: 0.3896 Epoch 17/30

626/626 [==============================] - 6s 9ms/step - loss: 0.8071 - accuracy:

0.6868 - val\_loss: 1.3997 - val\_accuracy: 0.3892 Epoch 18/30

626/626 [==============================] - 6s 10ms/step - loss: 0.8016 - accuracy:

0.6899 - val\_loss: 1.3596 - val\_accuracy: 0.3966 Epoch 19/30

626/626 [==============================] - 6s 10ms/step - loss: 0.7956 - accuracy:

0.6924 - val\_loss: 1.3722 - val\_accuracy: 0.3960 Epoch 20/30

626/626 [==============================] - 6s 9ms/step - loss: 0.7888 - accuracy:

0.6947 - val\_loss: 1.4650 - val\_accuracy: 0.3828 Epoch 21/30

626/626 [==============================] - 7s 12ms/step - loss: 0.7834 - accuracy:

0.6934 - val\_loss: 1.4539 - val\_accuracy: 0.3838 Epoch 22/30

Out[ ]:

In [ ]:

score **=** model**.**evaluate(X\_test, y\_test) score

626/626 [==============================] - 6s 9ms/step - loss: 0.7803 - accuracy:

0.6953 - val\_loss: 1.4166 - val\_accuracy: 0.3856 Epoch 23/30

626/626 [==============================] - 6s 10ms/step - loss: 0.7748 - accuracy:

0.6953 - val\_loss: 1.4169 - val\_accuracy: 0.3910 Epoch 24/30

626/626 [==============================] - 6s 10ms/step - loss: 0.7706 - accuracy:

0.6981 - val\_loss: 1.4582 - val\_accuracy: 0.3888 Epoch 25/30

626/626 [==============================] - 6s 9ms/step - loss: 0.7624 - accuracy:

0.7000 - val\_loss: 1.4915 - val\_accuracy: 0.3882 Epoch 26/30

626/626 [==============================] - 6s 9ms/step - loss: 0.7540 - accuracy:

0.7034 - val\_loss: 1.4678 - val\_accuracy: 0.3912 Epoch 27/30

626/626 [==============================] - 6s 10ms/step - loss: 0.7522 - accuracy:

0.7020 - val\_loss: 1.4476 - val\_accuracy: 0.3804 Epoch 28/30

626/626 [==============================] - 6s 9ms/step - loss: 0.7454 - accuracy:

0.7055 - val\_loss: 1.4566 - val\_accuracy: 0.3862

<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f5eaa2fde90>

### Оценка работы модели

Out[ ]:

In [ ]:

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy'] loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

epochs **=** range(1, len(acc) **+** 1)

plt**.**plot(epochs, acc, 'bo', label**=**'Training acc')

plt**.**plot(epochs, val\_acc, 'b', label**=**'Validation acc') plt**.**title('Training and validation accuracy')

plt**.**legend()

plt**.**figure()

plt**.**plot(epochs, loss, 'bo', label**=**'Training loss')

plt**.**plot(epochs, val\_loss, 'b', label**=**'Validation loss') plt**.**title('Training and validation loss')

plt**.**legend()

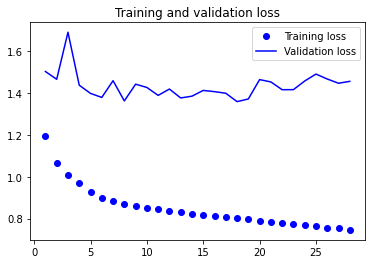
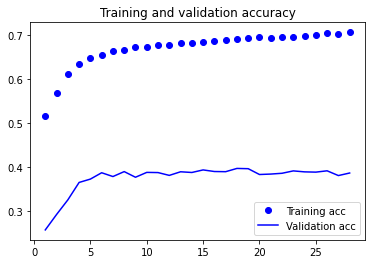
plt**.**show()

88/88 [==============================] - 0s 5ms/step - loss: 0.9877 - accuracy: 0.

6034

[0.987737774848938, 0.6033559441566467]

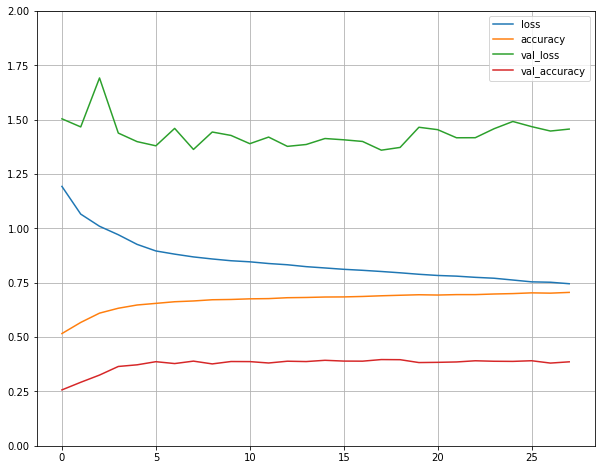
### Графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения



In [ ]:

pd**.**DataFrame(history**.**history)**.**plot(figsize **=** (10, 8)) plt**.**grid()

plt**.**gca()**.**set\_ylim(0, 2) plt**.**show()



## Двумерная сверточная нейронная сеть

### Преобразование данных для работы с моделью

In [ ]:

X\_train **=** X\_trainCopyNorm

y\_train **=** y\_trainCopyEncoded X\_test **=** X\_testCopyNorm

y\_test **=** y\_testCopyEncoded

*# До преобразования.*

X\_train**.**shape, X\_test**.**shape

*# Преобразование формы тензора.*

X\_train **=** X\_train[**...**, np**.**newaxis] X\_test **=** X\_test[**...**, np**.**newaxis]

*# После преобразования.*

X\_train**.**shape, X\_test**.**shape

### Создание модели

In [ ]:

**from** functools **import** partial

DefaultConv2D **=** partial(keras**.**layers**.**Conv2D, kernel\_size **=** 3, activation **=** "relu", model **=** keras**.**models**.**Sequential([

DefaultConv2D(filters **=** 64, kernel\_size **=** 7, input\_shape **=** [1000, 12, 1]),

keras**.**layers**.**MaxPooling2D(pool\_size **=** 2),

DefaultConv2D(filters **=** 128),

DefaultConv2D(filters **=** 128),

keras**.**layers**.**MaxPooling2D(pool\_size **=** 2),

DefaultConv2D(filters **=** 256),

DefaultConv2D(filters **=** 256),

keras**.**layers**.**MaxPooling2D(pool\_size **=** 2),

keras**.**layers**.**Flatten(),

keras**.**layers**.**Dense(units**=**128, activation **=** "relu"), keras**.**layers**.**Dropout(0.5),

keras**.**layers**.**Dense(units**=**64, activation **=** "relu"),

keras**.**layers**.**Dropout(0.5),

keras**.**layers**.**Dense(units **=** 5, activation **=** "softmax"),

])

print(model**.**summary())

*# # Список слоев модели. # print(model.layers)*

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |  |
| --- | --- | --- | --- |
| conv2d (Conv2D) | (None, | 1000, 12, 64) | 3200 |
| max\_pooling2d (MaxPooling2D | (None, 500, 6, 64) | | 0 |
| ) |  | |  |
| conv2d\_1 (Conv2D) | (None, 500, 6, 128) | | 73856 |
| conv2d\_2 (Conv2D) | (None, 500, 6, 128) | | 147584 |
| max\_pooling2d\_1 (MaxPooling 2D) | (None, 250, 3, 128) | | 0 |
| conv2d\_3 (Conv2D) | (None, 250, 3, 256) | | 295168 |
| conv2d\_4 (Conv2D) | (None, 250, 3, 256) | | 590080 |
| max\_pooling2d\_2 (MaxPooling 2D) | (None, 125, 1, 256) | | 0 |
| flatten (Flatten) | (None, 32000) | | 0 |
| dense (Dense) | (None, 128) | | 4096128 |
| dropout (Dropout) | (None, 128) | | 0 |
| dense\_1 (Dense) | (None, 64) | | 8256 |
| dropout\_1 (Dropout) | (None, 64) | | 0 |
| dense\_2 (Dense) | (None, 5) | | 325 |

=================================================================

Total params: 5,214,597

Trainable params: 5,214,597

Non-trainable params: 0

None

### Обучение модели

In [ ]:

*# Компиляция модели.*

model**.**compile(loss **=** 'sparse\_categorical\_crossentropy', optimizer **=** 'nadam',

metrics **=** ['accuracy'])

*# Реализация раннего прекращения.*

checkpoint\_filepath **=** './checkpoint\_2D/'

model\_checkpoint\_cb **=** tf**.**keras**.**callbacks**.**ModelCheckpoint(

filepath**=**checkpoint\_filepath, save\_weights\_only**=True**,

save\_best\_only**=True**)

early\_stopping\_cb **=** keras**.**callbacks**.**EarlyStopping( patience**=**10,

restore\_best\_weights**=True**)

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 10, validation\_split **=** 0.2, callbac

*# Сохранение модели.*

model**.**save('2D.h5')

*# Откат к наилучшей модели.*

model**.**load\_weights(checkpoint\_filepath)

### Оценка работы модели

In [ ]:

score **=** model**.**evaluate(X\_test, y\_test) score

Out[ ]:

In [ ]:

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy'] loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

epochs **=** range(1, len(acc) **+** 1)

plt**.**plot(epochs, acc, 'bo', label**=**'Training acc')

plt**.**plot(epochs, val\_acc, 'b', label**=**'Validation acc') plt**.**title('Training and validation accuracy')

plt**.**legend()

plt**.**figure()

plt**.**plot(epochs, loss, 'bo', label**=**'Training loss')

plt**.**plot(epochs, val\_loss, 'b', label**=**'Validation loss') plt**.**title('Training and validation loss')

plt**.**legend()

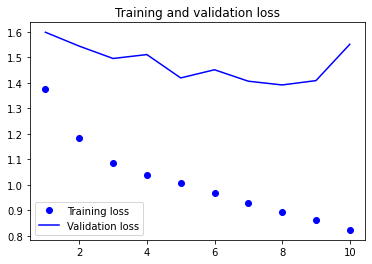
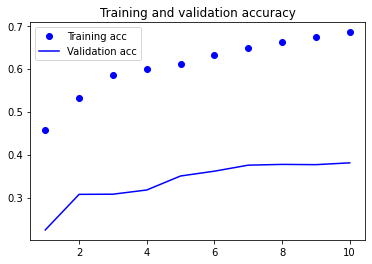
plt**.**show()

88/88 [==============================] - 3s 30ms/step - loss: 1.0279 - accuracy:

0.6012

[1.0278685092926025, 0.6012138724327087]

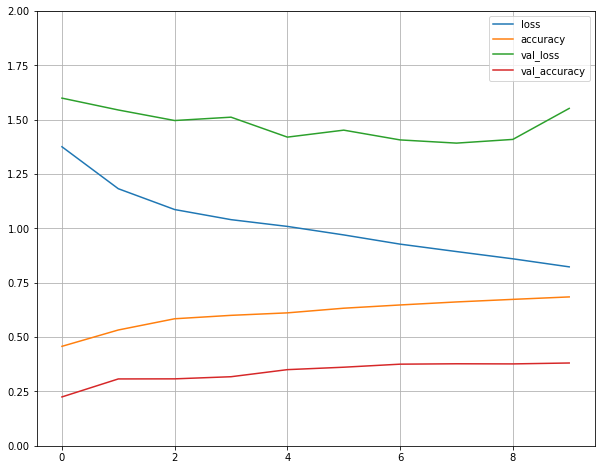
### Графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения



In [ ]:

pd**.**DataFrame(history**.**history)**.**plot(figsize **=** (10, 8)) plt**.**grid()

plt**.**gca()**.**set\_ylim(0, 2) plt**.**show()



## ResNet50

### Преобразование данных для работы с моделью

In [ ]:

X\_train **=** X\_trainCopyNorm

y\_train **=** y\_trainCopyEncoded X\_test **=** X\_testCopyNorm

y\_test **=** y\_testCopyEncoded

*# До преобразования.*

X\_train**.**shape, X\_test**.**shape

*# Преобразование формы тензора.*

X\_train **=** np**.**reshape(X\_train, (25025, **-**1, 32, 3))

X\_test **=** np**.**reshape(X\_test, (2801, **-**1, 32, 3))

*# После преобразования.*

X\_train**.**shape, X\_test**.**shape

Out[ ]:

In [ ]:

conv\_base **=** keras**.**applications**.**ResNet50(weights**=**'imagenet',

include\_top **= False**,

input\_shape **=** (125, 32, 3)

)

model **=** models**.**Sequential() model**.**add(conv\_base)

model**.**add(layers**.**Flatten())

model**.**add(layers**.**Dense(256, activation**=**'relu')) model**.**add(layers**.**Dense(5, activation**=**'softmax')) model**.**summary()

((25025, 125, 32, 3), (2801, 125, 32, 3))

### Загрузка модели

Downloading data from https://storage.googleapis.com/tensorflow/keras-application s/resnet/resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5

94773248/94765736 [==============================] - 2s 0us/step

94781440/94765736 [==============================] - 2s 0us/step

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| resnet50 (Functional) | (None, | 4, 1, | 2048) | 23587712 |
| flatten (Flatten) | (None, | 8192) |  | 0 |
| dense (Dense) | (None, | 256) |  | 2097408 |
| dense\_1 (Dense) | (None, | 5) |  | 1285 |

=================================================================

Total params: 25,686,405

Trainable params: 25,633,285

Non-trainable params: 53,120

### Обучение модели

In [ ]:

*# Замораживание сверточной основы.*

conv\_base**.**trainable **= False**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 5, validation\_split **=** 0.2)

Epoch 1/5

626/626 [==============================] - 30s 28ms/step - loss: 1.5195 - accurac

y: 0.3940 - val\_loss: 1.9083 - val\_accuracy: 0.0314 Epoch 2/5

626/626 [==============================] - 15s 25ms/step - loss: 1.4612 - accurac

y: 0.4199 - val\_loss: 1.9371 - val\_accuracy: 0.0314 Epoch 3/5

626/626 [==============================] - 16s 26ms/step - loss: 1.4591 - accurac

y: 0.4199 - val\_loss: 1.8357 - val\_accuracy: 0.0314 Epoch 4/5

626/626 [==============================] - 16s 25ms/step - loss: 1.4584 - accurac

y: 0.4199 - val\_loss: 1.8656 - val\_accuracy: 0.0314 Epoch 5/5

626/626 [==============================] - 15s 25ms/step - loss: 1.4581 - accurac

y: 0.4199 - val\_loss: 1.9802 - val\_accuracy: 0.0314

In [ ]:

*# Разморозка сверточной основы.*

conv\_base**.**trainable **= True**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Реализация раннего прекращения.*

checkpoint\_filepath **=** './checkpoint\_ResNet50/'

model\_checkpoint\_cb **=** tf**.**keras**.**callbacks**.**ModelCheckpoint( filepath**=**checkpoint\_filepath,

save\_weights\_only**=True**, save\_best\_only**=True**)

early\_stopping\_cb **=** keras**.**callbacks**.**EarlyStopping( patience**=**10,

restore\_best\_weights**=True**)

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 30, validation\_split **=** 0.2, callbac

*# Сохранение модели.*

model**.**save("ResNet50.h5")

*# Откат к наилучшей модели.*

model**.**load\_weights(checkpoint\_filepath)

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch 1/30  626/626 [==============================] - 75s 89ms/step | - loss: | 1.1972 | - accurac |
| y: 0.5618 - val\_loss: 1.8524 - val\_accuracy: 0.2492 Epoch 2/30  626/626 [==============================] - 54s 86ms/step | - loss: | 1.0176 | - accurac |
| y: 0.6128 - val\_loss: 1.9863 - val\_accuracy: 0.2797 |  |  |  |
| Epoch 3/30  626/626 [==============================] - 54s 87ms/step | - loss: | 0.9905 | - accurac |
| y: 0.6180 - val\_loss: 1.5982 - val\_accuracy: 0.1790 Epoch 4/30 |  |  |  |
| 626/626 [==============================] - 57s 92ms/step  y: 0.6235 - val\_loss: 1.4700 - val\_accuracy: 0.3606 Epoch 5/30  626/626 [==============================] - 53s 85ms/step | * loss: * loss: | 0.9939  0.9398 | * accurac * accurac |
| y: 0.6319 - val\_loss: 1.6144 - val\_accuracy: 0.3457 Epoch 6/30  626/626 [==============================] - 53s 85ms/step | - loss: | 0.9167 | - accurac |
| y: 0.6413 - val\_loss: 1.6112 - val\_accuracy: 0.3415 |  |  |  |
| Epoch 7/30  626/626 [==============================] - 54s 86ms/step | - loss: | 0.9106 | - accurac |
| y: 0.6484 - val\_loss: 1.5525 - val\_accuracy: 0.2266 Epoch 8/30 |  |  |  |
| 626/626 [==============================] - 53s 85ms/step  y: 0.6515 - val\_loss: 1.5964 - val\_accuracy: 0.3201 Epoch 9/30  626/626 [==============================] - 55s 88ms/step | * loss: * loss: | 0.8971  0.8837 | * accurac * accurac |
| y: 0.6553 - val\_loss: 1.4036 - val\_accuracy: 0.3750 Epoch 10/30  626/626 [==============================] - 53s 85ms/step | - loss: | 0.8648 | - accurac |
| y: 0.6646 - val\_loss: 1.4766 - val\_accuracy: 0.3774 |  |  |  |
| Epoch 11/30  626/626 [==============================] - 52s 83ms/step | - loss: | 0.8733 | - accurac |
| y: 0.6617 - val\_loss: 1.5979 - val\_accuracy: 0.3722 Epoch 12/30 |  |  |  |
| 626/626 [==============================] - 53s 84ms/step  y: 0.6671 - val\_loss: 1.6375 - val\_accuracy: 0.3489 Epoch 13/30  626/626 [==============================] - 52s 84ms/step | * loss: * loss: | 0.8471  0.8399 | * accurac * accurac |
| y: 0.6674 - val\_loss: 2.1129 - val\_accuracy: 0.3341 Epoch 14/30  626/626 [==============================] - 52s 83ms/step | - loss: | 0.8287 | - accurac |
| y: 0.6667 - val\_loss: 1.7548 - val\_accuracy: 0.3594 |  |  |  |
| Epoch 15/30  626/626 [==============================] - 52s 83ms/step | - loss: | 0.8756 | - accurac |
| y: 0.6545 - val\_loss: 1.4578 - val\_accuracy: 0.3475 Epoch 16/30 |  |  |  |
| 626/626 [==============================] - 52s 83ms/step  y: 0.6605 - val\_loss: 1.5944 - val\_accuracy: 0.3321 Epoch 17/30  626/626 [==============================] - 55s 87ms/step | * loss: * loss: | 0.8612  0.8884 | * accurac * accurac |
| y: 0.6484 - val\_loss: 1.4602 - val\_accuracy: 0.3658 Epoch 18/30  626/626 [==============================] - 55s 87ms/step | - loss: | 0.8723 | - accurac |
| y: 0.6589 - val\_loss: 1.4576 - val\_accuracy: 0.3702 |  |  |  |
| Epoch 19/30  626/626 [==============================] - 52s 84ms/step | - loss: | 0.8662 | - accurac |
| y: 0.6630 - val\_loss: 1.4177 - val\_accuracy: 0.3854 |  |  |  |

Out[ ]:

In [ ]:

score **=** model**.**evaluate(X\_test, y\_test) score

<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f558b1c21d0>

### Оценка работы модели

Out[ ]:

In [ ]:

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy'] loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

epochs **=** range(1, len(acc) **+** 1)

plt**.**plot(epochs, acc, 'bo', label**=**'Training acc')

plt**.**plot(epochs, val\_acc, 'b', label**=**'Validation acc') plt**.**title('Training and validation accuracy')

plt**.**legend()

plt**.**figure()

plt**.**plot(epochs, loss, 'bo', label**=**'Training loss')

plt**.**plot(epochs, val\_loss, 'b', label**=**'Validation loss') plt**.**title('Training and validation loss')

plt**.**legend()

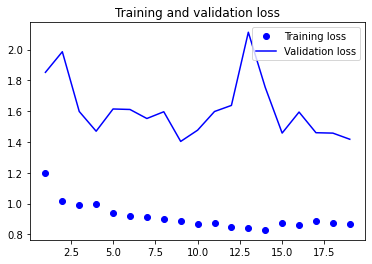
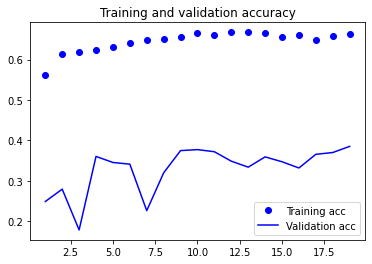
plt**.**show()

88/88 [==============================] - 2s 24ms/step - loss: 1.1361 - accuracy:

0.5484

[1.1360713243484497, 0.5483756065368652]

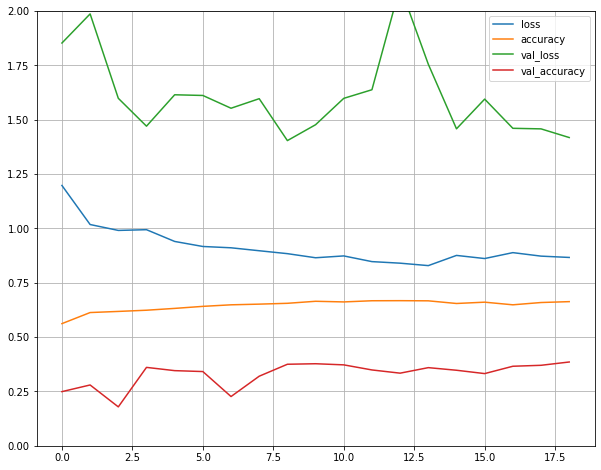
### Графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения



In [ ]:

pd**.**DataFrame(history**.**history)**.**plot(figsize **=** (10, 8)) plt**.**grid()

plt**.**gca()**.**set\_ylim(0, 2) plt**.**show()



## ResNet50V2

### Преобразование данных для работы с моделью

In [ ]:

X\_train **=** X\_trainCopyNorm

y\_train **=** y\_trainCopyEncoded X\_test **=** X\_testCopyNorm

y\_test **=** y\_testCopyEncoded

*# До преобразования.*

X\_train**.**shape, X\_test**.**shape

*# Преобразование формы тензора.*

X\_train **=** np**.**reshape(X\_train, (25025, **-**1, 32, 3))

X\_test **=** np**.**reshape(X\_test, (2801, **-**1, 32, 3))

*# После преобразования.*

X\_train**.**shape, X\_test**.**shape

### Загрузка модели

In [ ]:

conv\_base **=** keras**.**applications**.**ResNet50V2(weights**=**'imagenet',

include\_top **= False**,

input\_shape **=** (125, 32, 3)

)

model **=** models**.**Sequential() model**.**add(conv\_base)

model**.**add(layers**.**Flatten())

model**.**add(layers**.**Dense(256, activation**=**'relu')) model**.**add(layers**.**Dense(5, activation**=**'softmax')) model**.**summary()

Downloading data from https://storage.googleapis.com/tensorflow/keras-application s/resnet/resnet50v2\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5

94674944/94668760 [==============================] - 2s 0us/step

94683136/94668760 [==============================] - 2s 0us/step

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| resnet50v2 (Functional) | (None, | 4, 1, | 2048) | 23564800 |
| flatten (Flatten) | (None, | 8192) |  | 0 |
| dense (Dense) | (None, | 256) |  | 2097408 |
| dense\_1 (Dense) | (None, | 5) |  | 1285 |

=================================================================

Total params: 25,663,493

Trainable params: 25,618,053

Non-trainable params: 45,440

### Обучение модели

In [ ]:

*# Замораживание сверточной основы.*

conv\_base**.**trainable **= False**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 5, validation\_split **=** 0.2)

Epoch 1/5

626/626 [==============================] - 29s 26ms/step - loss: 1.2964 - accurac

y: 0.4819 - val\_loss: 1.7045 - val\_accuracy: 0.2511 Epoch 2/5

626/626 [==============================] - 14s 23ms/step - loss: 1.2252 - accurac

y: 0.5113 - val\_loss: 1.7106 - val\_accuracy: 0.2238 Epoch 3/5

626/626 [==============================] - 14s 22ms/step - loss: 1.1911 - accurac

y: 0.5249 - val\_loss: 1.5909 - val\_accuracy: 0.2957 Epoch 4/5

626/626 [==============================] - 15s 24ms/step - loss: 1.1656 - accurac

y: 0.5371 - val\_loss: 1.7434 - val\_accuracy: 0.2396 Epoch 5/5

626/626 [==============================] - 14s 23ms/step - loss: 1.1400 - accurac

y: 0.5448 - val\_loss: 1.6276 - val\_accuracy: 0.2903

In [ ]:

*# Разморозка сверточной основы.*

conv\_base**.**trainable **= True**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Реализация раннего прекращения.*

checkpoint\_filepath **=** './checkpoint\_ResNet50V2/'

model\_checkpoint\_cb **=** tf**.**keras**.**callbacks**.**ModelCheckpoint( filepath**=**checkpoint\_filepath,

save\_weights\_only**=True**,

save\_best\_only**=True**)

early\_stopping\_cb **=** keras**.**callbacks**.**EarlyStopping( patience**=**10,

restore\_best\_weights**=True**)

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 30, validation\_split **=** 0.2, callbac

*# Сохранение модели.*

model**.**save("ResNet50V2.h5")

*# Откат к наилучшей модели.*

model**.**load\_weights(checkpoint\_filepath)

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch 1/30  626/626 [==============================] - 71s 84ms/step | - loss: | 1.3228 | - accurac |
| y: 0.5051 - val\_loss: 1.5763 - val\_accuracy: 0.2603 Epoch 2/30  626/626 [==============================] - 52s 83ms/step | - loss: | 1.1102 | - accurac |
| y: 0.5709 - val\_loss: 1.4023 - val\_accuracy: 0.3355 |  |  |  |
| Epoch 3/30  626/626 [==============================] - 49s 79ms/step | - loss: | 1.0338 | - accurac |
| y: 0.6052 - val\_loss: 1.7224 - val\_accuracy: 0.2937 Epoch 4/30 |  |  |  |
| 626/626 [==============================] - 49s 78ms/step  y: 0.5547 - val\_loss: 1.4674 - val\_accuracy: 0.2945 Epoch 5/30  626/626 [==============================] - 49s 79ms/step | * loss: * loss: | 1.1499  1.0932 | * accurac * accurac |
| y: 0.5741 - val\_loss: 1.5473 - val\_accuracy: 0.2781 Epoch 6/30  626/626 [==============================] - 49s 79ms/step | - loss: | 1.0386 | - accurac |
| y: 0.5886 - val\_loss: 1.5171 - val\_accuracy: 0.3273 |  |  |  |
| Epoch 7/30  626/626 [==============================] - 51s 81ms/step | - loss: | 0.9769 | - accurac |
| y: 0.6161 - val\_loss: 1.3511 - val\_accuracy: 0.3710 Epoch 8/30 |  |  |  |
| 626/626 [==============================] - 49s 79ms/step  y: 0.6191 - val\_loss: 1.6220 - val\_accuracy: 0.2847 Epoch 9/30  626/626 [==============================] - 49s 79ms/step | * loss: * loss: | 0.9736  0.9702 | * accurac * accurac |
| y: 0.6184 - val\_loss: 1.4038 - val\_accuracy: 0.3652 Epoch 10/30  626/626 [==============================] - 49s 79ms/step | - loss: | 0.9551 | - accurac |
| y: 0.6290 - val\_loss: 1.5582 - val\_accuracy: 0.3500 |  |  |  |
| Epoch 11/30  626/626 [==============================] - 49s 79ms/step | - loss: | 0.9304 | - accurac |
| y: 0.6442 - val\_loss: 1.3544 - val\_accuracy: 0.3582 Epoch 12/30 |  |  |  |
| 626/626 [==============================] - 50s 79ms/step  y: 0.6465 - val\_loss: 1.4302 - val\_accuracy: 0.3718 Epoch 13/30  626/626 [==============================] - 49s 79ms/step | * loss: * loss: | 0.9118  0.8896 | * accurac * accurac |
| y: 0.6587 - val\_loss: 1.4249 - val\_accuracy: 0.3628 Epoch 14/30  626/626 [==============================] - 49s 79ms/step | - loss: | 0.8768 | - accurac |
| y: 0.6604 - val\_loss: 1.3904 - val\_accuracy: 0.3536 |  |  |  |
| Epoch 15/30  626/626 [==============================] - 49s 79ms/step | - loss: | 0.8627 | - accurac |
| y: 0.6664 - val\_loss: 1.8114 - val\_accuracy: 0.3153 Epoch 16/30 |  |  |  |
| 626/626 [==============================] - 49s 79ms/step  y: 0.6582 - val\_loss: 1.8094 - val\_accuracy: 0.3399 Epoch 17/30  626/626 [==============================] - 49s 79ms/step | * loss: * loss: | 0.9058  0.9006 | * accurac * accurac |
| y: 0.6484 - val\_loss: 1.5921 - val\_accuracy: 0.3530 |  |  |  |

Out[ ]:

In [ ]:

score **=** model**.**evaluate(X\_test, y\_test) score

<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fd440499690>

### Оценка работы модели

Out[ ]:

88/88 [==============================] - 2s 24ms/step - loss: 1.0848 - accuracy:

0.5502

[1.0847526788711548, 0.5501606464385986]

### Графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения

In [ ]:

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy'] loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

epochs **=** range(1, len(acc) **+** 1)

plt**.**plot(epochs, acc, 'bo', label**=**'Training acc')

plt**.**plot(epochs, val\_acc, 'b', label**=**'Validation acc') plt**.**title('Training and validation accuracy')

plt**.**legend()

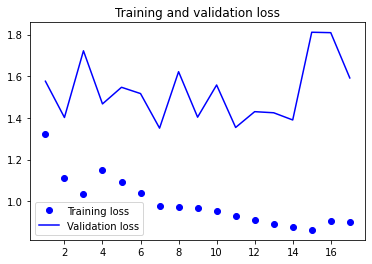
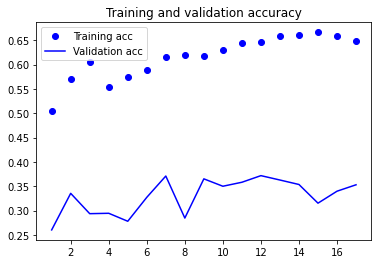
plt**.**figure()

plt**.**plot(epochs, loss, 'bo', label**=**'Training loss')

plt**.**plot(epochs, val\_loss, 'b', label**=**'Validation loss') plt**.**title('Training and validation loss')

plt**.**legend()

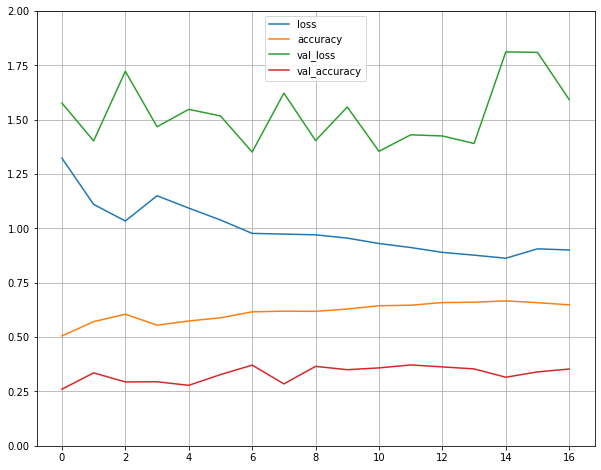
plt**.**show()



In [ ]:

pd**.**DataFrame(history**.**history)**.**plot(figsize **=** (10, 8)) plt**.**grid()

plt**.**gca()**.**set\_ylim(0, 2) plt**.**show()



## ResNet101

### Преобразование данных для работы с моделью

In [ ]:

X\_train **=** X\_trainCopyNorm

y\_train **=** y\_trainCopyEncoded X\_test **=** X\_testCopyNorm

y\_test **=** y\_testCopyEncoded

*# До преобразования.*

X\_train**.**shape, X\_test**.**shape

*# Преобразование формы тензора.*

X\_train **=** np**.**reshape(X\_train, (25025, **-**1, 32, 3))

X\_test **=** np**.**reshape(X\_test, (2801, **-**1, 32, 3))

*# После преобразования.*

X\_train**.**shape, X\_test**.**shape

Out[ ]:

In [ ]:

conv\_base **=** keras**.**applications**.**ResNet101(weights**=**'imagenet',

include\_top **= False**,

input\_shape **=** (125, 32, 3)

)

model **=** models**.**Sequential() model**.**add(conv\_base)

model**.**add(layers**.**Flatten())

model**.**add(layers**.**Dense(256, activation**=**'relu')) model**.**add(layers**.**Dense(5, activation**=**'softmax')) model**.**summary()

((25025, 125, 32, 3), (2801, 125, 32, 3))

### Загрузка модели

Downloading data from https://storage.googleapis.com/tensorflow/keras-application s/resnet/resnet101\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5

171450368/171446536 [==============================] - 3s 0us/step

171458560/171446536 [==============================] - 3s 0us/step

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| resnet101 (Functional) | (None, | 4, 1, | 2048) | 42658176 |
| flatten (Flatten) | (None, | 8192) |  | 0 |
| dense (Dense) | (None, | 256) |  | 2097408 |
| dense\_1 (Dense) | (None, | 5) |  | 1285 |

=================================================================

Total params: 44,756,869

Trainable params: 44,651,525

Non-trainable params: 105,344

### Обучение модели

In [ ]:

*# Замораживание сверточной основы.*

conv\_base**.**trainable **= False**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 5, validation\_split **=** 0.2)

Epoch 1/5

626/626 [==============================] - 48s 53ms/step - loss: 1.4747 - accurac

y: 0.4145 - val\_loss: 1.8435 - val\_accuracy: 0.0314 Epoch 2/5

626/626 [==============================] - 27s 43ms/step - loss: 1.4552 - accurac

y: 0.4199 - val\_loss: 2.1549 - val\_accuracy: 0.0314 Epoch 3/5

626/626 [==============================] - 26s 41ms/step - loss: 1.4517 - accurac

y: 0.4199 - val\_loss: 1.9002 - val\_accuracy: 0.0314 Epoch 4/5

626/626 [==============================] - 26s 41ms/step - loss: 1.4463 - accurac

y: 0.4199 - val\_loss: 1.9268 - val\_accuracy: 0.0314 Epoch 5/5

626/626 [==============================] - 26s 42ms/step - loss: 1.4400 - accurac

y: 0.4205 - val\_loss: 1.7263 - val\_accuracy: 0.0338

In [ ]:

*# Разморозка сверточной основы.*

conv\_base**.**trainable **= True**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Реализация раннего прекращения.*

checkpoint\_filepath **=** './checkpoint\_ResNet101/'

model\_checkpoint\_cb **=** tf**.**keras**.**callbacks**.**ModelCheckpoint( filepath**=**checkpoint\_filepath,

save\_weights\_only**=True**,

save\_best\_only**=True**)

early\_stopping\_cb **=** keras**.**callbacks**.**EarlyStopping( patience**=**10,

restore\_best\_weights**=True**)

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 30, validation\_split **=** 0.2, callbac

*# Сохранение модели.*

model**.**save("ResNet101.h5")

*# Откат к наилучшей модели.*

model**.**load\_weights(checkpoint\_filepath)

Out[ ]:

In [ ]:

score **=** model**.**evaluate(X\_test, y\_test) score

Epoch 1/30

626/626 [==============================] - 134s 156ms/step - loss: 1.2938 - accura

cy: 0.5159 - val\_loss: 1.5702 - val\_accuracy: 0.2615 Epoch 2/30

626/626 [==============================] - 91s 145ms/step - loss: 1.1369 - accurac

y: 0.5499 - val\_loss: 2.1732 - val\_accuracy: 0.2551 Epoch 3/30

626/626 [==============================] - 95s 152ms/step - loss: 1.0303 - accurac

y: 0.6003 - val\_loss: 1.5549 - val\_accuracy: 0.3524 Epoch 4/30

626/626 [==============================] - 94s 150ms/step - loss: 1.1067 - accurac

y: 0.5684 - val\_loss: 1.4837 - val\_accuracy: 0.2919 Epoch 5/30

626/626 [==============================] - 91s 145ms/step - loss: 1.1025 - accurac

y: 0.5668 - val\_loss: 1.5630 - val\_accuracy: 0.1337 Epoch 6/30

626/626 [==============================] - 91s 145ms/step - loss: 1.0549 - accurac

y: 0.5840 - val\_loss: 1.5579 - val\_accuracy: 0.3189 Epoch 7/30

626/626 [==============================] - 91s 146ms/step - loss: 0.9959 - accurac

y: 0.6090 - val\_loss: 1.8601 - val\_accuracy: 0.3285 Epoch 8/30

626/626 [==============================] - 91s 145ms/step - loss: 0.9685 - accurac

y: 0.6182 - val\_loss: 1.5453 - val\_accuracy: 0.2939 Epoch 9/30

626/626 [==============================] - 92s 146ms/step - loss: 0.9477 - accurac

y: 0.6256 - val\_loss: 1.6233 - val\_accuracy: 0.3421 Epoch 10/30

626/626 [==============================] - 91s 146ms/step - loss: 0.9575 - accurac

y: 0.6301 - val\_loss: 2.2444 - val\_accuracy: 0.2971 Epoch 11/30

626/626 [==============================] - 91s 146ms/step - loss: 0.9895 - accurac

y: 0.6159 - val\_loss: 1.8510 - val\_accuracy: 0.2186 Epoch 12/30

626/626 [==============================] - 92s 147ms/step - loss: 0.9748 - accurac

y: 0.6191 - val\_loss: 1.5552 - val\_accuracy: 0.3608 Epoch 13/30

626/626 [==============================] - 91s 146ms/step - loss: 1.0288 - accurac

y: 0.6016 - val\_loss: 1.5427 - val\_accuracy: 0.3367 Epoch 14/30

626/626 [==============================] - 92s 147ms/step - loss: 0.9578 - accurac

y: 0.6274 - val\_loss: 1.5564 - val\_accuracy: 0.3510

<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fe8b0ddce90>

### Оценка работы модели

Out[ ]:

In [ ]:

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy'] loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

epochs **=** range(1, len(acc) **+** 1)

plt**.**plot(epochs, acc, 'bo', label**=**'Training acc')

plt**.**plot(epochs, val\_acc, 'b', label**=**'Validation acc') plt**.**title('Training and validation accuracy')

plt**.**legend()

plt**.**figure()

plt**.**plot(epochs, loss, 'bo', label**=**'Training loss')

plt**.**plot(epochs, val\_loss, 'b', label**=**'Validation loss') plt**.**title('Training and validation loss')

plt**.**legend()

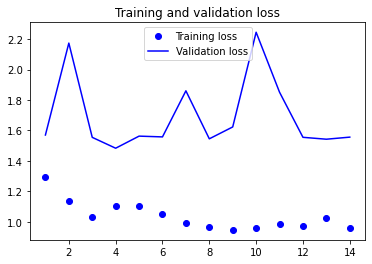
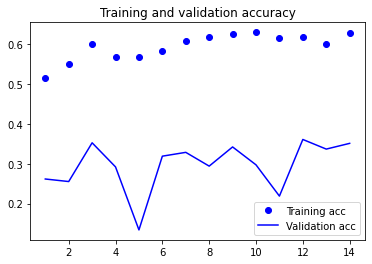
plt**.**show()

88/88 [==============================] - 3s 38ms/step - loss: 1.4233 - accuracy:

0.4223

[1.4233031272888184, 0.42234915494918823]

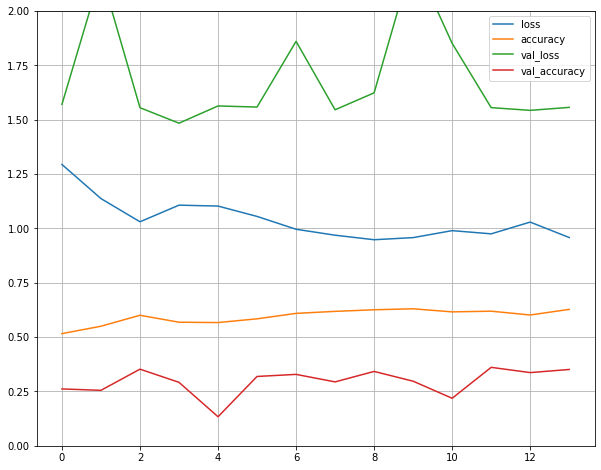
### Графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения



In [ ]:

pd**.**DataFrame(history**.**history)**.**plot(figsize **=** (10, 8)) plt**.**grid()

plt**.**gca()**.**set\_ylim(0, 2) plt**.**show()



## VGG16

### Преобразование данных для работы с моделью

In [11]:

X\_train **=** X\_trainCopyNorm

y\_train **=** y\_trainCopyEncoded X\_test **=** X\_testCopyNorm

y\_test **=** y\_testCopyEncoded

*# До преобразования.*

X\_train**.**shape, X\_test**.**shape

*# Преобразование формы тензора.*

X\_train **=** np**.**reshape(X\_train, (25025, **-**1, 32, 3))

X\_test **=** np**.**reshape(X\_test, (2801, **-**1, 32, 3))

*# После преобразования.*

X\_train**.**shape, X\_test**.**shape

Out[11]:

In [12]:

conv\_base **=** keras**.**applications**.**vgg16**.**VGG16(weights**=**'imagenet',

include\_top **= False**,

input\_shape **=** (125, 32, 3)

)

model **=** models**.**Sequential() model**.**add(conv\_base)

model**.**add(layers**.**Flatten())

model**.**add(layers**.**Dense(256, activation**=**'relu'))

((25025, 125, 32, 3), (2801, 125, 32, 3))

### Загрузка модели

model**.**add(layers**.**Dense(5, activation**=**'softmax')) model**.**summary()

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| vgg16 (Functional) | (None, | 3, 1, | 512) | 14714688 |
| flatten (Flatten) | (None, | 1536) |  | 0 |
| dense (Dense) | (None, | 256) |  | 393472 |
| dense\_1 (Dense) | (None, | 5) |  | 1285 |

=================================================================

Total params: 15,109,445

Trainable params: 15,109,445

Non-trainable params: 0

### Обучение модели

In [13]:

*# Заморозка весов заранее обученных слоев.*

**for** layer **in** conv\_base**.**layers: layer**.**trainable **= False**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 5, validation\_split **=** 0.2)

Epoch 1/5

626/626 [==============================] - 27s 25ms/step - loss: 1.4333 - accurac

y: 0.4279 - val\_loss: 1.8356 - val\_accuracy: 0.0655 Epoch 2/5

626/626 [==============================] - 14s 22ms/step - loss: 1.3976 - accurac

y: 0.4407 - val\_loss: 1.8282 - val\_accuracy: 0.0839 Epoch 3/5

626/626 [==============================] - 14s 22ms/step - loss: 1.3808 - accurac

y: 0.4469 - val\_loss: 1.6240 - val\_accuracy: 0.1678 Epoch 4/5

626/626 [==============================] - 14s 22ms/step - loss: 1.3655 - accurac

y: 0.4497 - val\_loss: 1.7195 - val\_accuracy: 0.1538 Epoch 5/5

626/626 [==============================] - 14s 22ms/step - loss: 1.3499 - accurac

y: 0.4560 - val\_loss: 1.8182 - val\_accuracy: 0.1469

In [14]:

*# Разморозка весов.*

**for** layer **in** conv\_base**.**layers: layer**.**trainable **= True**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Реализация раннего прекращения.*

checkpoint\_filepath **=** './checkpoint\_vgg16/'

model\_checkpoint\_cb **=** tf**.**keras**.**callbacks**.**ModelCheckpoint( filepath**=**checkpoint\_filepath,

save\_weights\_only**=True**, save\_best\_only**=True**)

early\_stopping\_cb **=** keras**.**callbacks**.**EarlyStopping( patience**=**10,

restore\_best\_weights**=True**)

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 30, validation\_split **=** 0.2, callbac

*# Сохранение модели.*

model**.**save("vgg16.h5")

*# Откат к наилучшей модели.*

model**.**load\_weights(checkpoint\_filepath)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Epoch 1/30  626/626 [==============================] - 47s 67ms/step | - | loss: | 23.1374 - accurac | |
| y: 0.4092 - val\_loss: 1.9009 - val\_accuracy: 0.0314  Epoch 2/30 |  |  |  | |
| 626/626 [==============================] - 40s 63ms/step  y: 0.4199 - val\_loss: 1.9069 - val\_accuracy: 0.0314 | - loss: | | 1.4589 | - accurac |
| Epoch 3/30  626/626 [==============================] - 40s 64ms/step | - loss: | | 1.4583 | - accurac |
| y: 0.4199 - val\_loss: 1.9212 - val\_accuracy: 0.0314 Epoch 4/30 |  | |  |  |
| 626/626 [==============================] - 40s 65ms/step  y: 0.4199 - val\_loss: 1.8956 - val\_accuracy: 0.0314 Epoch 5/30  626/626 [==============================] - 41s 65ms/step | * loss: * loss: | | 1.4578  1.4579 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.8860 - val\_accuracy: 0.0314 Epoch 6/30  626/626 [==============================] - 40s 65ms/step | - loss: | | 1.4951 | - accurac |
| y: 0.4158 - val\_loss: 1.8821 - val\_accuracy: 0.0314 |  | |  |  |
| Epoch 7/30  626/626 [==============================] - 40s 64ms/step | - loss: | | 1.4580 | - accurac |
| y: 0.4199 - val\_loss: 1.9352 - val\_accuracy: 0.0314 Epoch 8/30 |  | |  |  |
| 626/626 [==============================] - 40s 63ms/step  y: 0.4199 - val\_loss: 1.9299 - val\_accuracy: 0.0314 Epoch 9/30  626/626 [==============================] - 40s 63ms/step | * loss: * loss: | | 1.4573  1.4572 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.9266 - val\_accuracy: 0.0314 Epoch 10/30  626/626 [==============================] - 40s 64ms/step | - loss: | | 1.4570 | - accurac |
| y: 0.4199 - val\_loss: 1.8980 - val\_accuracy: 0.0314 |  | |  |  |
| Epoch 11/30  626/626 [==============================] - 40s 64ms/step | - loss: | | 1.4575 | - accurac |
| y: 0.4199 - val\_loss: 1.9423 - val\_accuracy: 0.0314 Epoch 12/30 |  | |  |  |
| 626/626 [==============================] - 40s 65ms/step  y: 0.4199 - val\_loss: 1.8710 - val\_accuracy: 0.0314 Epoch 13/30  626/626 [==============================] - 40s 63ms/step | * loss: * loss: | | 1.4571  1.4569 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.9401 - val\_accuracy: 0.0314 Epoch 14/30  626/626 [==============================] - 40s 63ms/step | - loss: | | 1.5373 | - accurac |
| y: 0.4161 - val\_loss: 1.8711 - val\_accuracy: 0.0314 |  | |  |  |
| Epoch 15/30  626/626 [==============================] - 41s 65ms/step | - loss: | | 1.4573 | - accurac |
| y: 0.4199 - val\_loss: 1.8701 - val\_accuracy: 0.0314 Epoch 16/30 |  | |  |  |
| 626/626 [==============================] - 40s 65ms/step  y: 0.4199 - val\_loss: 1.8592 - val\_accuracy: 0.0314 Epoch 17/30  626/626 [==============================] - 40s 63ms/step | * loss: * loss: | | 1.4566  1.4566 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.9045 - val\_accuracy: 0.0314 Epoch 18/30  626/626 [==============================] - 40s 64ms/step | - loss: | | 1.4562 | - accurac |
| y: 0.4199 - val\_loss: 1.9091 - val\_accuracy: 0.0314 |  | |  |  |
| Epoch 19/30  626/626 [==============================] - 40s 63ms/step | - loss: | | 1.4567 | - accurac |
| y: 0.4199 - val\_loss: 1.9138 - val\_accuracy: 0.0314 Epoch 20/30 |  | |  |  |
| 626/626 [==============================] - 40s 63ms/step  y: 0.4199 - val\_loss: 1.8908 - val\_accuracy: 0.0314 Epoch 21/30  626/626 [==============================] - 40s 63ms/step | * loss: * loss: | | 1.4565  1.4564 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.9098 - val\_accuracy: 0.0314  Epoch 22/30 |  | |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 626/626 [==============================] - 40s 64ms/step  y: 0.4199 - val\_loss: 1.8883 - val\_accuracy: 0.0314 | - loss: | 1.4566 | - accurac |
| Epoch 23/30  626/626 [==============================] - 40s 63ms/step | - loss: | 1.6918 | - accurac |
| y: 0.4183 - val\_loss: 1.9246 - val\_accuracy: 0.0314 Epoch 24/30 |  |  |  |
| 626/626 [==============================] - 40s 63ms/step  y: 0.4199 - val\_loss: 1.9335 - val\_accuracy: 0.0314 Epoch 25/30  626/626 [==============================] - 40s 63ms/step | * loss: * loss: | 1.4571  1.4566 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.8728 - val\_accuracy: 0.0314 Epoch 26/30  626/626 [==============================] - 40s 63ms/step | - loss: | 1.4567 | - accurac |
| y: 0.4199 - val\_loss: 1.9183 - val\_accuracy: 0.0314 |  |  |  |

Out[14]:

In [15]:

score **=** model**.**evaluate(X\_test, y\_test) score

<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f1e6df9a9d0>

### Оценка работы модели

Out[15]:

In [16]:

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy'] loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

epochs **=** range(1, len(acc) **+** 1)

plt**.**plot(epochs, acc, 'bo', label**=**'Training acc')

plt**.**plot(epochs, val\_acc, 'b', label**=**'Validation acc') plt**.**title('Training and validation accuracy')

plt**.**legend()

plt**.**figure()

plt**.**plot(epochs, loss, 'bo', label**=**'Training loss')

plt**.**plot(epochs, val\_loss, 'b', label**=**'Validation loss') plt**.**title('Training and validation loss')

plt**.**legend()

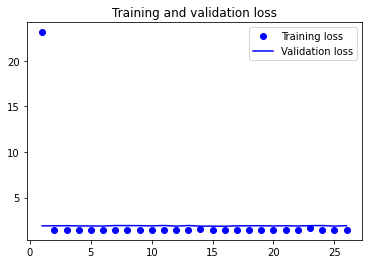
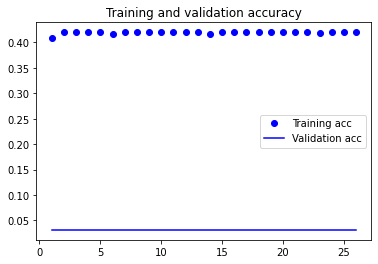
plt**.**show()

88/88 [==============================] - 2s 24ms/step - loss: 1.5352 - accuracy:

0.3442

[1.535159707069397, 0.34416279196739197]

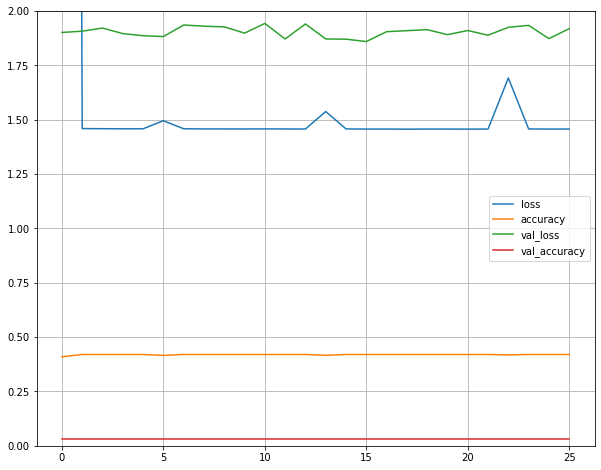
### Графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения



In [17]:

pd**.**DataFrame(history**.**history)**.**plot(figsize **=** (10, 8)) plt**.**grid()

plt**.**gca()**.**set\_ylim(0, 2) plt**.**show()



## VGG19

### Преобразование данных для работы с моделью

In [ ]:

X\_train **=** X\_trainCopyNorm

y\_train **=** y\_trainCopyEncoded X\_test **=** X\_testCopyNorm

y\_test **=** y\_testCopyEncoded

*# До преобразования.*

X\_train**.**shape, X\_test**.**shape

*# Преобразование формы тензора.*

X\_train **=** np**.**reshape(X\_train, (25025, **-**1, 32, 3))

X\_test **=** np**.**reshape(X\_test, (2801, **-**1, 32, 3))

*# После преобразования.*

X\_train**.**shape, X\_test**.**shape

Out[ ]:

In [ ]:

conv\_base **=** keras**.**applications**.**vgg19**.**VGG19(weights**=**'imagenet',

include\_top **= False**,

input\_shape **=** (125, 32, 3)

)

model **=** models**.**Sequential() model**.**add(conv\_base)

model**.**add(layers**.**Flatten())

model**.**add(layers**.**Dense(256, activation**=**'relu')) model**.**add(layers**.**Dense(5, activation**=**'softmax')) model**.**summary()

((25025, 125, 32, 3), (2801, 125, 32, 3))

### Загрузка модели

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| vgg19 (Functional) | (None, | 3, 1, | 512) | 20024384 |
| flatten (Flatten) | (None, | 1536) |  | 0 |
| dense (Dense) | (None, | 256) |  | 393472 |
| dense\_1 (Dense) | (None, | 5) |  | 1285 |

=================================================================

Total params: 20,419,141

Trainable params: 20,419,141

Non-trainable params: 0

### Обучение модели

In [ ]:

*# Замораживание сверточной основы.*

conv\_base**.**trainable **= False**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 5, validation\_split **=** 0.2)

Epoch 1/5

626/626 [==============================] - 28s 28ms/step - loss: 1.4324 - accurac

y: 0.4257 - val\_loss: 1.7885 - val\_accuracy: 0.0999 Epoch 2/5

626/626 [==============================] - 16s 26ms/step - loss: 1.3931 - accurac

y: 0.4437 - val\_loss: 1.8642 - val\_accuracy: 0.0991 Epoch 3/5

626/626 [==============================] - 19s 30ms/step - loss: 1.3770 - accurac

y: 0.4474 - val\_loss: 1.7624 - val\_accuracy: 0.1325 Epoch 4/5

626/626 [==============================] - 17s 27ms/step - loss: 1.3656 - accurac

y: 0.4565 - val\_loss: 1.7601 - val\_accuracy: 0.1327 Epoch 5/5

626/626 [==============================] - 16s 26ms/step - loss: 1.3592 - accurac

y: 0.4600 - val\_loss: 1.9278 - val\_accuracy: 0.1011

In [ ]:

*# Разморозка весов.*

conv\_base**.**trainable **= True**

*# Компиляция модели.*

model**.**compile(optimizer **=** tf**.**keras**.**optimizers**.**Nadam(), loss**=**'sparse\_categorical\_crossentropy', metrics**=**['accuracy'])

*# Реализация раннего прекращения.*

checkpoint\_filepath **=** './checkpoint\_vgg19/'

model\_checkpoint\_cb **=** tf**.**keras**.**callbacks**.**ModelCheckpoint( filepath**=**checkpoint\_filepath,

save\_weights\_only**=True**, save\_best\_only**=True**)

early\_stopping\_cb **=** keras**.**callbacks**.**EarlyStopping(

patience**=**10,

restore\_best\_weights**=True**)

*# Обучение.*

history **=** model**.**fit(X\_train, y\_train, epochs **=** 30, validation\_split **=** 0.2, callbac

*# Сохранение модели.*

model**.**save('vgg19.h5')

*# Откат к наилучшей модели.*

model**.**load\_weights(checkpoint\_filepath)

Epoch 1/30

626/626 [==============================] - 58s 84ms/step - loss: 38.9503 - accurac

y: 0.4085 - val\_loss: 1.9018 - val\_accuracy: 0.0314 Epoch 2/30

|  |  |  |  |
| --- | --- | --- | --- |
| 626/626 [==============================] - 49s 78ms/step  y: 0.4199 - val\_loss: 1.9127 - val\_accuracy: 0.0314 Epoch 3/30  626/626 [==============================] - 50s 80ms/step | * loss: * loss: | 1.4581  1.4576 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.8677 - val\_accuracy: 0.0314 Epoch 4/30  626/626 [==============================] - 50s 79ms/step | - loss: | 1.4570 | - accurac |
| y: 0.4199 - val\_loss: 1.8642 - val\_accuracy: 0.0314 |  |  |  |
| Epoch 5/30  626/626 [==============================] - 49s 78ms/step | - loss: | 1.4576 | - accurac |
| y: 0.4199 - val\_loss: 1.8991 - val\_accuracy: 0.0314  Epoch 6/30 |  |  |  |
| 626/626 [==============================] - 49s 78ms/step  y: 0.4199 - val\_loss: 1.8772 - val\_accuracy: 0.0314 Epoch 7/30  626/626 [==============================] - 49s 78ms/step | * loss: * loss: | 1.4570  1.4580 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.8882 - val\_accuracy: 0.0314 Epoch 8/30  626/626 [==============================] - 51s 81ms/step | - loss: | 1.4569 | - accurac |
| y: 0.4199 - val\_loss: 1.8945 - val\_accuracy: 0.0314 |  |  |  |
| Epoch 9/30  626/626 [==============================] - 51s 81ms/step | - loss: | 1.4571 | - accurac |
| y: 0.4199 - val\_loss: 1.9147 - val\_accuracy: 0.0314  Epoch 10/30 |  |  |  |
| 626/626 [==============================] - 49s 78ms/step  y: 0.4199 - val\_loss: 1.8776 - val\_accuracy: 0.0314 Epoch 11/30  626/626 [==============================] - 49s 78ms/step | * loss: * loss: | 1.4570  1.4572 | * accurac * accurac |
| y: 0.4199 - val\_loss: 1.8994 - val\_accuracy: 0.0314 Epoch 12/30  626/626 [==============================] - 49s 78ms/step | - loss: | 1.4570 | - accurac |
| y: 0.4199 - val\_loss: 1.9040 - val\_accuracy: 0.0314 |  |  |  |
| Epoch 13/30  626/626 [==============================] - 48s 77ms/step | - loss: | 1.4567 | - accurac |
| y: 0.4199 - val\_loss: 1.9433 - val\_accuracy: 0.0314  Epoch 14/30 |  |  |  |
| 626/626 [==============================] - 49s 78ms/step  y: 0.4199 - val\_loss: 1.9358 - val\_accuracy: 0.0314 | - loss: | 1.4567 | - accurac |

Out[ ]:

In [ ]:

<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f61c7fc6550>

### Оценка работы модели

88/88 [==============================] - 2s 27ms/step - loss: 1.5358 - accuracy:



score **=** model**.**evaluate(X\_test, y\_test) score

0.3442

Out[ ]:

In [ ]:

acc **=** history**.**history['accuracy']

val\_acc **=** history**.**history['val\_accuracy'] loss **=** history**.**history['loss']

val\_loss **=** history**.**history['val\_loss']

epochs **=** range(1, len(acc) **+** 1)

plt**.**plot(epochs, acc, 'bo', label**=**'Training acc')

plt**.**plot(epochs, val\_acc, 'b', label**=**'Validation acc') plt**.**title('Training and validation accuracy')

plt**.**legend()

plt**.**figure()

plt**.**plot(epochs, loss, 'bo', label**=**'Training loss')

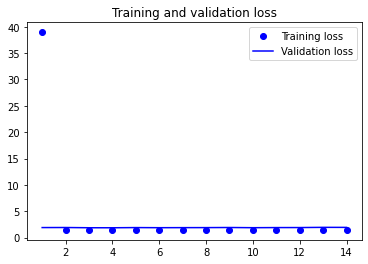
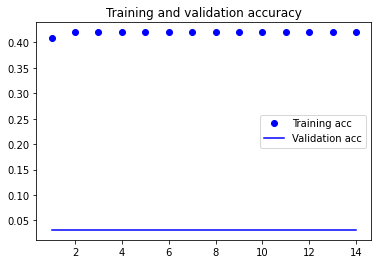
plt**.**plot(epochs, val\_loss, 'b', label**=**'Validation loss') plt**.**title('Training and validation loss')

plt**.**legend()

plt**.**show()

[1.5358237028121948, 0.34416279196739197]

### Графики изменения точности и потерь модели по обучающим и проверочным данным в процессе обучения



In [ ]:

pd**.**DataFrame(history**.**history)**.**plot(figsize **=** (10, 8)) plt**.**grid()

plt**.**gca()**.**set\_ylim(0, 2) plt**.**show()

