ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**Отчет о прохождении учебной практики**

|  |
| --- |
| Хвацкин Леонид Иосифович |

*(Ф.И.О. обучающегося)*

|  |
| --- |
| 2 курс, гр.3530901/80003 |

|  |
| --- |
| 09.03.01 «Информатика и вычислительная техника» |

*(Направление подготовки (код и наименование)*

|  |
| --- |
| **Место прохождения практики**: Высшая школа информационных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ» с использованием электронного обучения и дистанционных образовательных технологий. |

|  |
| --- |
| **Сроки практики:** с 22 июня по 18 июля 2020 г. |

|  |
| --- |
| **Руководитель практики:** |

|  |
| --- |
| Жвариков В. А., к.т.н., доцент ВШИСиСТ ИКНТ |

*(Ф.И.О., уч.степень, должность)*

|  |
| --- |
| **Оценка (зачет): \_\_\_\_\_\_\_\_\_\_\_** |

|  |
| --- |
|  |

|  |
| --- |
| Руководитель практики: Жвариков В. А. |

|  |
| --- |
| Обучающийся: Хвацкин Л. И. |

|  |
| --- |
| Дата: |

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**ИНДИВИДУАЛЬНЫЙ ПЛАН (ЗАДАНИЕ И ГРАФИК)**

**ПРОВЕДЕНИЯ ПРАКТИКИ**

|  |
| --- |
| Ф.И.О. обучающегося Хвацкин Леонид Иосифович |

|  |
| --- |
| **Направление подготовки** (код/наименование): 09.03.01, «Информатика и вычислительная техника» |
| **Профиль** (код/наименование): 09.03.01, «Информатика и вычислительная техника» |
| **Вид практики:** учебная |
| **Тип практики:** ознакомительная |
| **Место прохождения практики**: Высшая школа информационных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ» с использованием электронного обучения и дистанционных образовательных технологий. |

|  |
| --- |
|  |

|  |
| --- |
|  |
| Руководитель практики: Жвариков В.А., к.т.н., доцент ВШИСиСТ ИКНТ |
| *(Ф.И.О., уч.степень, должность)* |

**Рабочий график проведения учебной практики**

Сроки практики: с **22.06.2020** г. по **18.07.2020** г.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Этапы (периоды) практики | Вид работ | Сроки прохождения этапа (периода) практики |
| 1 | Организационный этап | Установочная лекция (вебинар) для разъяснения целей, задач, содержания и порядка прохождения практики, выдача сопроводительных документов по практике  Основная тема: Изучение технологий машинного обучения с применением модулей tensorflow и keras языка Python. Реализация классификатора рукописных цифр. | 22.06.2020 г. |
| 2 | Основной  этап | Изучение предлагаемого модулями tensorflow и keras набора функций для реализации машинного обучения.  Сбор информации, обработка, систематизация и анализ фактического и теоретического материала.  Содержание практики: Анализ, пре-процессинг, чистка исходных данных (база данных <http://yann.lecun.com/exdb/mnist>). Выделение опорных признаков, анализ распределения данных по признакам, выбор соответствующей архитектуры модели. Построение модели, обучение, тестирование; внесение поправок в модель до достижения приемлемой точности.  Планируемые результаты прохождения практики: Освоение основных технологий и принципов машинного обучения. Работающий с высокой точностью классификатор рукописных цифр. | 23.06.2020 - 15.07.2020 г. |
| 3 | Заключительный этап | Подготовка отчета | 16.07.- 17.07.2020 г. |
| Защита отчета по практике (сдача зачета) | 17.07 - 18.07.2020 г |

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Хвацкин Л. И./

Руководитель практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Жвариков В.А./

**Оглавление**

[Техническое задание 3](#_Toc45732909)

[Метод решения 3](#_Toc45732910)

[Терминология 3](#_Toc45732911)

[Исходные данные 4](#_Toc45732912)

[Предварительное изучение данных 5](#_Toc45732913)

[Работа над моделью 6](#_Toc45732914)

[Модель №1 7](#_Toc45732915)

[Модель №2 10](#_Toc45732916)

[Модель №3 12](#_Toc45732917)

[Модель №4 14](#_Toc45732918)

[Модель №5 16](#_Toc45732919)

[Модель №6 18](#_Toc45732920)

[Модель №7 19](#_Toc45732921)

[Модель №8 21](#_Toc45732922)

[Модель №9 23](#_Toc45732923)

[Модель №10 24](#_Toc45732924)

[Модель №11 25](#_Toc45732925)

[Модель №12 (финальная) 26](#_Toc45732926)

[Консольное приложение 28](#_Toc45732927)

[Работа программы 29](#_Toc45732928)

[Использованные источники 30](#_Toc45732929)

[Приложения 30](#_Toc45732930)

[Файл обучения модели Main.py 30](#_Toc45732931)

[Файл консольного приложения AppMain.py 33](#_Toc45732932)

# Техническое задание

Цель работы: разработка консольного приложения, работающего на базе технологии машинного обучения, на языке Python с помощью среды разработки PyCharm.

Идея заключалась в создании приложения для распознания написанной от руки цифры. Цифра пишется от руки любым доступным способом, затем надпись сохраняется в виде изображения в формате .jpg либо .png, и путь к этому файлу задаётся через консоль; приложение обрабатывает изображение и выводит в консоль предположение о том, какая цифра написана.

GitHub репозиторий проекта: <https://github.com/TheGreenBeaver/MNIST_Nums>

# Метод решения

## Терминология

* Признаки – числовые значения, поступающие на вход модели с тем, чтобы она подобрала для каждого из них коэффициент, позволяющий на выходе получить значение, максимально близкое к метке.
* Метка – предварительно обозначенное верное значение, к которому должна стремиться модель, получившая на вход соответствующий набор величин признаков.
* Переобучение – ситуация, когда модель «заучивает» признаки для обучающей выборки, так что для неё результаты оказываются сверхточными, но для новых данных – не точными совсем.
* Функция регуляризации – функция, ограждающая модель от переобучения за счёт искусственного наложения на веса признаков требования к тому, чтобы они оказались распределены примерно равномерно и оставались малы по модулю.
* Функция потерь – функционал, описывающий тем или иным способом, насколько текущий результат работы модели отличается от ожидаемого согласно метке. Функция потерь должна быть тем меньше, чем точнее сработала модель.
* Пакет (Batch) – некая часть исходной выборки, по которой модель единовременно совершает расчёты. Если на каждом шаге вычислять все необходимые данные по всей выборке, модель будет обучаться очень точно, но очень долго. Экспериментальным путём установлено, что до некоторого момента размер учитываемой части можно уменьшать без значительного ущерба точности, но зато значительно сокращая время обучения.
* Функция (алгоритм) оптимизации – правило, согласно которому на каждом шаге обновляются веса признаков.
* Эпоха – итерация обучения модели.
* Валидационная выборка – искусственно отделяемая часть обучающей выборки, которая в процессе обучения модели используется для проверки её работы с новыми данными.

## Исходные данные

Датасет MNIST состоит из обучающей выборки величиной в 60000 изображений и тестовой выборки величиной в 10000 изображений. Каждое изображение представляет собой квадрат со стороной 28 пикселей и отцентрировано относительно своего центра масс.

Этот датасет – один из «канонических»: им часто пользуются в простых учебных проектах либо для тестирования и отладки сложных систем. По этой причине в библиотеке tensorflow.keras предусмотрена возможность быстрого доступа к нему с помощью функции tensorflow.keras.datasets.mnist.load\_data(). Функция возвращает четыре объекта: массивы с обучающей и тестовой выборками изображений и массивы меток для них. Каждое изображение представлено в виде матрицы значений от 0 до 255, означающих по аналогии с форматом RGB интенсивность серого цвета в соответствующем пикселе. Это довольно большой разброс значений, что затрудняет для модели подбор разумных коэффициентов и может привести к некорректному, замедленному обучению, загнать модель в локальный минимум в результате обнуления весов или превращения их в бесконечность или NaN. Поэтому все значения в матрицах перед подачей на вход модели делим на 255 – таким образом, они окажутся в удобном узком диапазоне от 0 до 1.

## Предварительное изучение данных

Прежде чем строить модель, следует получить некоторое представление о датасете.

Каждое изображение по сути – вектор в -мерном пространстве. Воспользуемся методом главных компонент (sklearn.decomposition.PCA()), чтобы понизить размерность до 2 – это позволит нам представить данные на двухмерном графике.

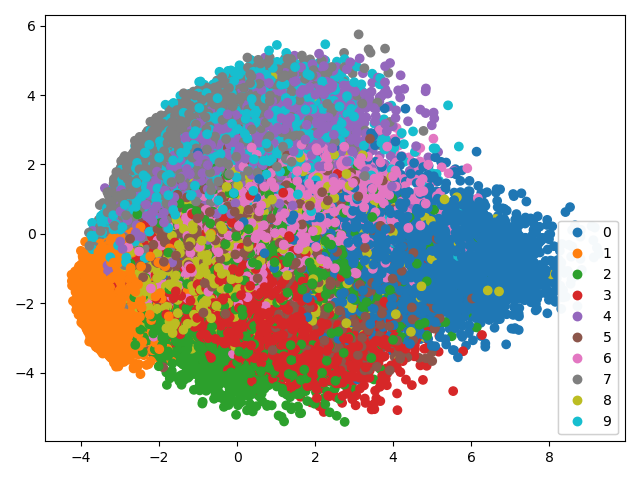


Рис. 1. Представление данных в двухмерном эквиваленте

Видно, что относительно хорошо отделены лишь области с метками 0 и 1. Это означает, что можно сразу отказаться от линейной модели: чтобы с высокой точностью разделить данные, придётся создавать нелинейные слои либо искусственно повышать размерность за счёт дополнительных слоёв и нейронов.

Проверим также, нельзя ли отбросить некоторую часть признаков.

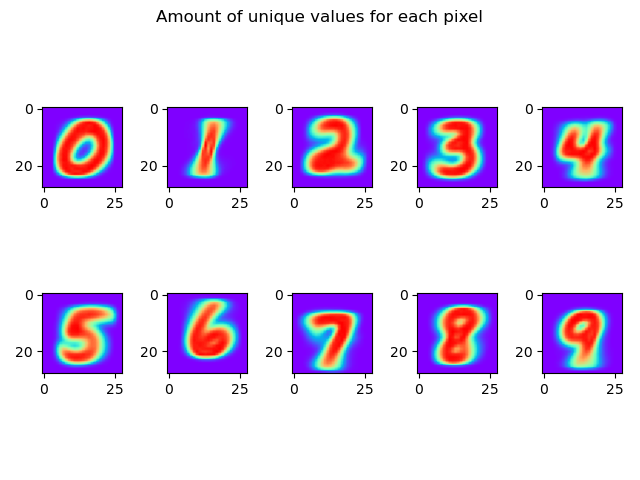


Рис. 2. Количество различных значений для каждого признака в зависимости от метки

Группы признаков с наибольшей вариативностью значений (следовательно, несущих наибольшее количество информации) расположены для разных цифр весьма различно – таким образом, отказ от каких-либо признаков представляется проблематичным и, учитывая изначально не слишком большое количество признаков, нерелевантным.

## Работа над моделью

Метки в исходном датасете – числа от 0 до 9. Однако, если строить модель таким образом, чтобы на основании значений признаков получалось конкретное число, есть риск при работе на новых данных получить результат, не попадающий в предполагаемый диапазон. Лучше рассматривать проблему как задачу классификации – требовать от модели 10 значений, которые бы означали «вероятность» того, что на предложенном изображении определённая цифра.

Исходя из таких соображений, в качестве функции потерь была выбрана дискретная перекрёстная энтропия (keras.losses.SparseCategoricalCrossentropy()), а в список отслеживаемых метрик, кроме значения этой самой функции потерь, был включён её несмещённый эквивалент (keras.metrics.SparseCategoricalCrossentropy()) и дискретная категориальная точность (keras.metrics.SparseTopKCategoricalAccuracy()). Это означает, что модель каждую метку переводит в массив, заполненный нулями кроме индекса, соответствующего значению метки. Чтобы обеспечить модели возможность сравнения результата работы с таким представлением метки, её архитектура всегда строится таким образом, чтобы получать 784 признака на вход и возвращать 10 чисел на выходе.

Однозначно угадать точную архитектуру модели и все параметры с первого раза практически невозможно. Изначально была построена простейшая модель, затем она так или иначе изменялась в зависимости от результирующих показателей. Здесь представлены данные о двенадцати основных этапах работы.

### Модель №1

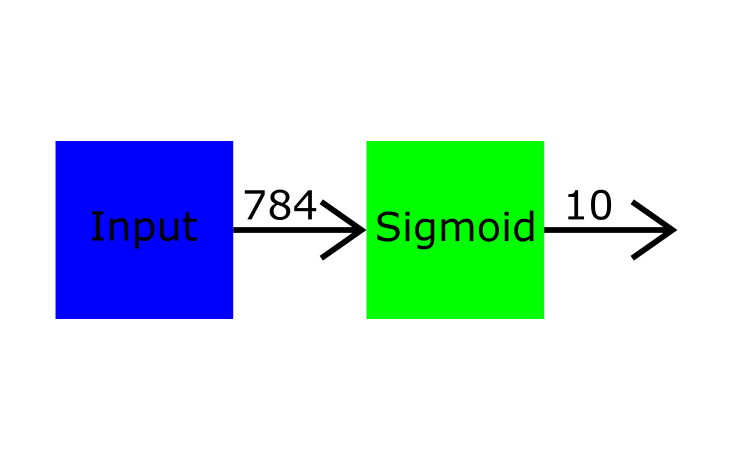


Рис. 3. Схематическое представление первоначальной модели

Первая модель состояла всего из одного слоя. В качестве нелинейной функции активации была выбрана сигмоидная: ; это удобно по той причине, что данная функция может возвращать значения только в диапазоне от 0 до 1, вследствие чего их легко интерпретировать как вероятности. С использованием сигмоида, однако, сопряжён риск ухода в асимптоты – из-за этого на веса необходимо накладывать довольно жёсткую регуляризацию.

Параметры модели:

* Batch size: 10
* Количество эпох: 10
* Validation split: 0.2
* Регуляризация:
  + Внутри единственного слоя: l1 + l2 (keras.regularizers.l1\_l2()), l1\_rate = 0.001, l2\_rate = 0.0001
* Оптимизация: стохастический градиентный спуск (tf.keras.optimizers.SGD()) без ускорения по Нестерову, learning rate = 0.01

Результаты работы модели:

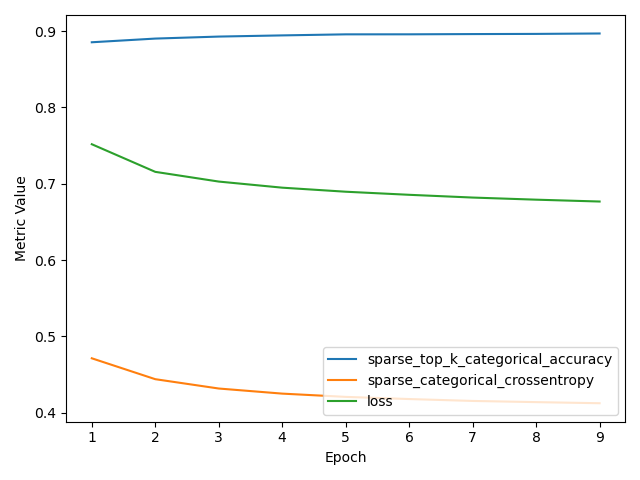


Рис. 4. Значения метрик по мере обучения модели

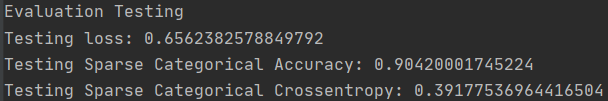
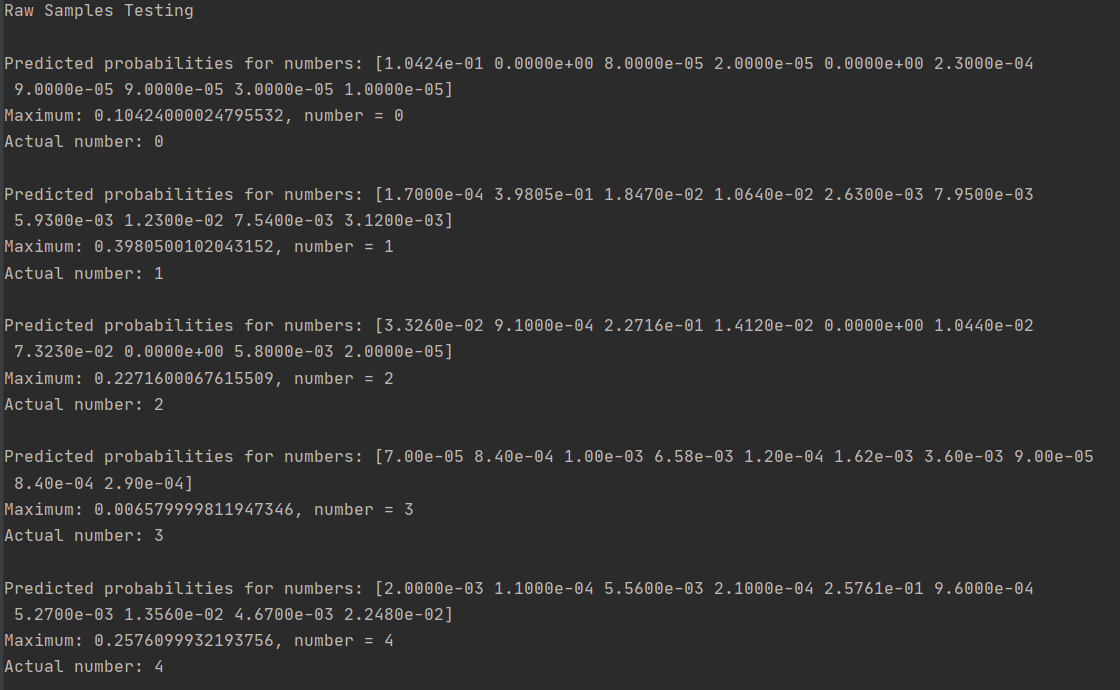


Рис. 5. Значения метрик для тестовой выборки



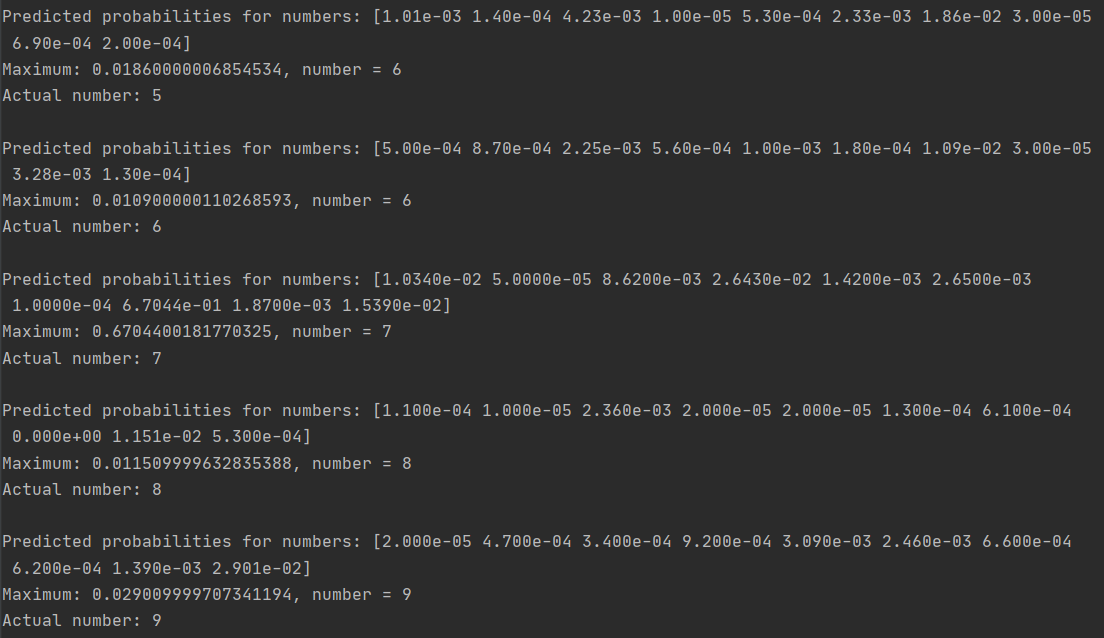


Рис. 6-7. Результаты работы модели для конкретных экземпляров тестовой выборки

Точность уже довольно велика (90.4%), хоть и далека от оптимальной. Однако значения других метрик говорят о том, что такой результат – скорее везение, чем закономерность, а также сигнализируют о наличии большого смещения. Кроме того, как видно при тестировании модели на конкретных экземплярах, даже максимальные вычисленные ею вероятности оказываются чрезвычайно малы – это делает модель не слишком удобной для расшифровки.

### Модель №2

Первая модель обучалась всего 10 эпох, однако графики её метрик успели практически выродиться в параллельные оси абсцисс линии. По этой причине была предпринята попытка одновременно повысить скорость обучения, разрешить ускорение по Нестерову и увеличить количество эпох; также была понижена степень регуляризации – это может помочь выправить экстремально малые результирующие величины.

Изменённые относительно модели №1 параметры:

* Количество эпох: 50
* Регуляризация:
  + l1\_rate = 0.00001, l2\_rate = 0. 00000001
* Оптимизация: стохастический градиентный спуск с ускорением по Нестерову, learning rate = 0.07

Результаты работы модели:

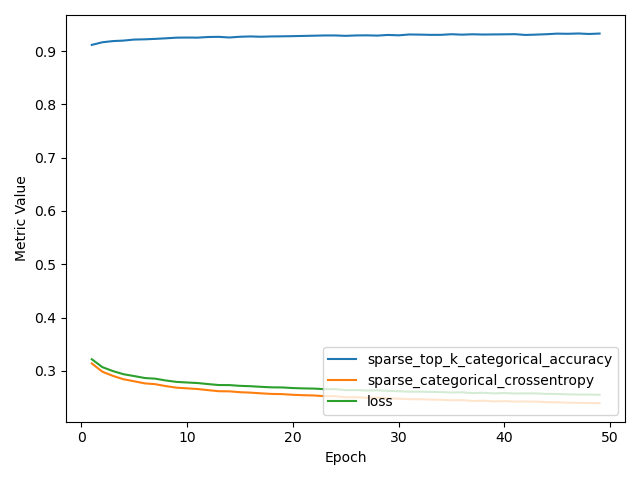


Рис. 8. Значения метрик по мере обучения модели

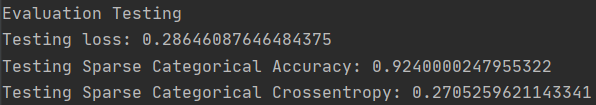
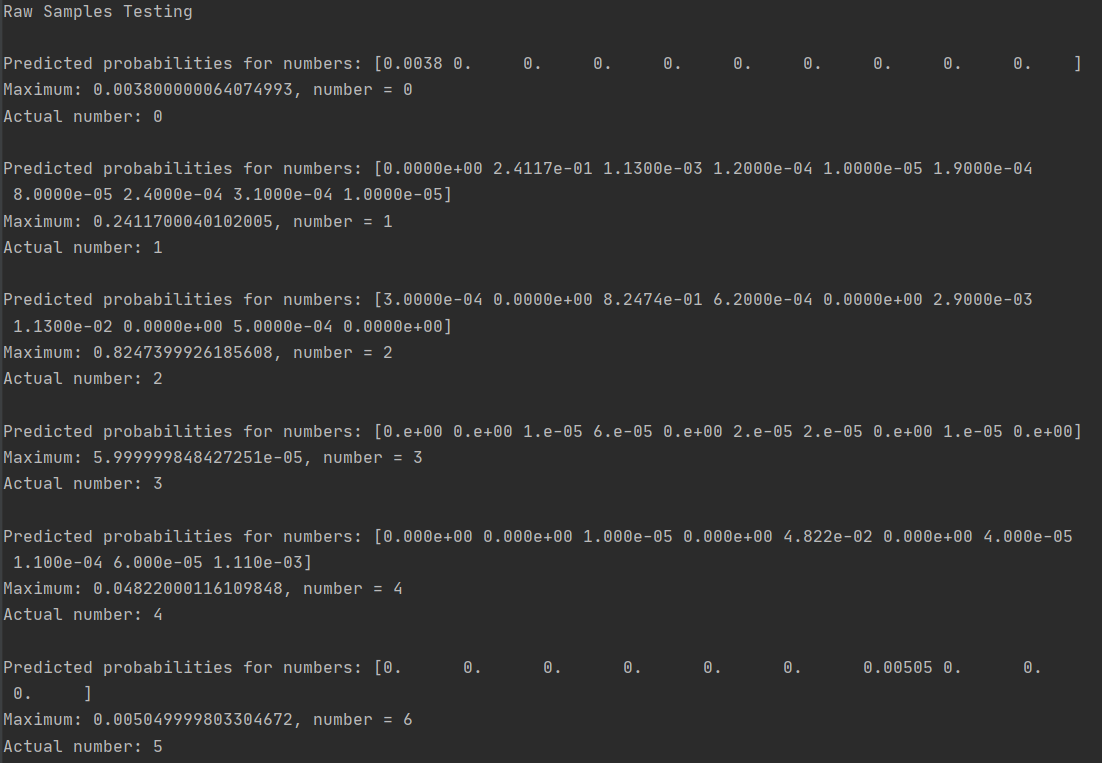


Рис. 9. Значения метрик для тестовой выборки



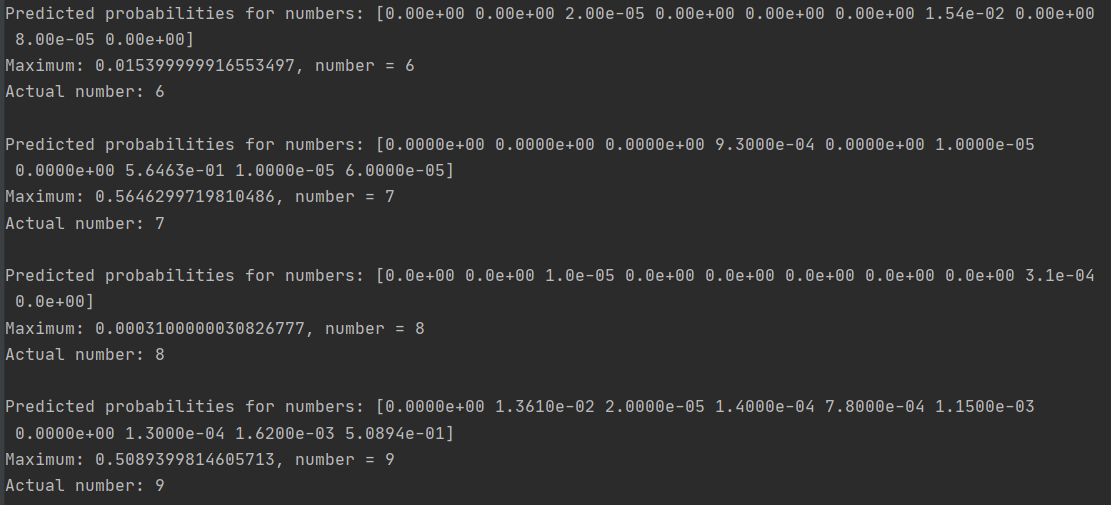


Рис. 10-11. Результаты работы модели на конкретных экземплярах тестовой выборки

Нам удалось значительно уменьшить энтропию и смещение, однако точность повысилась едва ли на 2%, и результирующие значения всё ещё выглядят очень неаккуратно.

### Модель №3

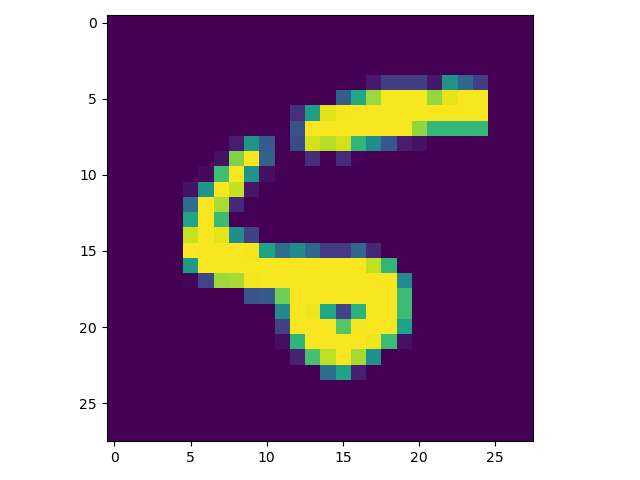


Рис. 12. Изображение, на котором до сих пор ошибались все модели

Данное изображение помечено как пятёрка, однако модель оценивает её как шестёрку, хоть и с очень низкой вероятностью. Очевидно, что такое написание немного нестандартно – можно сказать, что данное изображение является выбросом. Получается, одна из проблем нынешней архитектуры – в недостаточном учёте тех значений признаков, которые встречаются редко. Следующие изображения, также неверно классифицированные, подтверждают такое предположение: модель не запоминает редко используемую в Америке «перекладину» семёрки, «треугольную» четвёрку или восьмёрку с более крупным верхним кольцом.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Чтобы исправить это, попробуем поменять алгоритм оптимизации. Также увеличим batch size, чтобы несколько повысить точность расчёта градиента для этого алгоритма.

Изменённые относительно модели №2 параметры:

* Batch size: 20
* Количество эпох: 70
* Регуляризация:
  + l1\_rate = 0.0001, l2\_rate = 0.000000001
* Оптимизация: NAdam (keras.optimizers.NAdam()), learning rate = 0.07,

Результаты работы модели:

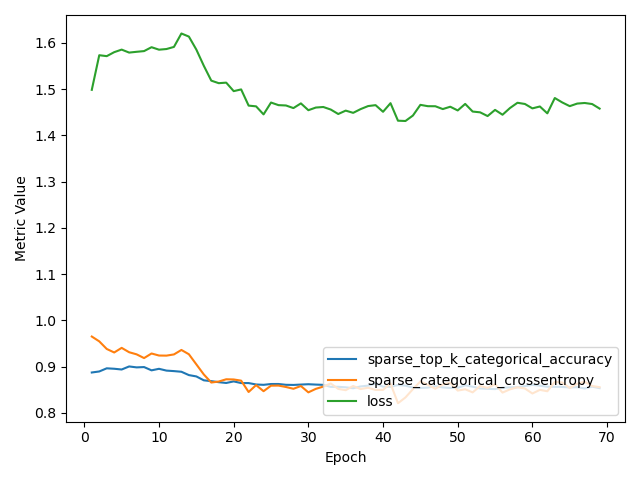


Рис. 13. Значения метрик по мере обучения модели

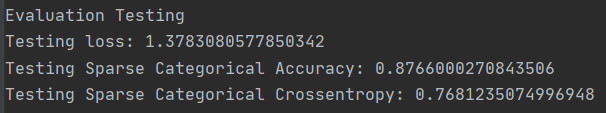
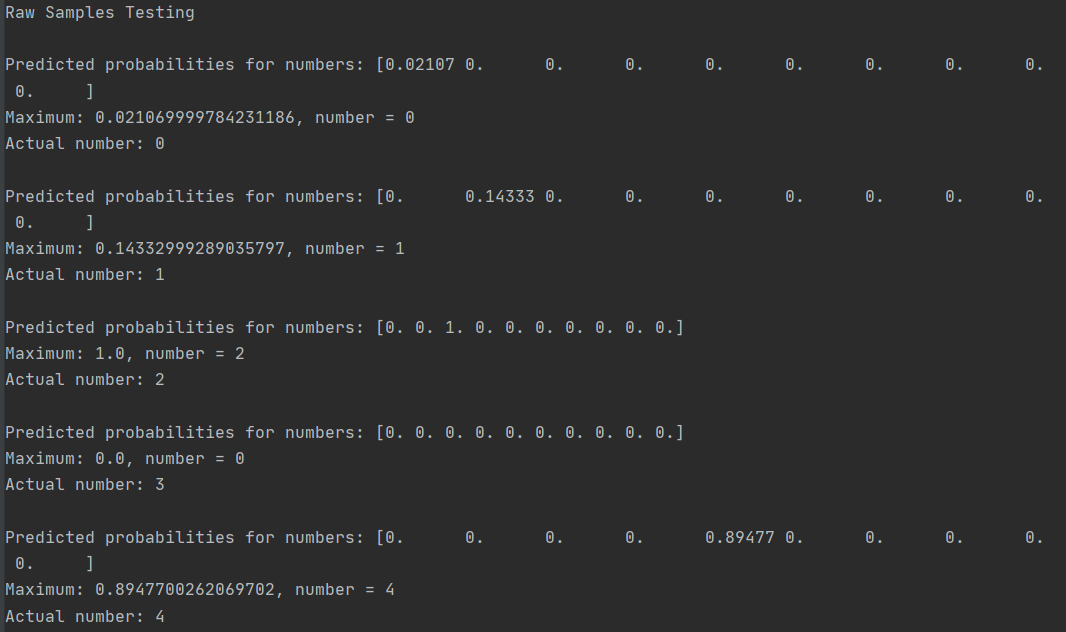


Рис. 14. Значения метрик на тестовой выборке



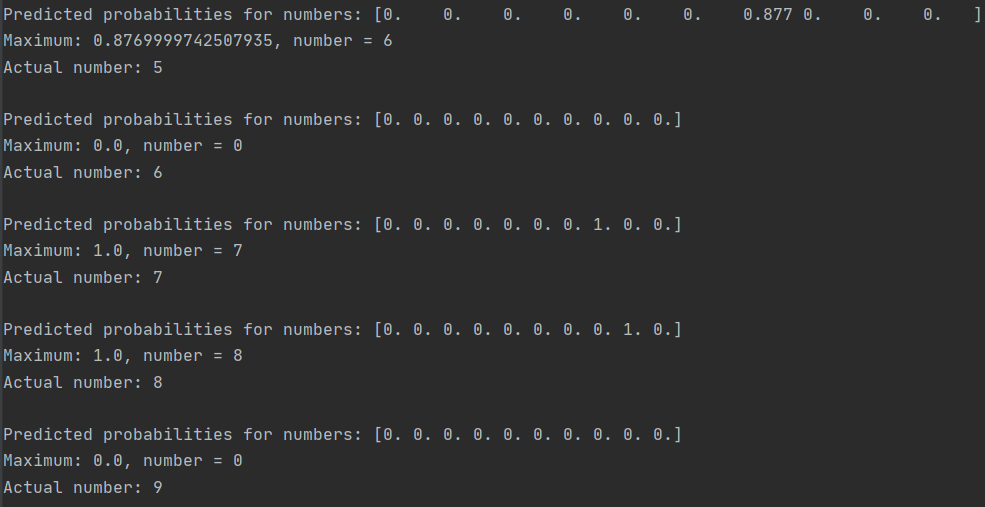


Рис. 15-16. Результаты работы модели на конкретных экземплярах тестовой выборки

Очевидно, что, вопреки ожиданиям, большинство характеристик сильно ухудшились. В то же время нам почти удалось добиться желаемого результата относительно величин в результирующем векторе – значит, регуляризация, скорее всего, сделана на этот раз удачно.

### Модель №4

«Скачущий» график обыкновенно говорит о том, что была установлена слишком высокая скорость обучения – из-за этого модель при обучении «проскакивает» нужные шаги. Поэтому в новой модели скорость обучения была снижена вдвое, а размер пакета, напротив – увеличен.

Изменённые относительно модели №3 параметры:

* Batch size: 30
* Learning rate: 0.035

Результаты работы модели:

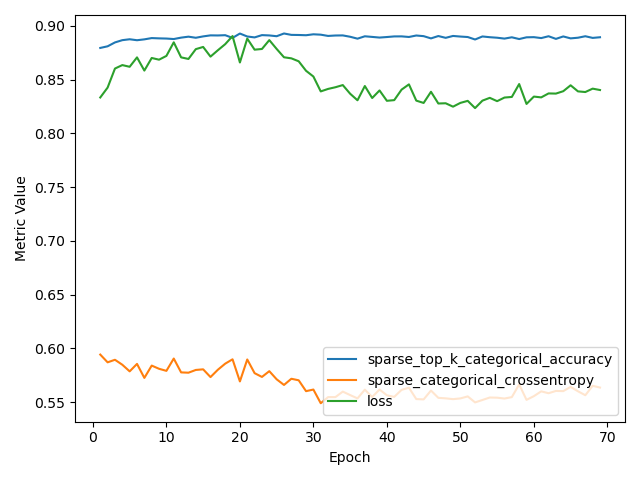


Рис. 17. Значения метрик по мере обучения модели

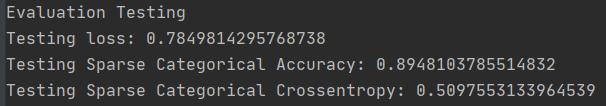
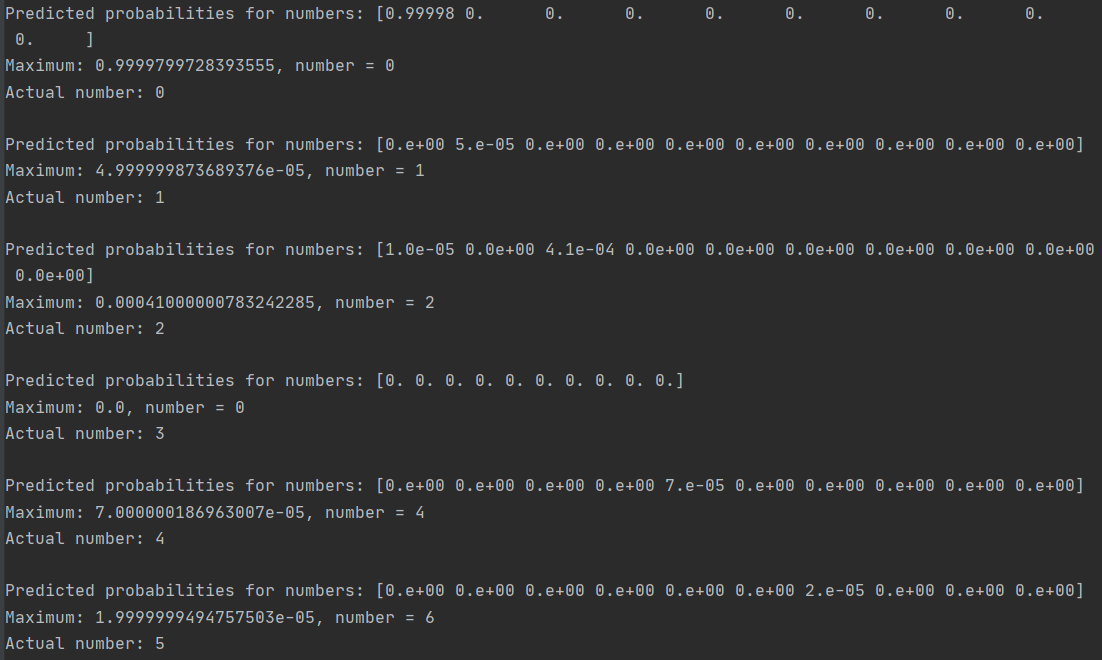


Рис. 18. Значения метрик на тестовой выборке



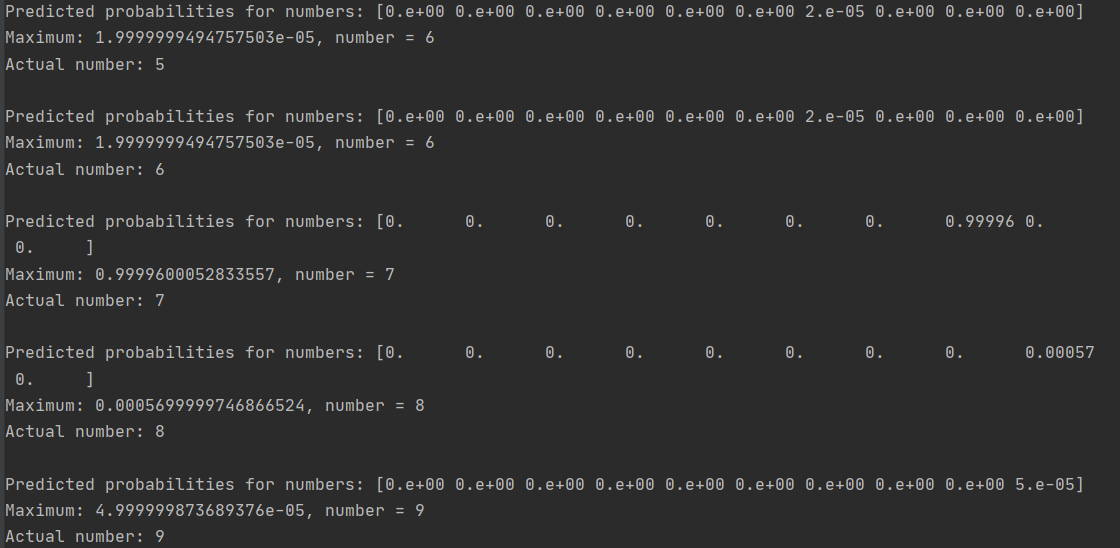


Рис. 19-20. Результаты работы модели с конкретными экземплярами тестовой выборки

### Модель №5

Предыдущие изменения едва ли поправили ситуацию. Попробуем ещё уменьшить скорость обучения и коэффициент ускорения, а также задать более осторожно начальный сдвиг для оптимизатора.

Изменённые относительно модели №4 параметры:

* Batch size: 40
* Learning rate: 0.03

Результаты работы модели:

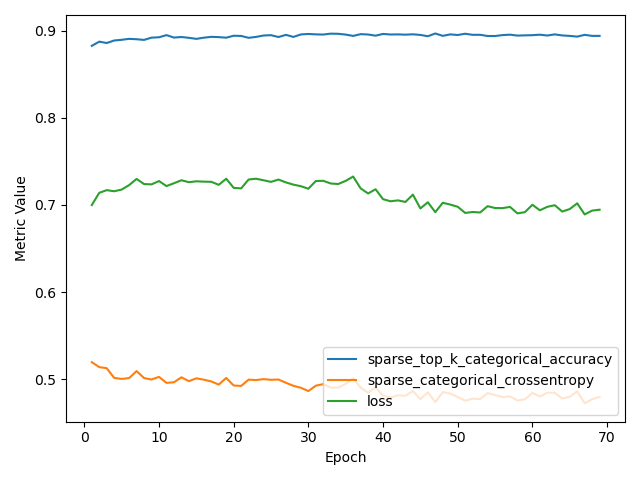


Рис. 21. Значения метрик по мере обучения

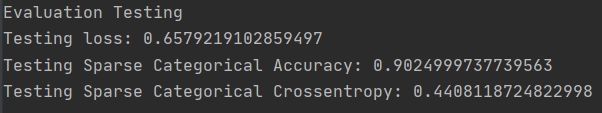
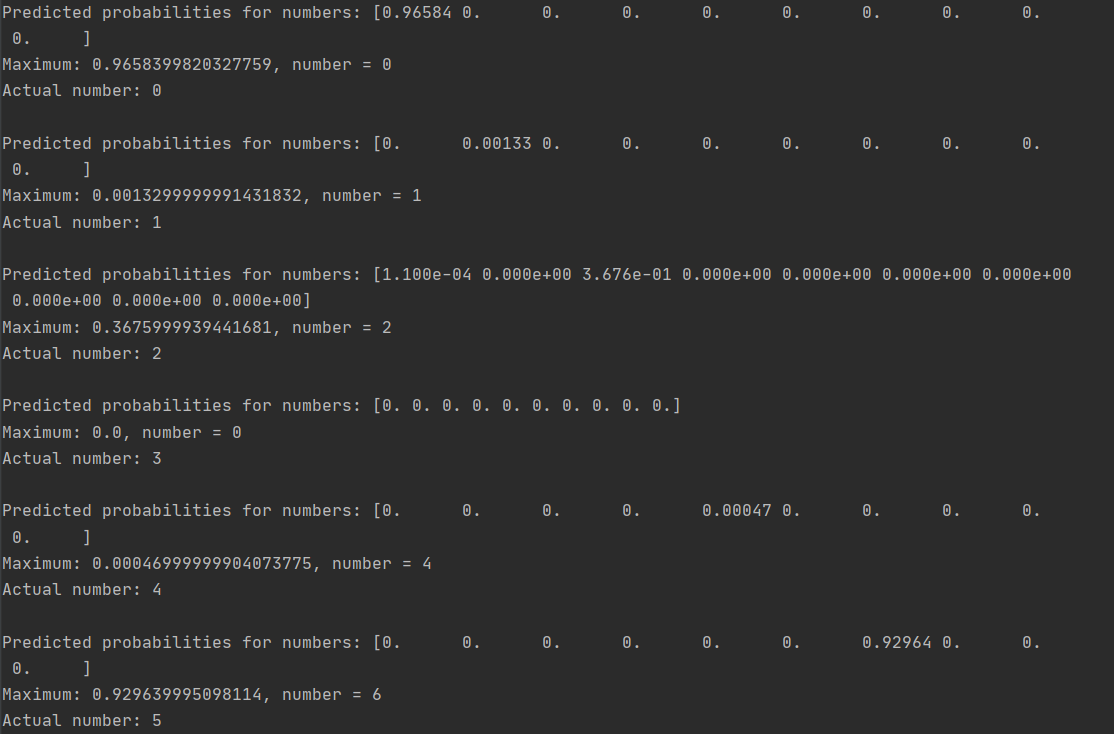


Рис. 22. Значения метрик на тестовой выборке



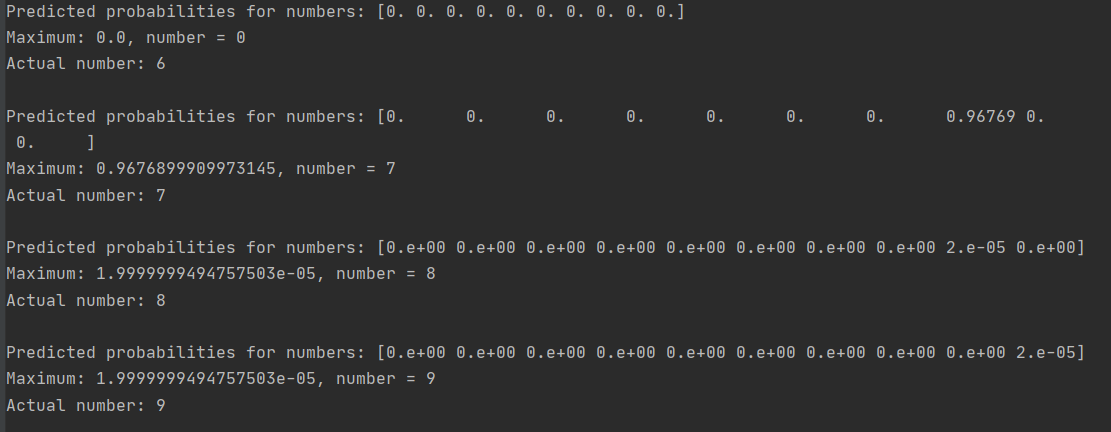


Рис. 23-24. Результат работы модели на конкретных экземплярах тестовой выборки

Графики немного выровнялись, но численные показатели всё ещё далеки от приемлемых.

### Модель №6

Снова поправим величины тех параметров, которые более всего влияют на точность расчётов.

Изменённые относительно модели №4 параметры:

* Batch size: 100
* Learning rate: 0.02

Результаты работы модели:

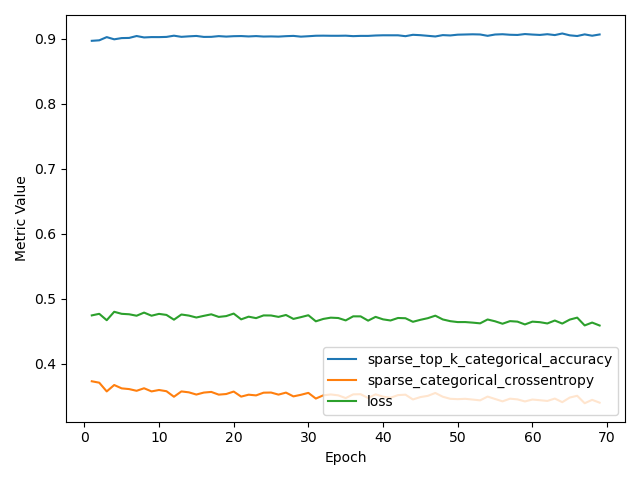


Рис. 25. Значения метрик по мере обучения модели

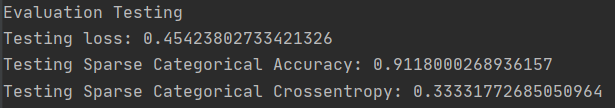
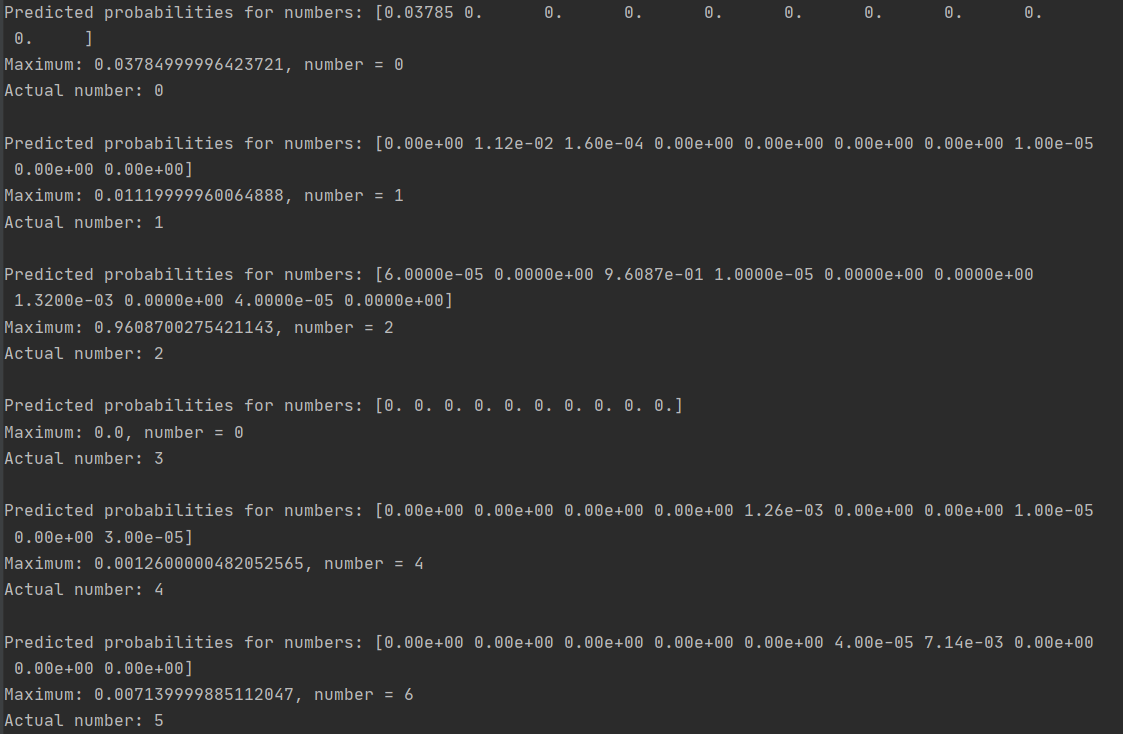


Рис. 26. Значения метрик на тестовой выборке



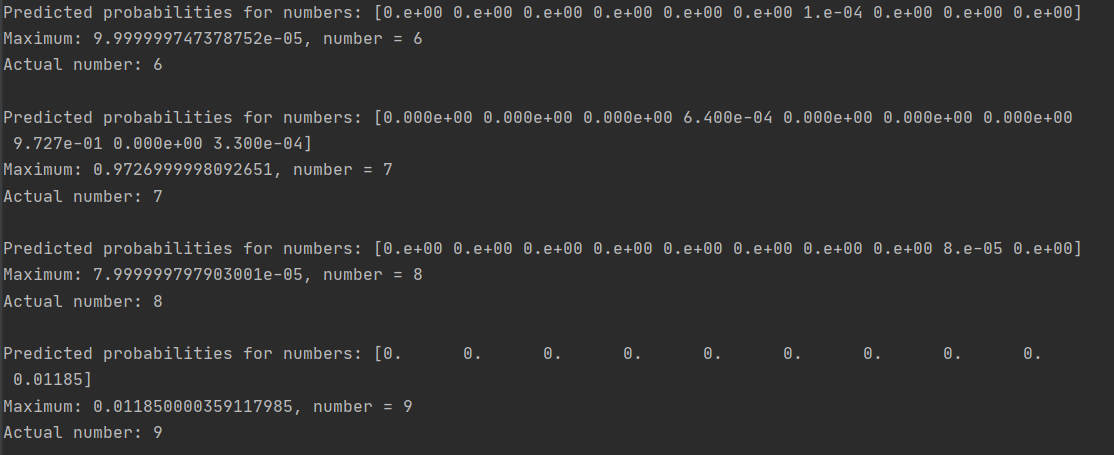


Рис. 27-28. Результаты работы модели на конкретных экземплярах тестовой выборки

С одной стороны, показатели почти пришли в норму, с другой – модель практически не обучается, с трудом только сохраняя изначальный уровень.

### Модель №7

При изучении первых трёх моделей было установлено, что задача в принципе допускает весьма высокую скорость обучения. Попробуем снова увеличить её и коэффициент её увеличения, применив иную функцию активации.

Изменённые относительно модели №6 параметры:

* Количество эпох: 50
* Функция активации: tanh(x) (вместо сигмоида)
* Learning rate: 0.07

Результаты работы модели:

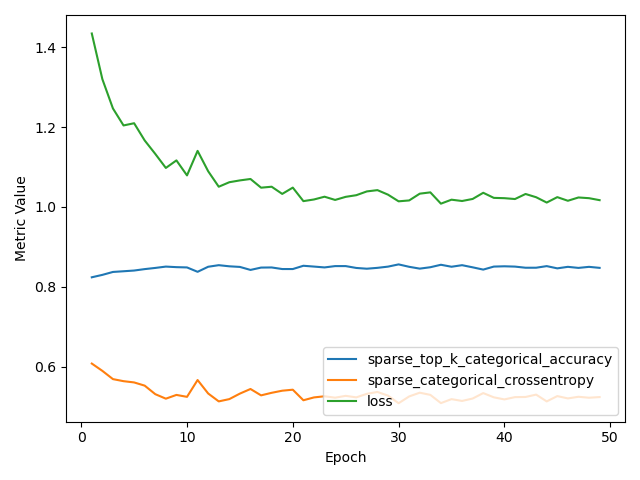


Рис. 29. Значения метрик по мере обучения модели

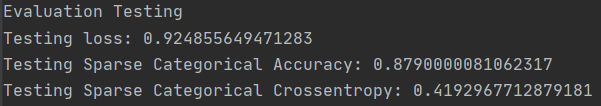
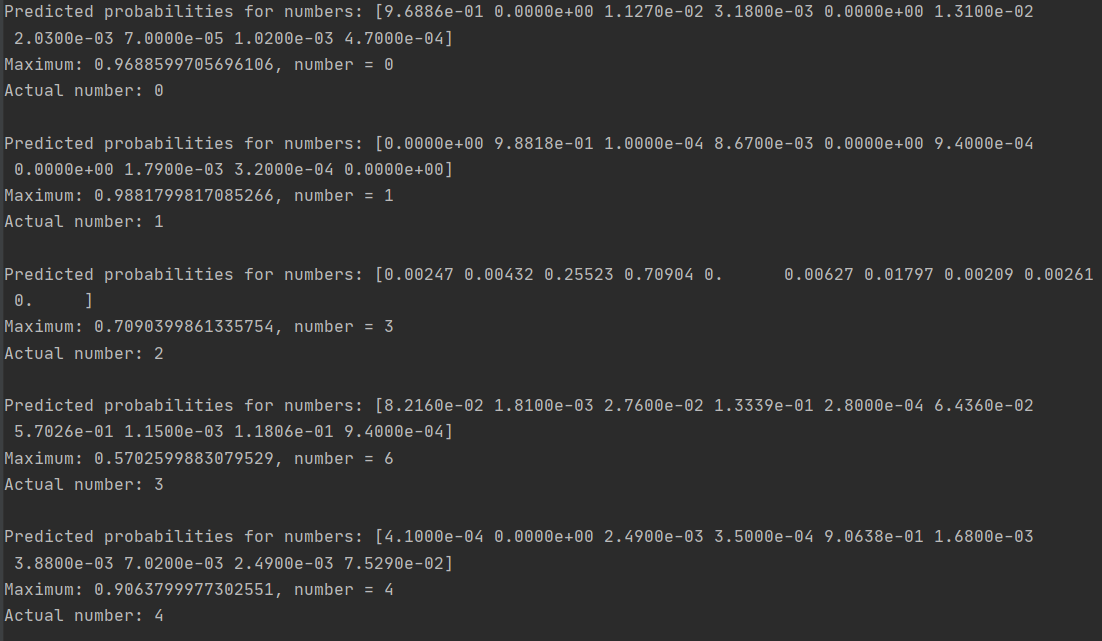


Рис. 30. Значения метрик на тестовой выборке



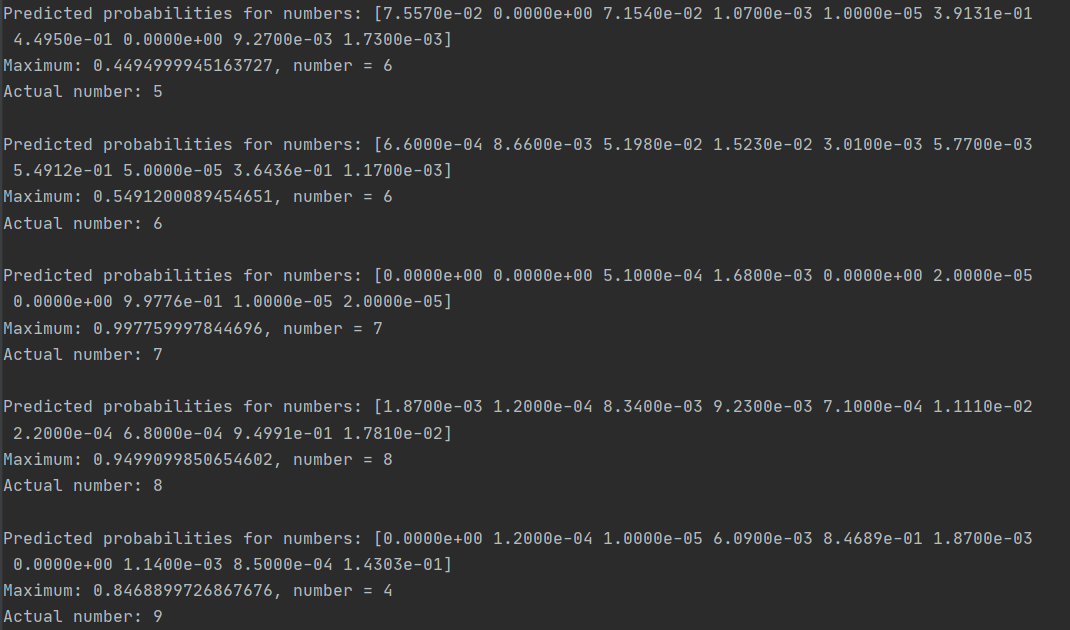


Рис. 31-32. Результаты работы модели на конкретных экземплярах тестовой выборки

### Модель №8

Изменение одной только функции активации толком не помогло. Попробуем также сменить алгоритм оптимизации – будем использовать ускорение с помощью накопления градиента вместо ускорения по Нестерову. Кроме того, попробуем искусственно увеличить размерность изучаемого пространства: поднимем количество нейронов в tanh-слое. При большом количестве нейронов удобно применять dropout регуляризацию – добавим для этого дополнительный слой. Чтобы на выходе получать необходимые 10 значений, добавим также слой с softmax-активацией, который гарантирует, что сумма результатов будет равна 1; таким образом, скорее всего, мы сможем окончательно разобраться с проблемой «некрасивых» результирующих величин.

Изменённые относительно модели №7 параметры:

* Batch size: 30
* Регуляризация:
  + В слое с tanh-активацией: l1 + l2, l1\_rate = 0.000001, l2\_rate = 0.0000000001
  + В дополнительном слое: Dropout, rate = 0.1
* Оптимизация: Adam (keras.optimizers.Adam()), learning rate = 0.05,
* Количество нейронов в слоях: 50 (tanh) – 50 (dropout) – 10 (softmax)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

Рис. 33-40. Ошибочно классифицированные экземпляры тестовой выборки

Снова обратим внимание на конкретные примеры ошибок в работе модели: выборочно отрисуем некоторые из неверно классифицированных изображений. Верхнее правое изображение было определено как семёрка – похоже, теперь модель даже слишком хорошо запомнило «перекладину»; следовательно, стоит уменьшить коэффициент, учитывающий редкость появления признака. Правое нижнее изображение, классифицированное как девятка, говорит о том, что нужно ещё поработать над регуляризацией: веса для пикселей, составляющих диагональ из правого верхнего угла в левый нижний, оказались слишком велики, так что всё, находящееся ниже, оказалось практически отброшено, заставляя модель признать изображение девяткой. Две четвёрки в первом ряду и девятка (второй ряд, второе изображение) написаны просто неаккуратно – такие ошибки можно опустить, вина за них лежит не на модели; остальные же экземпляры написаны вполне чётко, и это означает, что модель сама по себе просто ещё весьма неточна.

### Модель №9

В этой модели, кроме прочего, снова было увеличено количество нейронов. Вообще говоря, чтобы разрешить нелинейность таким путём, нужно, чтобы результирующая размерность была хотя бы на 1 больше исходной. Однако наличие нескольких нелинейных активаций не позволяет однозначно оценить соотношение между исходной и результирующей размерностью; поэтому, учитывая, что большое число нейронов сильно замедляет обучение, будем пока стараться ограничиваться относительно малыми приращениями.

Также попробуем немного сдвинуть валидационное дробление: благодаря этому у модели будет немного больше данных, доступных для обучения.

Изменённые относительно модели №7 параметры:

* Batch size: 100
* Количество эпох: 40
* Validation split: 0.15
* Регуляризация:
  + В слое с tanh-активацией: l1 + l2, l1\_rate = 0.000001, l2\_rate = 0.00000001
  + В дополнительном слое: Dropout, rate = 0.2
* Оптимизация: learning rate = 0.02,
* Количество нейронов в слоях: 60 (tanh) – 60 (dropout) – 10 (softmax)

Результаты работы модели:

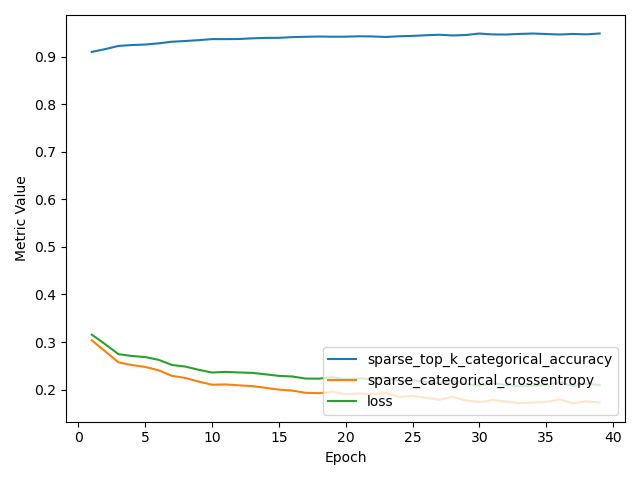


Рис. 41. Значения метрик по мере обучения модели

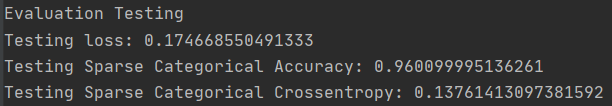


Рис. 42. Значения метрик на тестовой выборке

Показатели улучшились, и очень значительно: почти нет смещения для функции потерь, и точность возросла до 96%. Можно заключить, что направление изменений выбрано верно.

### Модель №10

После небольшой промежуточной проверки было принято решение вернуться к сигмоидной активации, также поменяв снова некоторые другие параметры согласно выявленным тенденциям.

Изменённые относительно модели №9 параметры:

* Batch size: 150
* Оптимизация:

Результаты работы модели:

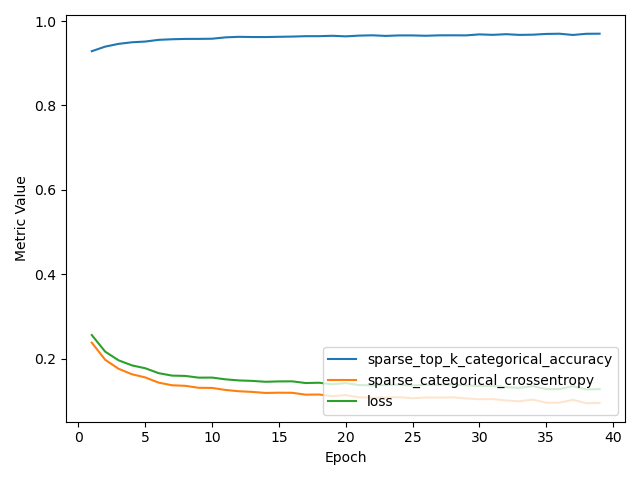


Рис. 43. Значения метрик по мере обучения модели

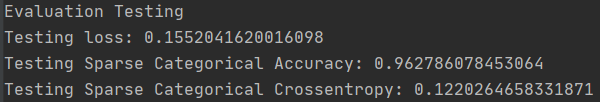


Рис. 44. Значения метрик на тестовой выборке

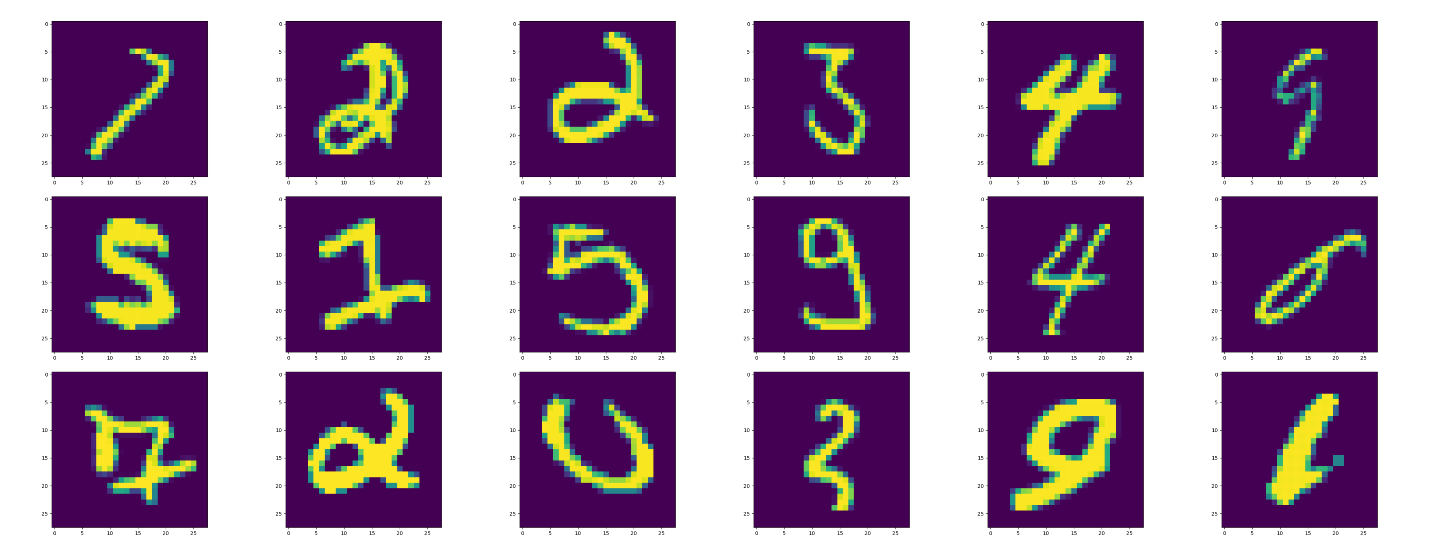


Рис. 45. Примеры ошибочной классификации

Выборочная отрисовка неверно классифицированных изображений позволила установить, что в таком виде модель ошибается почти исключительно на очень странных либо совсем неаккуратно написанных цифрах: слишком толстых и смазанных, покрывающих одной линией несколько групп пикселей сразу (7, 17, 19 изображения), сильно наклонённых вправо (5, 11, 12 изображения), изобилующих помарками (2, 16 изображения).

### Модель №11

Графики, которые мы наблюдали для прошлой модели, почти параллельны оси абсцисс; учитывая, что текущая скорость обучения была установлена как оптимальная, и менять её смысла нет, увеличивать время обучения для повышения точности тоже бессмысленно. Таким образом, можно пытаться повышать точность за счёт наращения размерности, а количество ошибок с нестандартным написанием снизить немного за счёт более жёсткой регуляризации (при синхронном увеличении числа нейронов это не должно ничего испортить).

Изменённые относительно модели №10параметры:

* Регуляризация:
  + l1+l2 внутри сигмоидного слоя: l1\_rate = 0.0001, l2\_rate = 0.0000001
  + Dropout в dropout-слое, rate = 0.25
* Количество нейронов в слоях: 80 (sigmoid) – 80 (dropout) – 10 (softmax)

Результаты работы модели:

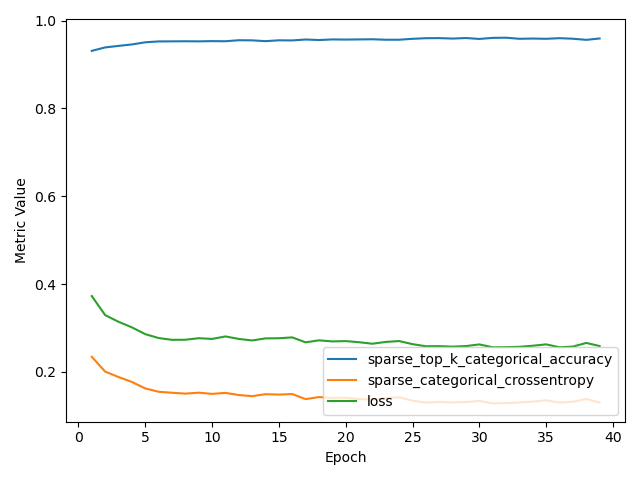


Рис. 46. Значения метрик по мере обучения модели

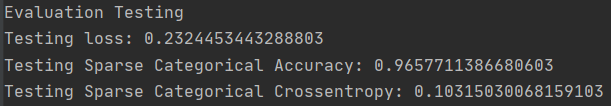


Рис. 47. Значения метрик на тестовой выборке

Точность немного повысилась, почти достигнув 97%, и несмещённая перекрёстная энтропия совсем мала, но функция потерь выросла – скорее всего, предыдущий вид модели был всё же лучше, хотя эта конфигурация позволила сделать вывод о вполне прямой зависимости точности от количества нейронов.

### Модель №12 (финальная)

Параметры:

* Batch size: 150
* Количество эпох: 30
* Validation split: 0.15
* Регуляризация:
  + Внутри сигмоидного слоя: l1 + l2, l1\_rate = 0.000001, l2\_rate = 0.000001
  + Dropout, rate = 0.27
* Оптимизация: стохастический градиентный спуск (tf.keras.optimizers.SGD()) без ускорения по Нестерову, learning rate = 0.01
* Оптимизация: Adam, learning rate = 0.02,
* Количество нейронов в слоях: 1000 (sigmoid) – 1000 (dropout) – 10 (softmax)

Результаты работы модели:

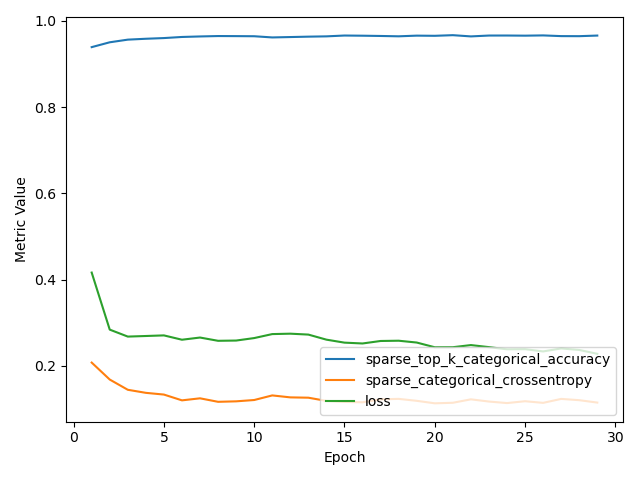


Рис. 48. Значения метрик по мере обучения модели

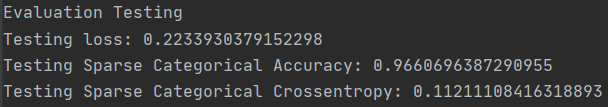


Рис. 49. Значения метрик на тестовой выборке

Точность немного повышена относительно модели №11; перекрёстная энтропия немного увеличилась, но зато относительное её смещение стало меньше.

## Консольное приложение

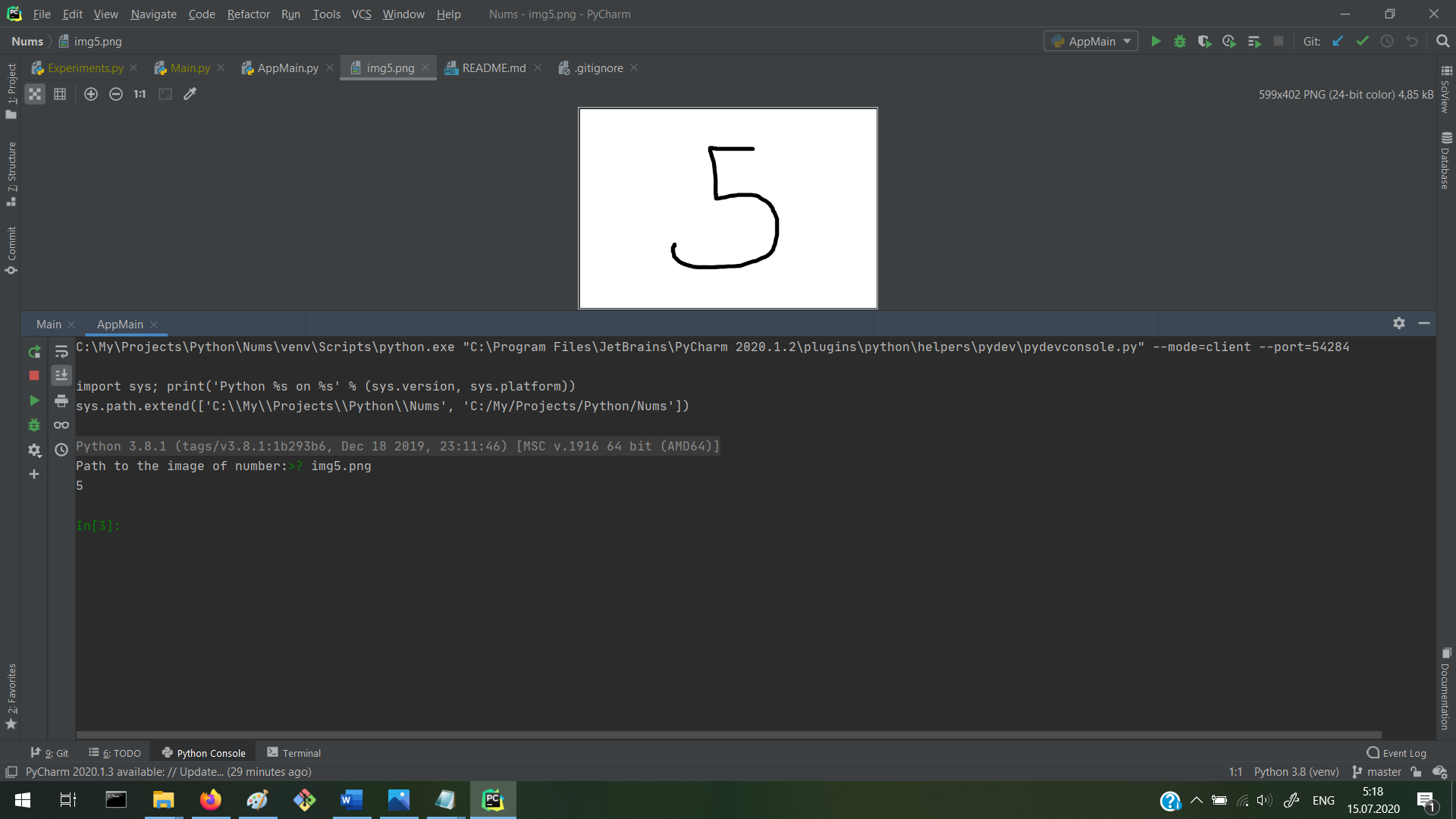
Обученная модель (архитектура и веса) сохраняется с помощью метода model.save(). Затем она восстанавливается в файле собственно приложения с применением keras.models.load\_model().

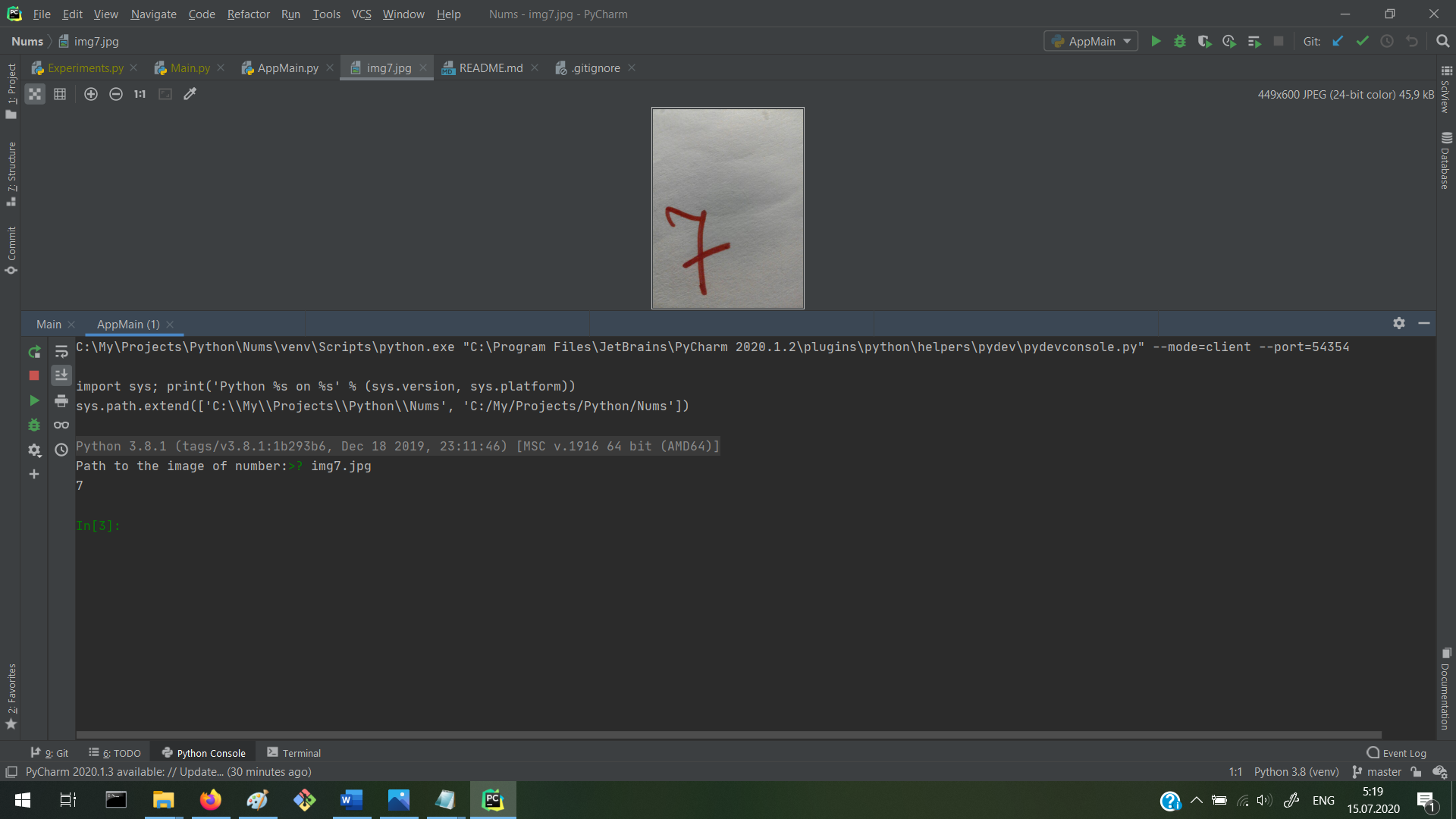
Пользователь вводит путь к изображению, цифру на котором хочет распознать, в командной строке. Изображение загружается с помощью метода библиотеки PIL Image.open(). После этого его необходимо привести к понятному для модели виду.

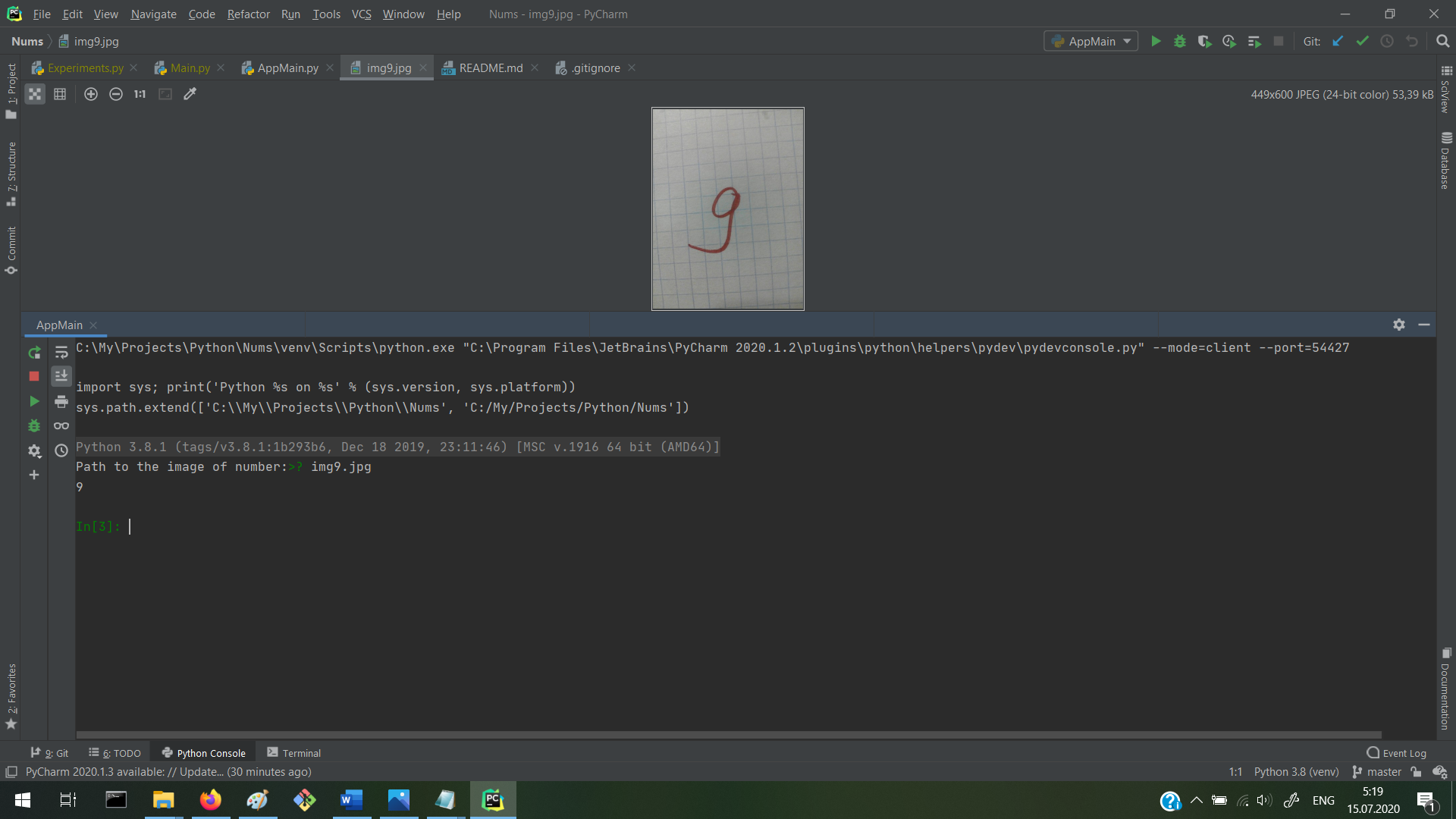
1. Изображение переводится в чёрно-белую цветовую гамму (RGB значения для каждого пикселя усредняются), полученные тона серого инвертируются и переводятся в матрицу чисел от 0 до 255, каждое из которых затем делится на 255
2. Слишком светлые части изображения принимаются белыми (пустыми), также отбрасываются пустые края изображения, чтобы цифра занимала бо́льшую его часть
3. Изображение центрируется относительно пиксельного центра масс (его координаты определяются с помощью метода библиотеки scipy ndimage.center\_of\_mass())
4. К изображению добавляются блоки пустых пикселей, чтобы сделать его квадратным, затем оно сжимается до размера 28х28 пикселей с помощью метода библиотеки PIL Image.resize()

Обработанное изображение передаётся на вход модели; применяется метод model.predict(). Результат работы выводится в консоль.

# Работа программы







# Использованные источники

* <https://keras.io> - официальная документация модуля keras
* <https://www.tensorflow.org> - официальная документация модуля tensorflow
* <https://habr.com/ru/post/318970> - справка по методам оптимизации
* <https://developers.google.com/machine-learning/crash-course> - изучение основ машинного обучения с применением Python

# Приложения

## Файл обучения модели Main.py

import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras import layers  
from matplotlib import pyplot  
import pandas  
import numpy  
  
  
# --- Data Preprocessing ---  
  
(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()  
  
x\_train = x\_train.reshape(len(x\_train), 784).astype('float32') / 255  
x\_test = x\_test.reshape(len(x\_test), 784).astype('float32') / 255  
  
# --- For Future Sample Analysis  
  
x\_train\_sorted = [[], [], [], [], [], [], [], [], [], []]  
for i in range(len(x\_train)):  
 label = y\_train[i]  
 x\_train\_sorted[label].append(list(x\_train[i]))  
  
means = [numpy.array(num).mean(axis=0) for num in x\_train\_sorted]  
  
# --- Hyperparameters ---  
  
batch\_size = 150  
epochs = 30  
learning\_rate = 0.02  
  
top\_k = 1  
metrics = [  
 keras.metrics.SparseTopKCategoricalAccuracy(  
 k=top\_k  
 ),  
 keras.metrics.SparseCategoricalCrossentropy()  
]  
  
validation\_split = 0.15  
l1\_regularization\_rate = 0.000001  
l2\_regularization\_rate = 0.000001  
  
b1 = 0.99  
b2 = 0.99  
eps = 1e-07  
  
units\_in\_layers = 1000  
dropout\_rate = 0.27  
  
# --- Model Building ---  
  
inputs = keras.Input(  
 shape=(784,),  
 batch\_size=batch\_size,  
 name='Input')  
  
second\_layer = layers.Dense(  
 units=units\_in\_layers,  
 activation='sigmoid',  
 kernel\_regularizer=keras.regularizers.l1\_l2(  
 l1=l1\_regularization\_rate,  
 l2=l2\_regularization\_rate  
 )  
)(inputs)  
  
dropout\_layer = layers.Dropout(  
 rate=dropout\_rate  
)(second\_layer)  
  
softmax\_layer = layers.Dense(  
 units=10,  
 activation='softmax'  
)(dropout\_layer)  
  
model = keras.Model(  
 inputs=inputs,  
 outputs=softmax\_layer,  
 name='MNIST\_model')  
  
model.compile(  
 optimizer=keras.optimizers.Adam(  
 learning\_rate=learning\_rate,  
 beta\_1=b1,  
 beta\_2=b2,  
 epsilon=eps  
 ),  
 loss=keras.losses.SparseCategoricalCrossentropy(),  
 metrics=metrics)  
  
# --- Model Training ---  
  
history = model.fit(  
 x\_train,  
 y\_train,  
 batch\_size=batch\_size,  
 epochs=epochs,  
 validation\_split=validation\_split)  
  
model.save(filepath='learnedModel')  
  
history\_for\_plotting = pandas.DataFrame(history.history)  
epochs\_for\_plotting = history.epoch  
  
# --- Training Process Plotting  
  
fig, ax = pyplot.subplots()  
  
ax.set\_xlabel('Epoch')  
ax.set\_ylabel('Metric Value')  
  
metrics\_for\_plotting = [metric.name for metric in metrics]  
metrics\_for\_plotting.append('loss')  
  
colours = [  
 'r',  
 'b',  
 'y',  
]  
c\_ind = 0  
  
for metric in metrics\_for\_plotting:  
 x\_arr = history\_for\_plotting[metric]  
 ax.plot(epochs\_for\_plotting[1:], x\_arr[1:], label=metric)  
 ax.legend(loc='lower right')  
  
fig.show()  
  
# --- Model Testing ---  
  
testing = model.evaluate(  
 x=x\_test,  
 y=y\_test,  
 batch\_size=batch\_size  
)  
print(  
 f'\n\nEvaluation Testing\n'  
 f'Testing loss: {testing[0]}\n'  
 f'Testing Sparse Categorical Accuracy: {testing[1]}\n'  
 f'Testing Sparse Categorical Crossentropy: {testing[2]}'  
)  
  
# --- Raw Sample Testing ---  
  
sample\_indices = [list(y\_test).index(num) for num in range(10)]  
samples = numpy.array([x\_test[ind] for ind in sample\_indices])  
predictions = model.predict(samples).round(5)  
n = 0  
print('\n\nRaw Samples Testing\n')  
for prediction in predictions:  
 max\_pred = max(prediction)  
 print(  
 f'Predicted probabilities for numbers: {prediction}\n'  
 f'Maximum: {max\_pred}, number = {list(prediction).index(max\_pred)}\n'  
 f'Actual number: {n}\n'  
 )  
 n += 1  
  
# --- Mean Comparing ---  
  
print('\nMean Comparing\n')  
  
predictions = model.predict(x\_test)  
for index in range(len(predictions)):  
 prediction = predictions[index]  
 max\_pred = max(prediction)  
 guess = list(prediction).index(max\_pred)  
 label = y\_test[index]  
 if label != guess:  
 mean = means[label]  
 sample = x\_test[index]  
 difference = sum([(sample[i] - mean[i]) \*\* 2 for i in range(784)]) / 784  
 print(f'Mistake with sample #{index}: Guess = {guess}, Actual = {label}, Difference with mean = {difference}')

## Файл консольного приложения AppMain.py

import numpy  
from scipy import ndimage  
from PIL import Image  
  
# Disabling tensorflow GPU register warnings  
import os  
os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'  
  
from tensorflow import keras  
  
  
# Loading the image the user wants to examine  
path = str(input('Path to the image of number:'))  
# Example images at img5.png, img7.jpg and img9.jpg  
image = Image.open(path)  
width = image.size[0]  
height = image.size[1]  
pix = image.load()  
  
left\_border = width - 1  
right\_border = 0  
top\_border = height - 1  
bottom\_border = 0  
  
step\_1 = [[] for \_ in range(height)]  
# If the inverted gray value of a pixel is lower, it's considered blank to raise the contrast of the image  
white\_gray\_threshold = 0.6  
  
# Replacing all the colors in the picture with shades of gray  
for i in range(width):  
 for j in range(height):  
  
 # Red, green and blue channel values for each pixel  
 r = pix[i, j][0]  
 g = pix[i, j][1]  
 b = pix[i, j][2]  
  
 gray = (r + g + b) // 3  
 inv\_gray = (255 - gray) / 255  
 threshold\_inv\_gray = 0 if inv\_gray < 0.6 else inv\_gray  
  
 if threshold\_inv\_gray != 0:  
 if i < left\_border:  
 left\_border = i  
 if i > right\_border:  
 right\_border = i  
 if j < top\_border:  
 top\_border = j  
 if j > bottom\_border:  
 bottom\_border = j  
  
 step\_1[j].append(threshold\_inv\_gray)  
  
# Discarding the parts of the image that almost certainly don't contain any useful information  
step\_2 = numpy.array([line[left\_border:right\_border + 1] for line in step\_1[top\_border:bottom\_border + 1]])  
width = right\_border - left\_border + 1  
height = bottom\_border - top\_border + 1  
  
'''  
The images in the set that the model was learned on were centered along their pixel center of mass, so the user's images  
should be pre-processed in the same way  
'''  
center\_of\_mass = ndimage.center\_of\_mass(step\_2)  
center\_x = int(center\_of\_mass[1])  
center\_y = int(center\_of\_mass[0])  
  
# These values define where the supporting blocks of pixels would be placed  
to\_left = width - center\_x > center\_x  
to\_top = height - center\_y > center\_y  
  
# Generating and appending the aligning blocks pf pixels  
h\_main = max(width - center\_x - 1, center\_x)  
h\_slave = width - h\_main - 1  
h\_addition = h\_main - h\_slave  
new\_width = width + h\_addition  
h\_addition\_block = numpy.zeros((height, h\_addition))  
stack\_order = (h\_addition\_block, step\_2) if to\_left else (step\_2, h\_addition\_block)  
step\_3 = numpy.hstack(stack\_order)  
  
v\_main = max(height - center\_y - 1, center\_y)  
v\_slave = height - v\_main - 1  
v\_addition = v\_main - v\_slave  
new\_height = height + v\_addition  
v\_addition\_block = numpy.zeros((v\_addition, new\_width))  
stack\_order = (v\_addition\_block, step\_3) if to\_top else (step\_3, v\_addition\_block)  
step\_4 = numpy.vstack(stack\_order)  
  
# Making the image square, same as the ones the model was trained on  
landscape = new\_width > new\_height  
bigger = max(new\_height, new\_width)  
smaller = min(new\_height, new\_width)  
  
whole\_eq\_addition = bigger - smaller  
  
eq\_addition\_1 = whole\_eq\_addition // 2  
eq\_block\_1\_shape = (eq\_addition\_1, new\_width) if landscape else (new\_height, eq\_addition\_1)  
eq\_block\_1 = numpy.zeros(eq\_block\_1\_shape)  
  
eq\_addition\_2 = whole\_eq\_addition - eq\_addition\_1  
eq\_block\_2\_shape = (eq\_addition\_2, new\_width) if landscape else (new\_height, eq\_addition\_2)  
eq\_block\_2 = numpy.zeros(eq\_block\_2\_shape)  
  
step\_5 = numpy.vstack((eq\_block\_1, step\_4, eq\_block\_2)) if landscape else numpy.hstack((eq\_block\_1, step\_4, eq\_block\_2))  
  
# Resizing the user's image to the size of the ones that the model was trained on  
training\_img\_size = 28  
img = Image.fromarray(step\_5 \* 255)  
r\_img = img.resize(size=(training\_img\_size, training\_img\_size))  
width = r\_img.size[0]  
height = r\_img.size[1]  
  
pix = r\_img.load()  
work\_with = [[] for \_ in range(28)]  
  
# Generating the final matrix to feed the model  
for i in range(width):  
 for j in range(height):  
 # Ensuring that no values have dropped below 0 due to calculation inaccuracy  
 work\_with[j].append(max(0, pix[i, j] / 255))  
  
# Loading the trained model  
model = keras.models.load\_model('learnedModel')  
# The original model worked with one-dimensional vectors, so the 28\*28 matrix should be reshaped the proper way  
prediction = model.predict(numpy.array(work\_with).reshape((1, training\_img\_size \*\* 2)))  
mp = max(prediction[0])  
print(list(prediction[0]).index(mp))