МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра дифференциальных уравнений и системного анализа

РАСПОЗНАВАНИЕ МАТЕМАТИЧЕСКИХ ФОРМУЛ ПО ИЗОБРАЖЕНИЮ С ИСПОЛЬЗОВАНИЕМ НЕЙРОННЫХ СЕТЕЙ

Курсовая работа

Кутищева Ильи Сергеевича

студента 2 курса,
специальность 1-31 03 09
Компьютерная математика
и системный анализ

Научный руководитель:
кандидат физ.-мат. наук,
доцент А. Э. Малевич

Минск, 2022

Белорусский государственный
университетМеханико-математический
факультет
Кафедра дифференциальных уравнений и системного анализа

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент второго курса Кутищев Илья Сергеевич

1 **Тема** Распознавание математических формул по изображению с использованием нейронных сетей

2 **Срок представления курсовой работы к защите** 16 мая 2022г.

3 Исходные данные для научного исследования

3.1 Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение. – М.: «ДМК Пресс», 2017.

3.2 Хайкин С. Нейронные сети. Полный курс – М.: Вильямс, 2006.

3.3 Тарик, Рашид Создаём нейронную сеть. — СПб.: ООО «Альфа-книга», 2017.

3.4 Wenqi Zhao, EtAl. Handwritten Mathematical Expression Recognition with Bidirectionally Trained Transformer. — arXiv 16.05.2021. — https://arxiv.org/abs/2105.02412

4 Содержание курсовой работы

4.1 Вывести все формулы математической модели решаемой задачи.

4.2 Описать архитектуру используемой нейросети и обосновать её выбор.

4.3 Описать работу выбранного алгоритма обучения нейросети.

4.4 Реализовать на компьютере описанный алгоритм.

4.5 Проверить работу алгоритма на тестовых примерах.

Руководитель курсовой работы 04 окт. 2021, А.Э. Малевич
(подпись, дата, инициалы, фамилия)

Задание принял к исполнению

(подпись, дата)

Примечание:
Допускается дополнять или исключать пункты в бланке задания.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

АКТУАЛЬНОСТЬ ИССЛЕДУЕМОЙ ТЕМЫ:

В академической сфере LaTeX широко используется для написания научных статей, содержащих порой длинные формулы с доказательствами. Люди тратят много времени, чтобы написать посимвольно формулы и математические выражения. Таким образом пришла идея реализации инструмента, с помощью которого можно будет распознавать математические формулы с картинки и перевести их сразу на язык LaTeX, чтобы значительно облегчить задачу для математиков, сохраняя их драгоценное время на что-то более важное, нежели поиск символов для формул.

Если рассматривать методы реализации, то без использования нейронных сетей обойтись невозможно. Нейронные сети на данный момент развиваются очень быстро, в связи с быстрым развитием вычислительных мощностей. Кроме того, одна из крупнейших компаний в мире Google предоставляет бесплатный доступ к части своих вычислительных мощностей посредствам среды Google Colaboratory. Для выполнения кода используются графические или тензорные процессоры, что также значительно ускоряет обучение нейронных сетей. Данная среда кроме преимуществ имеет и свои недостатки, которые мы в данной курсовой работе будем обходить.

ЗАДАЧИ И ЦЕЛИ ИССЛЕДОВАНИЯ:

1. Проанализировать существующие решения на данную тему

2. Найти подходящий набор данных

3. Стандартизировать данные.

4. Выбрать архитектуру нейронной сети и спроектировать её структуру на базе библиотеки keras tensorflow.

5. Обучить нейронную сеть полученным набором данных.

6. Стандартизировать выходные данные нейронной сети.

7. Проверить корректность работы нейронной сети

ГЛАВА 1
ОБЩИЕ ПОЛОЖЕНИЯ

1.1 Анализ существующих решений

На данный момент существует несколько альтернативных решений, которые могут частично решать задачу распознавания математических формул на изображениях с помощью нейронных сетей, каждая из которых имеет свои преимущества и недостатки.

PhotoMath – удобное приложение для решения несложных математических задач. Одной из функций приложения является распознавание как печатных, так и рукописных формул. [1]

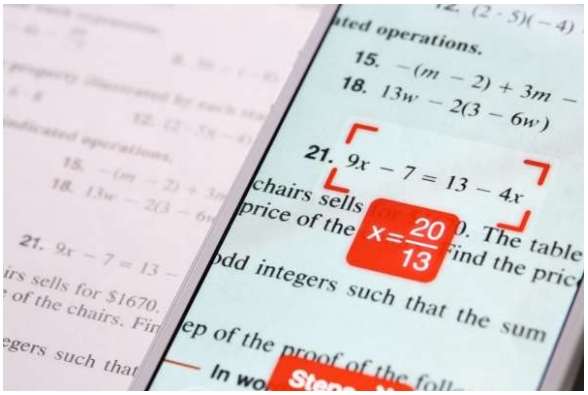


Рисунок 1.1 Photomath

Приложение работает на телефоне, но обработка изображений и вычисления происходит на сервере, поэтому требуется доступ к интернету

Dexify [2] – Онлайн приложение по распознаванию рукописных математических символов и переводу их на язык LaTeX. Плюсом является огромное количество распознаваемых символов и высокая точность. Минусом

является то, что приложение способно классифицировать лишь один символ, поданный на вход.

Рисунок 1.2 Dexify

Также разработчик оставил на странице приложения ссылку на базу данных, на которой проводилось обучение данной нейронной сети для желающих разработать и обучить свою нейронную сеть по распознаванию символов.

1.2 Офлайн и онлайн обработка

Существует некоторое разделение нейронных сетей, распознающих рукописные символы и слова. В данном случае есть разделение на «онлайн» и «офлайн» обработку символов в зависимости от того, что и каким образом подаётся на вход нейронной сети.

«Онлайн» обработка имеет преимущество по точности определения символов и слов. На вход кроме изображения подаётся направление и скорость отслеживаемого курсора, с помощью которого рисуются символы. Данным

подходом успешно пользуется компания Google в своём приложении GBoard, которое является аналогом базовой клавиатуры для ввода на Android.

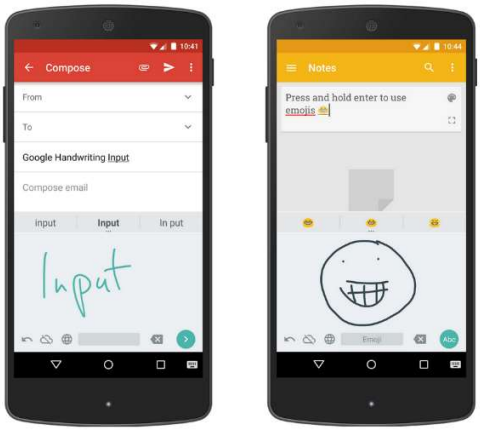


Рисунок 1.3 GBoard [3]

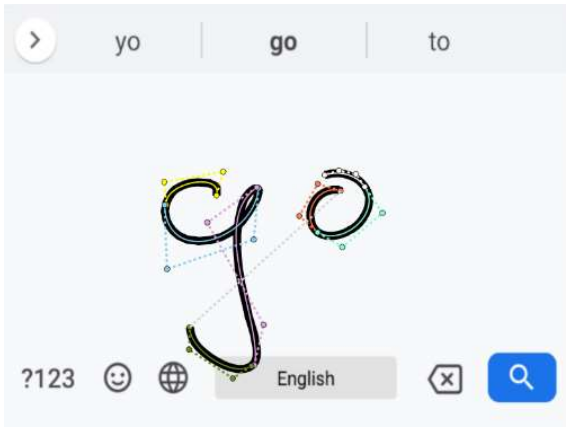


Рисунок 1.4 Пример ввода GBoard [4]

«Офлайн» обработка имеет преимущество при обработке документов, которые уже написаны и не имеют данных, кроме самого изображения текста, соответственно такой подход более универсален для применения, хоть и даёт менее точный результат. В данной курсовой работе я выбрал именно «офлайн» подход, так как с одной стороны он более универсален, а с другой стороны найти набор данных для обучения такой нейронной сети будет куда проще, чем в первом случае с «онлайн» обработкой.

1.3 Основные определения

LaTeX - это высококачественная система набора текста; он включает в себя функции, предназначенные для производства технической и научной документации. LaTeX является стандартом де-факто для обмена и публикации научных документов. [5]

Нейронные сети - это сеть или цепь биологических нейронов , или, в современном понимании, искусственная нейронная сеть , состоящая из искусственных нейронов или узлов. Таким образом, нейронная сеть — это либо биологическая нейронная сеть, состоящая из биологических нейронов, либо искусственная нейронная сеть, используемая для решения задач искусственного интеллекта. [6]

Jupyter notebook - это интерактивная веб-ориентированная вычислительная среда для создания документов блокнота. [7]

Google Colaboratory (также известная как Colab) — это бесплатная среда для Jupyter Notebook файлов, которая работает в облаке и хранит свои блокноты на Google Диске. [7]

В компьютерном зрении, *сегментация* — это процесс разделения цифрового изображения на несколько сегментов (множество пикселей, также называемых суперпикселями). Цель сегментации заключается в упрощении и/или изменении представления изображения, чтобы его было проще и легче анализировать. [8]

Свёрточная нейронная сеть (convolutional neural network, CNN) — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание образов. Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения. [9]

Операция свёртки:

$$B_{ij} = (X * K)_{ij} = \sum_{\alpha=0}^{k-1} \sum_{\beta=0}^{k-1} X_{i+\alpha, j+\beta} K_{\alpha\beta}, \quad 0 \leq i, j < x - k + 1$$

Рисунок 1.5 Операция свёртки (Источник [10])

Набор данных (с англ. Dataset) для машинного обучения – это обработанная и структурированная информация в табличном виде. Строки такой таблицы называются объектами, а столбцы – признаками. [11]

Tensorflow – открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. [12]

Keras - открытая библиотека, написанная на языке Python и обеспечивающая взаимодействие с искусственными нейронными сетями. Она представляет собой надстройку над фреймворком TensorFlow. Нацелена на оперативную работу с сетями глубинного обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. [13]

ГЛАВА 2
РЕАЛИЗАЦИЯ НЕЙРОСЕТИ

2.1 Поиск подходящего набора данных

Самое главное для обучения нейронной сети – это найти подходящий набор данных. Существует множество готовых наборов данных для обучения нейронных сетей близких к теме «Математические символы». Приведу несколько примеров.

Набор рукописных цифр «mnist», включающий в себя 60000 изображений для обучения нейронной сети и 10000 для её тестирования. Хороший набор данных для обучения нейронной сети распознаванию рукописных цифр, но в данном случае он нам вряд ли подойдёт, так как кроме цифр нам нужно распознавать математические выражения и знаки.

Набор рукописных цифр, рукописных строчных и заглавных английских букв «Emnist ByClass», включает в себя 697 932 изображений для обучения нейронной сети и 116 323 изображений для её тестирования. Такой набор данных уже больше подходит для цели распознавания математических формул, но всё еще в этом наборе, как и в прошлом, не хватает специальных математических символов вроде интеграла или суммы, знаков сложения, вычитания, деления, умножения и так далее. [14]

Тогда было принято решение зайти на портал Kaggle [15].

Kaggle — система организации конкурсов по исследованию данных, а также социальная сеть специалистов по обработке данных и машинному обучению. Принадлежит корпорации Google (с марта 2017 года). [16]

На данной публичной платформе пользователи и организации могут публиковать свои размеченные наборы данных, специализированные для обучения нейронных сетей.

По поисковому запросу «handwritten math symbols» было найдено 6 наборов данных, из них подходящих всего 4. Из этих 4 я выбрал оптимальный по размеру набор данных, состоящий из 375 974 картинок размером 45x45 пикселей.

В описании набора данных сказано:

«Он (набор данных) включает в себя основные символы греческого алфавита, такие как: альфа, бета, гамма, мю, сигма, фи и тета. Включены английские буквенно-цифровые символы. Все математические операторы, операторы множеств. Основные предопределенные математические функции, такие как: log, lim, cos, sin, tan. Математические символы, такие как: \int, \sum, \sqrt, \delta и другие». [17]

Обработка набора данных для нейронной сени с использованием библиотеки pyhon OpenCV

Перед началом хотелось бы отметить, что основные вычисления будут производиться на Google Colaboratory (бесплатный облачный сервис на основе Jupyter Notebook [18]). Подготовка данных будет частично связана с этим фактом.

2.1.1 Почему набор данных нельзя загрузить в Google Colaboratory в первоначальном виде

Google Colaboratory это отличный, а главное быстрый вычислительный облачный сервис. Большое число разработчиков (в том числе и я) выбирают его для работы. Данный сервис, как альтернатива jupyter notebook, обладает большой вычислительной мощностью, но в нем отсутствует нормальный файловый

менеджер. После множества безуспешных попыток загрузить набор данных как напрямую в среду Colaboratory, так и через Google Drive, было принято решение сделать первоначальную обработку данных на локальном компьютере, сделать подобие файла «Mnist», а уже потом для обучения нейронной сети загрузить данные в виде файла на сервис Google Drive, и уже из данного сервиса импортировать базу двумя файлами (растры изображений и их описание).

2.1.2 Считывание обучающего набора данных в переменную с помощью библиотеки OpenCV

Стоит заметить, что наш начальный обучающий набор состоит из чёрно-белых картинок 45x45, а для обучения нейронной сети нам нужно считать эти картинки и преобразовать в матрицы. Кроме того, нам будет достаточно матриц размера 28x28, для наших символов это не сыграет большой роли (большинство обучающих наборов данных, таких как Emnist, Mnist, как раз содержат картинки 28x28, что является стандартом для нейронных сетей, распознающих рукописные символы), но поможет нам позже в 2.56 раза сэкономить в оперативной памяти, что значительно ускорит процесс обучения.

Также стоит обратить внимание на структуру набора данных. В папке “extracted images” находится 82 папки, название каждой из которых является описанием для лежащих внутри них изображений.

```
x = []
y = []
count = 0
datadir = "D:\\KURSOVAYA\\data\\extracted_images\\"  # путь к папке с изображениями

max_members = 3000  # константа количества вхождений каждого символа
for folder in os.listdir(datadir):
    path = os.path.join(datadir, folder)
    print(folder)
    count = 0
    i=0
    len_dir = len(os.listdir(path))
    # цикл для выравнивания классов по количеству элементов
    while i < len_dir:
        images = os.listdir(path)[i]  # имена файлов в каждой папке
        img = cv2.imread(os.path.join(path, images),0)  # полный путь, imread - перевод в матрицу
        x.append(cv2.resize(img,(28,28)))  # downscale матрицы и добавление в массив
        y.append(folder)                   # добавление описания к данной матрице
        count+=1
        i+=1
        if count>max_members:
            break
        if i == len_dir and count<max_members:
            i=0
```

Рисунок 2.1 Код стандартизации данных

Таким способом в цикле, с помощью функций “os.listdir()” и “os.path.join” извлекая путь до файлов, мы проходимся по картинкам, считываем их в матрицы с помощью функции “cv2.imread”, меняем размер матрицы с помощью функции “cv2.resize”, добавляем в массив матриц “x” матрицы соответствующих изображений, а в массив “y” добавляем их описание.

2.1.3 Решение проблемы неравенства (неполноты) набора данных

В течении обучения нейронной сети огромной проблемой является неравномерное распределение данных.

```
g = sns.countplot(np.array(y_train,dtype="str"))
sns.set(rc={'figure.figsize':(40,13)})
#это оказалось большой проблемой в обучении, нужно хоть немного уравнять пропорции иначе сеть будет плохо обучаема
```

```
d:\changeface\miniconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
```

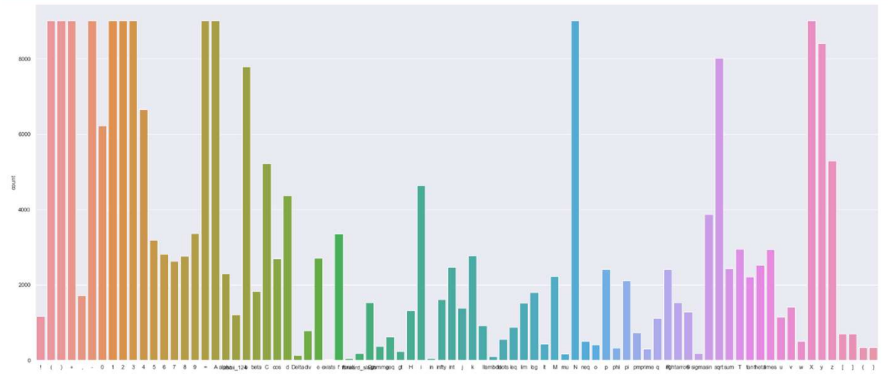


Рисунок 2.2 Гистограмма неравенства набора данных

На данной гистограмме показано количество различных символов с наборе данных. Легко заметить, что символы ()*123-ANX входят в наш изначальный набор в количестве более чем 10000, в то время как $\exists\sigma\mu\gamma$ имеют меньше сотни картинок в наборе. Такая большая разница в количестве, если её не устранить, может фатально сказаться на работе нашей нейронной сети. Так как картинки для обучения будут подаваться в случайном порядке из набора, то вероятность того что наша нейронная сеть будет получать картинки, которые встречаются часто, довольно велика и может сложиться ложное статистическое представление, что данные символы и в распознаваемом тексте будут встречаться чаще. Из-за этого число ошибок будет большим. Для устранения данной ошибки, чтобы в итоговой выборке изображения каждого символа попадались в одинаковой пропорции, я ограничил максимальное количество вхождений одного символа константой, а если количество изображений некоторого символа меньше данной константы, то изображения копируются по кругу. Таким образом, как мы можем увидеть, проблема была решена.

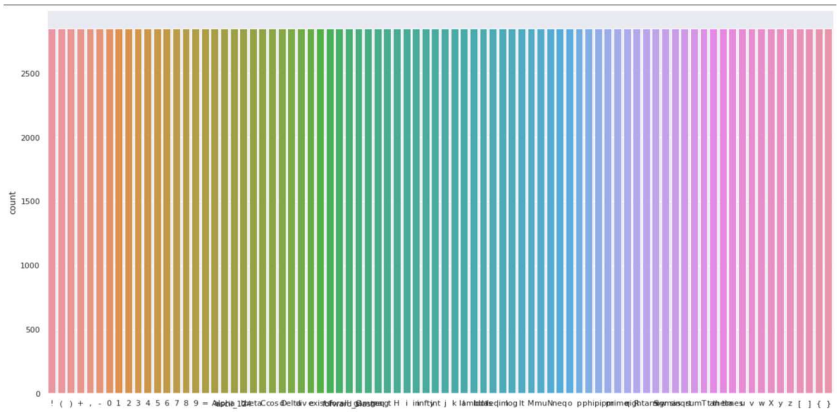


Рисунок 2.3 Гистограмма преобразованного набора данных

Большое количество дублирующихся изображений может тоже негативно сказаться на работе нейронной сети, но этот недостаток мы исправим уже на реализации нейронной сети при помощи встроенного генератора изображений.

2.1.4 Вывод данных в единый файл

```
# сохраним базу в файл чтобы передать в google colaboratory
a_file = open("y.txt", "w")
np.savetxt(a_file, y,fmt='%s')
a_file.close()

a_file = open("x.txt", "w")
np.savetxt(a_file, x,fmt='%s')
a_file.close()
```

Рисунок 2.4 Код сохранения в файл

С помощью функции “open()” открываем файл для записи, далее используя библиотеку “numpy” и функцию “savetxt” сохраняем значения переменных в текстовые файлы. Таким образом вместо 375 974 отдельных картинок мы получили два файла, которые уже куда легче импортировать как в оперативную память, так и на облачный сервис Google Colaboratory для последующей работы.

2.1.5 Разбиение общего набора данных на тренировочную и тестовую выборки, нормализация

Для правильного обучения нейронной сети и объективной оценки её работы необходимо делать разделение на «обучающий» набор данных и «проверочный». «Обучающий» набор данных используется нейронной сетью при корректировке весов, «проверочный» же используется для контроля точности, исключая возможность «заучивания» ответов нейронной сетью.

```
# делим на тестовый и тренировочный наборы 1 к 20
x_test, y_test = x[::20], y[::20]

x_train= np.array([item for index, item in enumerate(x) if (index) % 20 != 0])
y_train= np.array([item for index, item in enumerate(y.reshape(len(y))) if (index) % 20 != 0])
print(x_train.shape, y_train.shape)

# нормализация
x_train = np.array(x_train)/255
x_test = np.array(x_test)/255
```

Рисунок 2.5 Код разбиения набора данных

Таким образом отношение «обучающего» и «проверочного» набора будут связаны отношением 1:20. В нашем случае для тестовой выборки будет достаточно 11000 изображений.

В машинном обучении *нормализацией* называют метод предобработки числовых признаков в обучающих наборах данных с целью приведения их к некоторой общей шкале без потери информации о различии диапазонов. [19]

Нормализация значений нужна для корректной работы нейронной сети, поэтому данные преобразуются из диапазона [0,255] в диапазон [0,1] просто делением поэлементно на 255.

2.1.6 Создание соответствия классов

При обучении нейронной сети используются единичные вектора классов, поэтому создадим список из 82 номеров, каждому из которых сопоставим класс.

```
# составляем список соответствий значаний и номеров для нейросети
from operator import itemgetter
enum_labels = [[index,item] for index, item in enumerate(sorted(list(set(y))))]

def label_to_num(label,enum_labels):
    for i in enum_labels:
        if i[1] == label:
            return i[0]
```

Рисунок 2.6 Код создания соответствия классов

```
print(enum_labels) # перевод из подписей в нумерацию для нейросети
```

```
[[0, '!'], [1, '('], [2, ')'], [3, '+'], [4, ','], [5, '-'], [6, '0'], [7, '1'], [8, '2'], [9, '3'], [10, '4'], [11, '5'], [12,
'6'], [13, '7'], [14, '8'], [15, '9'], [16, '='], [17, 'A'], [18, 'C'], [19, 'Delta'], [20, 'G'], [21, 'H'], [22, 'M'], [23,
'N'], [24, 'R'], [25, 'S'], [26, 'T'], [27, 'X'], [28, '['], [29, ']'], [30, 'alpha'], [31, 'ascii_124'], [32, 'b'], [33, 'bet
a'], [34, 'cos'], [35, 'd'], [36, 'div'], [37, 'e'], [38, 'exists'], [39, 'f'], [40, 'forall'], [41, 'forward_slash'], [42, 'ga
mma'], [43, 'geq'], [44, 'gt'], [45, 'i'], [46, 'in'], [47, 'infty'], [48, 'int'], [49, 'j'], [50, 'k'], [51, 'l'], [52, 'lambd
a'], [53, 'ldots'], [54, 'leq'], [55, 'lim'], [56, 'log'], [57, 'lt'], [58, 'mu'], [59, 'neq'], [60, 'o'], [61, 'p'], [62, 'ph
i'], [63, 'pi'], [64, 'pm'], [65, 'prime'], [66, 'q'], [67, 'rightarrow'], [68, 'sigma'], [69, 'sin'], [70, 'sqrt'], [71, 'su
m'], [72, 'tan'], [73, 'theta'], [74, 'times'], [75, 'u'], [76, 'v'], [77, 'w'], [78, 'y'], [79, 'z'], [80, '{'], [81, '}']]
```

Рисунок 2.7 Пример работы функции

Далее в цикле преобразуем наши описания изображений в номера классов.

```
y_train = [label_to_num(label,enum_labels) for label in y_train]
y_test = [label_to_num(label,enum_labels) for label in y_test]
```

Рисунок 2.8 Код преобразования описаний

А после этого при помощи функции to_categorical библиотеки keras преобразуем каждое число в 82-мерный единичный вектор.

```
y_train = to_categorical(y_train, num_classes = len(list(set(y))))
y_test = to_categorical(y_test, num_classes = len(list(set(y))))
```

Рисунок 2.9 Код преобразования в вектор

На выходе получатся массивы [0,…,1,…,0] с 1 на n-ом месте, которые будут соответствовать числу n.

Данная операция необходима так как нейронная сеть будет подавать на выход похожие данные, а именно единичный 82-мерный вектор, который в итоге можно будет сравнить с правильным ответом для вычисления ошибки.

2.2 Решение проблемы сегментации составных рукописных формул на изображениях

Нейронная сеть, которую мы будем обучать, должна получать на вход символы, а точнее их растры 28x28. Мы же хотим подавать на вход программе цельную строку с формулой. В таком случае необходимо сегментировать поступающее на вход изображение.

```
# Извлекаем символы с изображения и переводим в матрицу
def letters_extract(image_file: str, out_size=28):
    img = cv2.imread(image_file)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    img_erode = cv2.erode(thresh, np.ones((3, 3), np.uint8), iterations=2)#1it

    # Выделяем контуры
    contours, hierarchy = cv2.findContours(img_erode, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    # plt.imshow(img)
    output = img.copy()
```

Рисунок 2.10 Код функции сегментации

Первоначально поступившее на вход изображение считываем при помощи функции “cv2.imread” преобразуем в вид многомерного массива. По умолчанию считаем, что на вход подаётся цветное изображение, тогда нужно его преобразовать в чёрно-белое. Для этого используем функцию “cv2.cvtColor” для преобразования цветового пространства, в аргументах указываем вид преобразования пространства. Так как в нашем случае из цветной картинки надо получить оттенки серого,

указывается аргумент “cv2.COLOR_BRG2GRAY”. Далее идёт работа с шумами и выделение порогового изображения для последующего определения контура.

На нашей картинке могут быть как нежелательные шумы, так и оттенки серого, которые нужно убрать, чтобы точнее выделить слова, символы и их границы на картинке. Для выделения порогового изображения используем функцию “cv2.threshold”. [20]

Вот как математически работает функция для отсечки шума:

- **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Рисунок 2.11 Описание функции обработки порога [21]

В документации можно подробно прочитать про параметры функции подавления шумов и обработки порога изображения. В нашем случае самым оптимальным мне показался метод “thresh_binary+thresh_otsu” без размытия Гаусса, пока на вход будут подаваться достаточно чистые исходные изображения. В дальнейшем же, при использовании программы на обычных фотографиях с плохой контрастностью, освещённостью, шумами, можно будет добавить размытие по Гауссу, чтобы уменьшить влияние шумов на результат и увеличить качество выделения порога изображения.

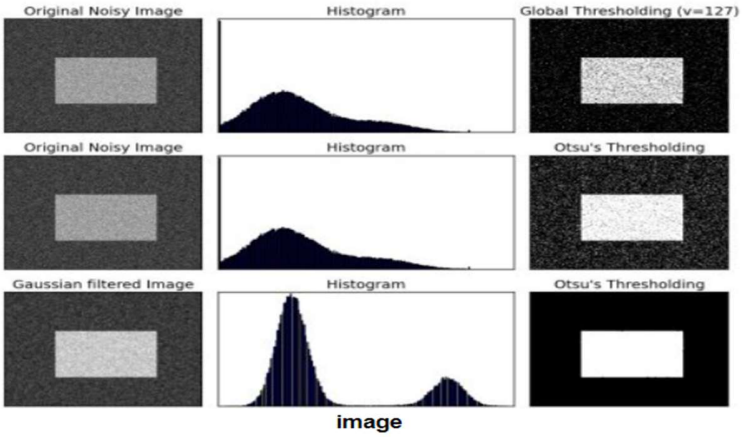


Рисунок 2.12 Пример работы функции с разными параметрами

Следующим шагом после выделения порога нам нужно немного размыть границы объектов изображения. Для этого используется функция “cv2.erode”. Вот как это размытие описывается в документации:

“Основная идея эрозии (cv2.erode) аналогична только эрозии почвы, она размывает границы объекта переднего плана. Так что же она делает? Ядро скользит по изображению (как в 2D-свертке). Пиксель в исходном изображении (либо 1, либо 0) будет считаться 1, только если все пиксели под ядром равны 1, в противном случае он стирается (обнуляется).

Функция расширения (cv2.dilate) Это прямо противоположно эрозии. Здесь элемент пикселя равен «1», если хотя бы один пиксель в ядре равен «1». Таким образом, увеличивается белая область на изображении или увеличивается размер объекта переднего плана. Обычно в таких случаях, как удаление шума, за эрозией следует расширение. Потому что эрозия удаляет белые шумы, но также уменьшает наш объект. Так что расширяем. Поскольку шум ушел, они не вернутся, но площадь нашего объекта увеличится. Это также полезно при соединении сломанных частей объекта.” [22]

Рисунок 2.13 Original, Erosion, Dilation [22]

Как можно заметить функции erode и dilate полностью противоположны, таким образом если фон изображения белый, а не чёрный как в документации, то данные функции выполняют ровно противоположные действия, то есть в нашем случае функция erode размывает границы. Пример можно увидеть на картинке.

Рисунок 2.14 Original image

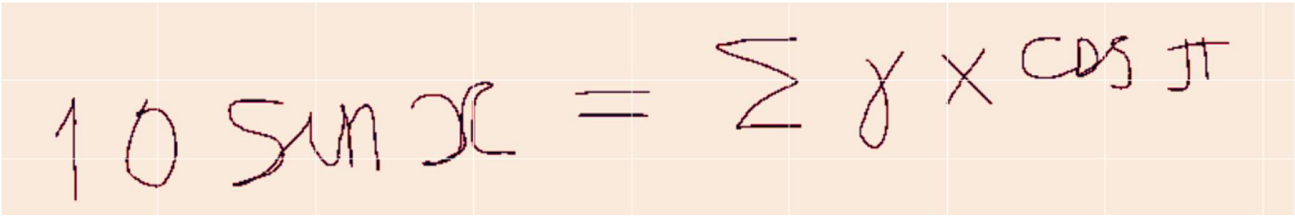


Рисунок 2.15 Thresholded image

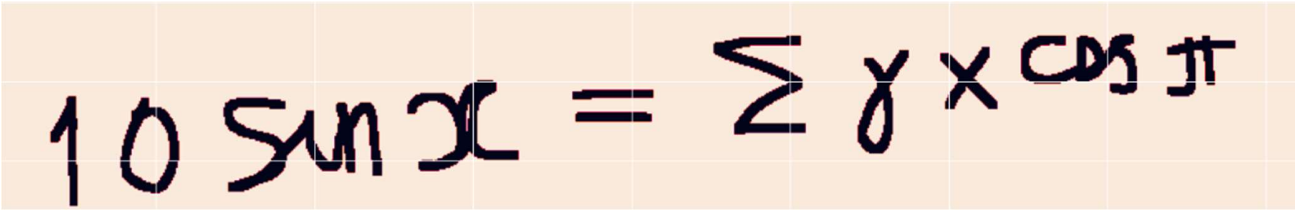


Рисунок 2.16 Eroded image

Нужно это для того чтобы соединить символы вроде “i j”, чтобы точки над ними в дальнейшем не определились как отдельные объекты.

Далее переходим к выделению контуров наших символов.

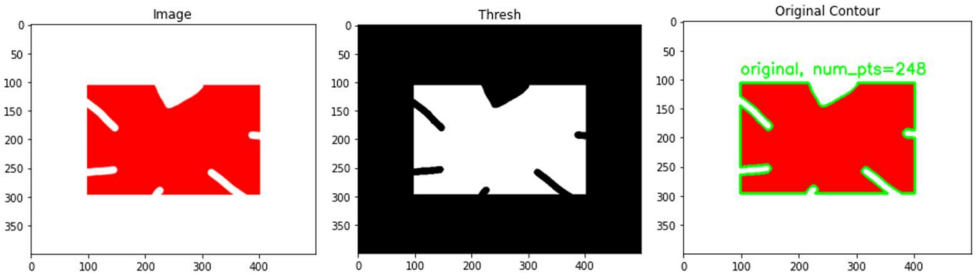


Рисунок 2.17 Выделение контуров [23]

Подобное выделение контуров осуществляется с помощью функции “cv2.findContours”, функция с данными параметрами возвращает множество контуров без аппроксимации.

И наконец в цикле мы выделяем в прямоугольники края наших контуров, а затем дополняем эти прямоугольники до квадратов. Таким образом в результате мы выделили каждый отдельный символ в квадрат, из которого мы извлекаем изображение, интерполируем его, если это необходимо, чтобы подогнать в рамки матрицы 28x28, для дальнейшей подачи на вход нейронной сети.

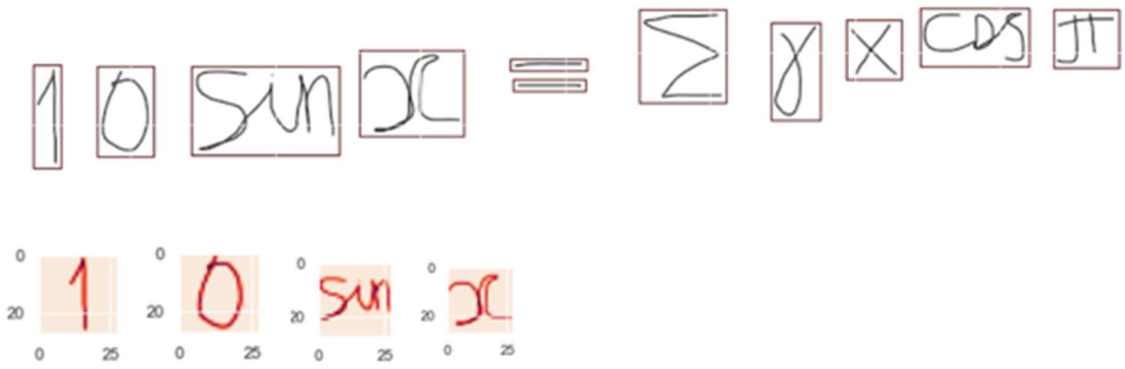


Рисунок 2.17 Результат работы функций сегментации

2.3 Сравнение полносвязной и свёрточной нейронных сетей

2.3.1 Полносвязные нейронные сети

В задаче классификации часто встречается использование полносвязной нейронной сети. Структура полносвязных нейронных сетей проста: нейрон с одной стороны получает множество значений, которые умножаются на соответствующие им веса, затем суммируются, а их сумма проходит через функцию активации, формируя входные значения для нейронов последующего слоя.

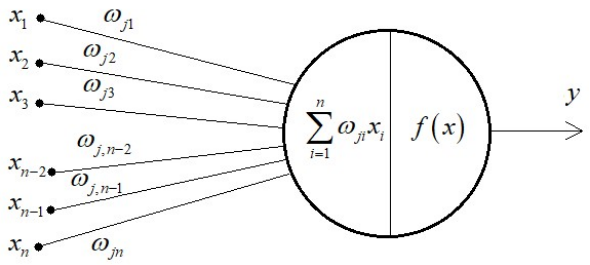


Рисунок 2.18 Визуализация нейрона [24]

Полносвязной нейронная сеть называется потому, что каждый нейрон слоя n связан с каждым нейроном слоя n+1.

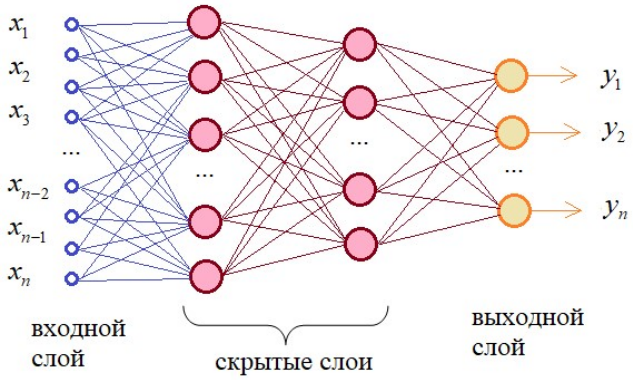


Рисунок 2.19 Полносвязная нейронная сеть [25]

Для классификации объектов количестве классов больше 2 в скрытых слоях в основном используется функция активации RelU=max(0,U).

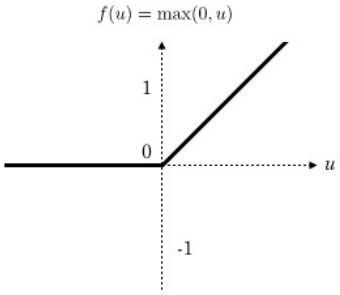


Рисунок 2.20 График функции RelU [26]

На последнем слое полносвязной нейронной сети используется функция SoftMax.

Softmax — это обобщение логистической функции для многомерного случая. Функция преобразует вектор z размерности K в вектор σ той же размерности, где каждая координата σ_i полученного вектора представлена вещественным числом в интервале [0,1] и сумма координат равна 1.

Координаты σ_i вычисляются следующим образом [27]:

$$\sigma_i\,(\mathrm{z}) = \frac{\mathrm{e}^{\mathrm{z_i}}}{\sum_{k=1}^{K} e^{z_k}}$$

Обучение полносвязной нейронной сети происходит методом обратного распространения ошибки (backpropagation).

Алгоритм: ***BackPropagation*** $(\eta, \alpha, \{x_i^d, t^d\}_{i=1,d=1}^{n,m}, \text{steps})$

1. Инициализировать $\{w_{ij}\}_{i,j}$ маленькими случайными значениями, $\{\Delta w_{ij}\}_{i,j} = 0$
2. Повторить NUMBER_OF_STEPS раз:

   .Для всех d от 1 до m:

   1. Подать $\{x_i^d\}$ на вход сети и подсчитать выходы o_i каждого узла.
   2. Для всех $k \in Outputs$

      $$\delta_k = -o_k(1 - o_k)(t_k - o_k).$$

   3. Для каждого уровня l, начиная с предпоследнего:

      Для каждого узла j уровня l вычислить

      $$\delta_j = o_j(1 - o_j) \sum_{k \in Children(j)} \delta_k w_{j,k}.$$

   4. Для каждого ребра сети {i, j}

      $$\Delta w_{i,j}(n) = \alpha \Delta w_{i,j}(n-1) + (1 - \alpha)\eta \delta_j o_i.$$
      $$w_{i,j}(n) = w_{i,j}(n-1) - \Delta w_{i,j}(n).$$

3. Выдать значения w_{ij}.

Рисунок 2.21 Описание алгоритма обратного распространения ошибки [28]

Где $0 < \eta < 1$ множитель задающий скорость движения, o_i – выход i-ого узла, t_k – правильный ответ k-ого узла, α - коэффициент инерциальности для сглаживания резких скачков при перемещении по поверхности целевой функции.

2.3.2 Свёрточные нейронные сети

Свёрточные нейронные сети – одна из самых эффективных архитектур нейронных сетей глубокого обучения.

Несмотря на все свои преимущества, полносвязная нейронная сеть справляется с задачей классификации изображений хуже, чем свёрточная, затрачивая при этом больше по времени вычисления и по памяти. Такой вывод можно сделать из множества практических исследований и статей. Пример статьи: [29].

Вывод статьи: «Хотя полносвязные сети не делают никаких предположений о входных данных, они, как правило, менее производительны и не подходят для извлечения признаков. Кроме того, у них есть большее количество весов, что приводит к увеличению времени обучения, в то время как свёрточные нейронные сети обучены идентифицировать и извлекать главные признаки из изображений для рассматриваемой проблемы с относительно меньшим количеством параметров для обучения.»

В связи с этим мною был сделан выбор использовать свёрточную нейронную сеть в комбинации с полносвязной для решения поставленной задачи.

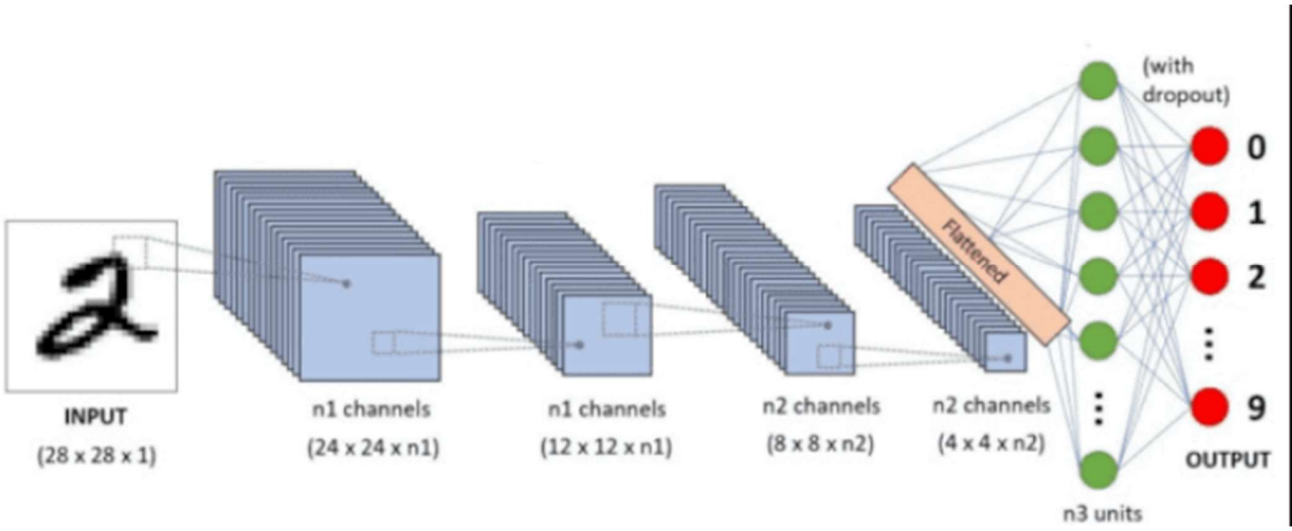


Рисунок 2.22 Структура свёрточной нейронной сети [30]

2.4 Сравнение архитектур свёрточных нейронных сетей

Свёрточные нейронные сети были придуманы уже довольно давно (1988г), поэтому в открытом доступе находится множество уже разработанных архитектур нейронных сетей по классификации, которые будут выдавать максимальный результат при минимальном времени. Мы рассмотрим несколько из них.

Самая простая из основных архитектур свёрточных нейронных сетей - это LeNet.

«LeNet — это структура сверточной нейронной сети, предложенная Yann LeCun et al. в 1989 году. В целом LeNet относится к LeNet-5 и представляет собой простую сверточную нейронную сеть, своего рода нейронная сеть с прямой связью, искусственные нейроны которой могут реагировать на часть окружающих ячеек в диапазоне покрытия и хорошо работать при крупномасштабной обработке изображений.» [31]

Более наглядно можно увидеть структуру этой сети на картинке.

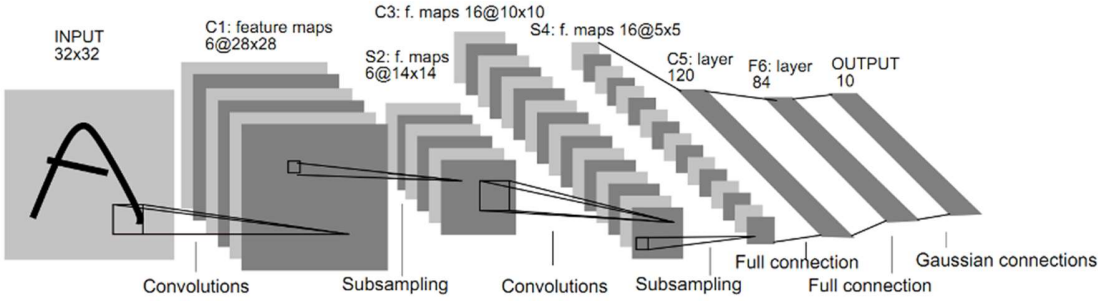


Рисунок 2.23 Структура LeNet [10]

С самого начала эта архитектура была разработана для распознавания символов Mnist, что очень близко к нашему набору данных. Ранее я уже работал с похожей архитектурой над распознаванием рукописных цифр, после обучения такая архитектура позволяет добиться точности 99.8% при правильном распределении в наборе данных. Так как нам не нужно распознавать структуры более сложные, чем символы, то я решил взять данную архитектуру за основу своей нейронной сети, при этом есть возможность отойти от стандарта и экспериментировать с количеством слоёв, размерами ядер свёрток, количеством слоёв и нейронов в полносвязной нейронной сети, чтобы в итоге добиться максимального результата.

Следующей по сложности является архитектура AlexNet. Используется для классификации более сложных цветных изображений 227x227 пикселей.

Рисунок 2.24 Примеры входных изображений AlexNet [32]

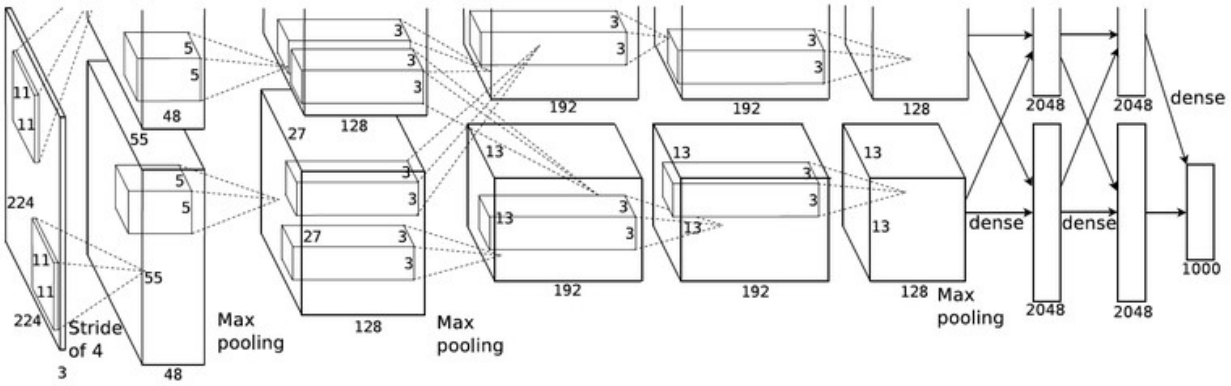


Рисунок 2.25 Структура AlexNet [33]

Данная архитектура была разработана таким образом, чтобы ускорить процедуру обучения нейронной сети, путём разделения каждого слоя на 2 потока, что при наличии двух внешних видеокарт позволяет значительно облегчить вычисления на каждой из них.

В этом случае архитектура уже куда более сложная и не подходит для решаемой задачи, так как такое значительное увеличение сложности структуры было спроектировано специально для задачи распознавания более сложных объектов, чем рукописные символы.

Далее после AlexNet (2012 г.) были разработаны такие архиектуры, как VGG (2013 г.), GoogLeNet (2014 г.), ResNet (2015 г.). С каждым поколением архитектуры решают всё более сложные задачи классификации, добиваясь всё большей точности. И тогда как LeNet содержала в сумме 7 слоёв, то ResNet содержала уже 152 с возможностью расширить более чем до 1000 слоёв.

2.5 Разработка архитектуры CNN на базе keras

Keras – очень хорошая библиотека, благодаря которой можно достаточно просто создавать модели нейронных сетей и тестировать их.

В моем случае имели место нескольких неудачных попыток обучить нейронную сеть. Изначально я попробовал такую структуру слоёв:

*input->[convolution->activate->pooling->]*2->flatten->fc_1->fc_2->fc_3->out*

Однако не помогало изменение параметров скорости обучения (Learning rate), количество нейронов в скрытых слоях полносвязной нейронной сети, размер ядер свёрточных слоёв. В любом случае даже когда обучение доходило до 80%, при подаче на вход изображения нарисованного от руки, точность нейронной сети не превышала 50%, особенно в случае сложных символов, состоящих из нескольких букв.

Несколько раз я менял структуру, самой стабильной оказалась структура с 3 свёрточными слоями и 3 полносвязными:

*input->[convolution->activate->pooling->]*3->flatten->fc_1->fc_2->fc_3->out*

Также в надежде увеличить точность изначально я обучал нейронную сеть на картинках в их сходном качестве 45x45, но в таком случае обучение происходило крайне медленно, занимая несколько часов на эпоху, при этом во время работы в

Google colaboratory уже через несколько эпох полностью заполнялись 12 гигабайт оперативной памяти и ядро автоматически прерывалось, не сохраняя результат.

Тогда было решено вернуться к базовому качеству 28x28, что значительно ускорило как обучение модели нейронной сети, так и стандартизацию данных.

Рассмотрим саму реализацию модели нейронной сети на python:

```
regularizer = l2(0.01)
model = Sequential()
model.add(Input(shape=input_shape))
model.add(Conv2D(32, (3, 3), strides=(1, 1), padding='same',
                 kernel_initializer=glorot_uniform(seed=0),
                 name='conv1', activity_regularizer=regularizer))
model.add(Activation(activation='relu', name='act1'))
model.add(MaxPool2D((2, 2), strides=(2, 2)))
model.add(Conv2D(32, (3, 3), strides=(1, 1), padding='same',
                 kernel_initializer=glorot_uniform(seed=0),
                 name='conv2', activity_regularizer=regularizer))
model.add(Activation(activation='relu', name='act2'))
model.add(MaxPool2D((2, 2), strides=(2, 2)))
model.add(Conv2D(64, (3, 3), strides=(1, 1), padding='same',
                 kernel_initializer=glorot_uniform(seed=0),
                 name='conv3', activity_regularizer=regularizer))
model.add(Activation(activation='relu', name='act3'))
model.add(MaxPool2D((2, 2), strides=(2, 2)))
model.add(Flatten())
```

Рисунок 2.26 Код структуры свёрточной части модели

Параметр regularizer будет отвечать за регуляризацию весов свёрточных слоёв, то есть добавлять штраф к размеру веса для функции потерь относительно суммы квадратов весов (L2).

Далее создаётся модель, добавляется входной слой. После чего реализуется наша структура. В первом свёрточном слое будет 32 ядра размера 3x3. Kernel_initializer позволяет задать начальное значение весов ядра. В данном случае используется glorot_uniform, задающий случайное значение в нормальном распределении, ограниченное с двух сторон постоянным пределом [-L,+L]. L

высчитывается по формуле $\sqrt{(6 / (fan\_in + fan\_out))}$, fan_in— количество входных единиц в тензоре весов и fan_out— количество выходных единиц. [34]

После свёрточных слоёв идет функция активации и пулинг ядром 2x2.

После трёх свёрточных слоёв выходной тензор преобразуется в одномерный массив, который уже далее поступает на вход полносвязной нейронной сети.

```
model.add(Dropout(0.5))
model.add(Dense(720, activation='relu', kernel_initializer=glorot_uniform(seed=0), name='fc1'))
model.add(Dense(504, activation='relu', kernel_initializer=glorot_uniform(seed=0), name='fc2'))
model.add(Dense(82, activation='softmax', kernel_initializer=glorot_uniform(seed=0), name='fc3'))
```

Рисунок 2.27 Код структуры полносвязной части модели

«Глубокие нейронные сети с большим количеством параметров — очень мощные системы машинного обучения. Однако в таких сетях серьезной проблемой является переобучение. Большие нейронные сети также медленны в использовании, что затрудняет работу с переоснащением путем объединения прогнозов множества разных крупных нейронных сетей во время тестирования. Dropout — это метод решения этой проблемы. Ключевая идея состоит в том, чтобы случайным образом отбрасывать нейроны (вместе с их соединениями) из нейронной сети во время обучения. Это предотвращает чрезмерное перенасыщение.» [35]

После дропаута идет три полносвязных слоя, последний из которых является выходным и имеет ровно 82 нейрона, как и количество классов в нашем наборе данных.

В конце в модель был добавлен оптимизатор.

```
optimizer = Adam()
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

Рисунок 2.28 Код сборки модели

Данный график может объяснить выбор данного оптимизатора, как более выгодную альтернативу методу среднеквадратичного распространения (RMSProp).

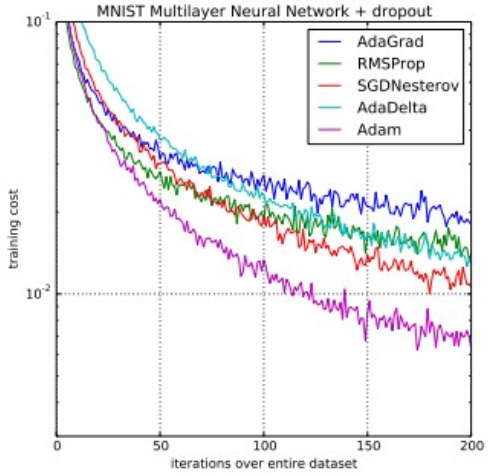


Рисунок 2.29 Сравнение Адама с другими алгоритмами оптимизации [36]

Последним шагом идёт компиляция модели.

Таким образом мы составили полную модель нейронной сети.

2.6 Обучение нейронной сети и расширение поданной обучающей базы

```
aug = ImageDataGenerator(zoom_range=0.1,
                         rotation_range=5,
                        width_shift_range=0.05,
                        height_shift_range=0.05)
```

Рисунок 2.30 Код создания генератора изображений

С помощью функции ImageDataGenerator расширяется набор данных, что положительно влияет на обучение модели нейронной сети и предотвращает переобучение. В качестве параметров функции на вход принимаются все возможные максимальные значения деформаций изображения, которые потом будут применяться случайным образом при обучении.

```
#тренировка модели

#model = keras.models.load_model('Math_CNN_1_structure.h5')

if len(x_train)>len(y_train):
  x_train = x_train[:-1]
hist = model.fit(aug.flow(x_train.reshape(len(x_train),28,28,1), y_train, batch_size=128,shuffle=True),
                 shuffle=True,
                 batch_size=128,
                 epochs=100,
                 validation_data=(x_test.reshape(len(x_test),28,28,1), y_test))

model.save('Math_CNN_2_structure.h5')
```

```
Epoch 1/100
 468/1827 [======>.....................] - ETA: 40s - loss: 3.0327 - accuracy: 0.4053
```

Рисунок 2.30 Код тренировки модели

Далее созданная модель подаётся в функцию fit, которая начинает её тренировку и проверку. Размер пакета при тренировке также играет немалую роль. Объем оперативной памяти виртуальной машины позволяет поднять batch_size до 3000, но на начальном этапе, когда ошибка еще достаточно большая, это лишь замедлит скорость обучения. Стоит увеличивать размер пакета в зависимости от величины ошибки, чем меньше ошибка, тем больше нужна точность, соответственно и размер пакета должен быть больше.

Также стоит параметр shuffle=True, так как изначально при стандартизации данных мы перебирали в цикле папки с символами, соответственно символы в итоговом наборе данных идут группами, поэтому для корректного обучения нейронной сети нужно брать случайную выборку из набора.

Таким образом модель показала хорошую динамику обучения за 100 эпох (2 часа) достигнув 97% точности на тестовой и 94,5% на обучающей выборке.

```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

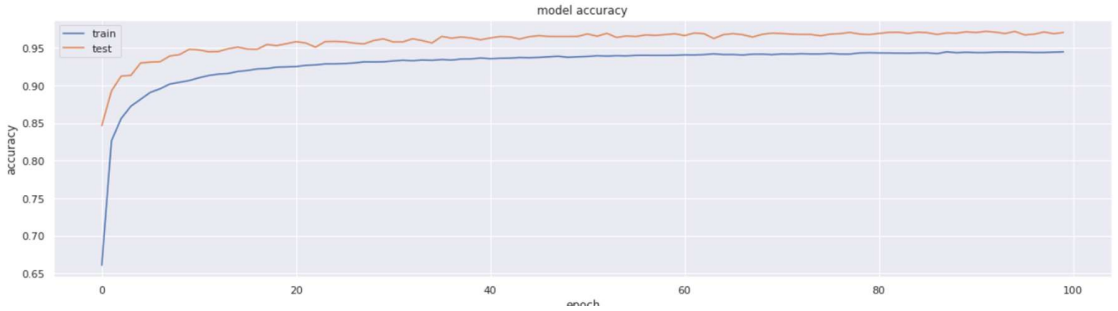


Рисунок 2.31 Код вывода и график изменения точности

После чего размер пакета был увеличен до 1024 и тренировка продолжилась.

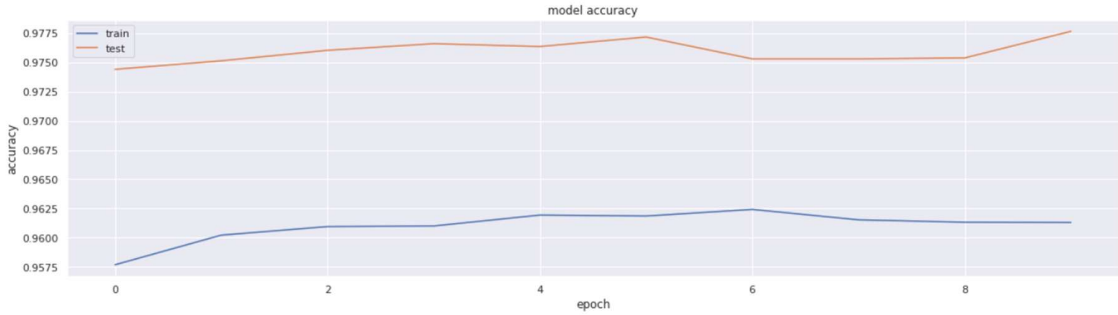


Рисунок 2.32 График изменения точности

Таким образом удалось поднять точность сети до 97.8% на тестовой и 96.3% на тренировочной выборке. Далее при увеличении до 4000 точность поднялась незначительно, 97.9% и 96.65% соответственно. В итоге поднимая постепенно размер пакета до 20000 изображений (при большем значении вызывается ошибка OutOfMemory, то есть переполнение оперативной памяти) мне удалось поднять точность до 98% и 96.9% соответственно. Дальше значения точности и ошибки лишь колеблются в небольшом диапазоне.

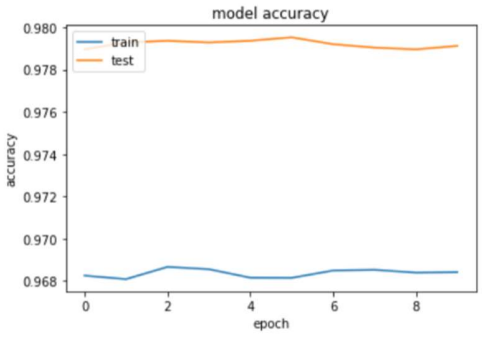


Рисунок 2.33 График изменения точности

2.7 Проверка работы нейронной сети, тестирование

```
#предсказываем тестовый набор
predictions = model.predict(x_test)
for i in range(2):
    result = np.argmax(predictions[i*500])
    print(enum_labels[result][1])
    plt.figure(figsize=(1, 1))
    plt.imshow(x_test[i*500])
    plt.colorbar()
    plt.grid(False)
    plt.show()
```

Рисунок 2.34 Код вывода предсказаний

С помощью такого нехитрого кода мы проходим по циклу и смотрим верно ли предсказала нейронная сеть тестовые данные. Бывает так, что в наборе данных был нарушен порядок и символы стали неправильно подписаны или не подписаны вовсе. Тогда на этом моменте это можно понять. В нашем же случае всё прошло успешно, все поданные на вход тестовые символы нейронная сеть распознаёт без труда.

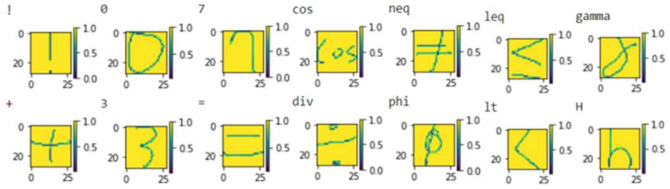


Рисунок 2.35 Картинки и их предсказания

2.8 Перевод выходного формата на язык LaTeX

Для перевода выходного формата в язык LaTeX я написал функцию для обработки выходных значений нейронной сети. У данной нейронной сети 82 выхода, каждому из которых я сопоставил соответствующий ему символ на языке LaTeX.

```
#перевод в язык latex:
def photo_to_latex(im_path,model):
    to_latex=[[0, '!'], [1, '('], [2, ')'], [3, '+'], [4, ','], [5, '-'], [6, '0'], [7, '1'], [8, '2'], [9, '3'], [10, '4'], [11,
    res = ""
    for letter in letters_extract(im_path):
        res += str(emnist_predict_img(model,letter[2],to_latex))+" "
    return res
```

```
res = photo_to_latex("math5.jpg",model)
print(res)
```

```
\sin = X - 6 \sum u f , ) \log \gamma \sum \times \infty 1 2 3 8 \leq \cos
```

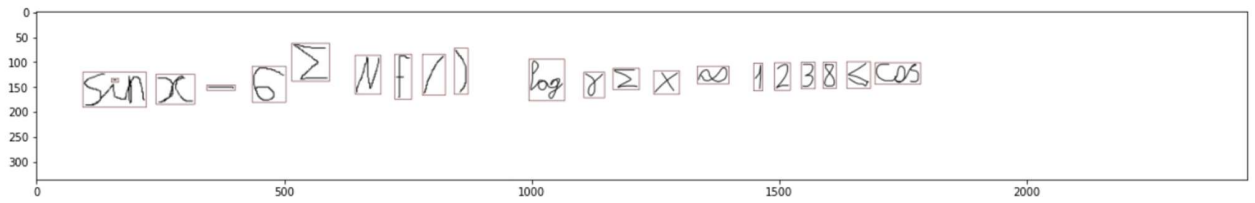


Рисунок 2.36 Код и работа функции перевода фото на LaTeX

2.9 Сбор последовательности: загрузка изображения – Сегментация – Нейронная сеть – перевод в LaTeX

Таким образом функция photo_to_latex объеденяет все звенья цепи. Для распознавания конкретного символа надо лишь загрузить его в среду Google Colaboratory и вызвать функцию с именем файла и обученной моделью в качестве параметров. [Приложение А]

В цикле изображение разбивается на символы и по очереди символы подаются на вход нейронной сети, на выходе символ расшифровывается в LaTeX символ через массив связей to_latex и конкатенируется с общей строкой выхода. В итоге данная функция полностью реализует нужный функционал.

2.10 Тесты нейронной сети

```
res = photo_to_latex("math8.jpg",model)
print(res)
```


```
\sin X - 6 \sum N f , ) \log \gamma \sum X \infty 1 2 3 8 \leq \cos
```

Рисунок 2.37 Тест 1

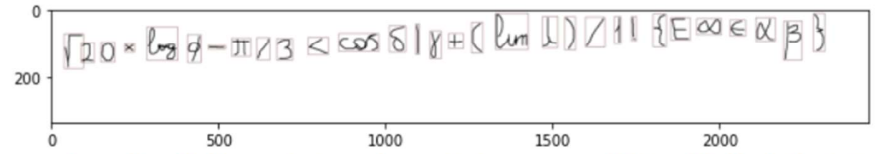
Такой выход можно спокойно переводить в LaTeX через math mode в OverLeaf. [37]

$$\sin X - 6 \sum N f,) \log \gamma \sum X \infty 1238 \leq \cos$$

Рисунок 2.37 Перевод на LaTeX в OverLeaf

Ошибки всё же иногда присутствуют, порой нейронная сеть путает похожие символы, а иногда выдаёт странные ответы (x перепутала с cos, \sqrt и e). Причиной этого может быть однообразный набор данных, с одной толщиной линий, в некоторых случаях символы могли быть начерчены не качественно. Поэтому есть еще много перспектив для улучшения работы данной нейронной сети.

```
res = photo_to_latex("math9.png",model)
print(res)
```




```
e 2 0 \cos \log \phi = \pi f 3 k \cos 8 \char124 8 + ( \lim d ) \backslash f \char124 { e \infty G d \beta }
```

Рисунок 2.38 Тест 2

$$e20 \cos \log \phi = \pi f 3k \cos 8|8 + (\lim d) \backslash f |\{e \infty G d \beta\}$$

Рисунок 2.39 Перевод на LaTeX в OverLeaf

Протестировать получившийся код может любой желающий по ссылке в Приложении Б.

ЗАКЛЮЧЕНИЕ

1. В этой курсовой работе были рассмотрены существующие решения для решения проблемы распознавания математических символов по изображению.

2. Был найден подходящий для данной цели набор данных.

3. Набор данных был стандартизирован и приведен к необходимому виду для обучения нейронной сети на.

4. Были рассмотрены различные архитектуры, структуры нейронных сетей, проведен анализ наилучших параметров для создания модели.

5. Нейронная сеть была обучена с помощью расширенного набора данных, модель нейронной сети была доведена до предела обучаемости и достигла отличных результатов точности.

6. Выходные данные нейронной сети были специально стандартизированы для использования в системе LaTeX.

7. Программа была собрана в единую структуру, для упрощения работы с ней.

8. Обученная модель нейронной сети была проверена с помощью реальных данных не из тестовой выборки, проверена целостность и точность итоговой программы.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Photomath official site [Электронный ресурс] – Режим доступа: https://photomath.com/ru/
2. Detexify LaTeX letters recognition [Электронный ресурс] – Режим доступа: http://detexify.kirelabs.org/classify.html
3. “Google Handwriting Input in 82 languages on your Android mobile device” [Статья] – Режим доступа: https://ai.googleblog.com/2015/04/google-handwriting-input-in-82.html
4. «Google улучшила рукописный ввод в Gboard с помощью ИИ» [Статья] – Режим доступа: https://www.kv.by/news/1056765-google-uluchshila-rukopisnyy-vvod-v-gboard-s-pomoshchyu-ii
5. Официальный сайт проекта “LaTeX” [Электронный ресурс] – Режим доступа: https://www.latex-project.org/
6. The Free Encyclopedia Wikipedia, “Neural Network” [Статья] – Режим доступа: https://en.wikipedia.org/wiki/Neural_network
7. The Free Encyclopedia Wikipedia, “Project Jupyter” [Статья] – Режим доступа: https://en.wikipedia.org/wiki/Project_Jupyter
8. Linda G. Shapiro and George C. Stockman (2001): «Computer Vision», pp 279—325, New Jersey, Prentice-Hall, ISBN 0-13-030796-3
9. The Free Encyclopedia Wikipedia, “Свёрточная нейронная сеть” [Статья] – Режим доступа: https://ru.wikipedia.org/wiki/Свёрточная_нейронная_сеть
10. Сайт об электронике, робототехнике и Arduino [Электронный ресурс] – Режим доступа: http://arduino.zl3p.com/robots/NN_CNN

11. Big Data School, “КАК СФОРМИРОВАТЬ ДАТАСЕТ ДЛЯ МАШИННОГО ОБУЧЕНИЯ” [Статья] – Режим доступа: https://www.bigdataschool.ru/blog/dataset-data-preparation.html
12. The Free Encyclopedia Wikipedia, “TensorFlow” [Статья] – Режим доступа: https://ru.wikipedia.org/wiki/TensorFlow
13. The Free Encyclopedia Wikipedia, “Keras” [Статья] – Режим доступа: https://ru.wikipedia.org/wiki/Keras
14. Официальный сайт разработчиков TensorFlow [Электронный ресурс] – Режим доступа: https://www.tensorflow.org/datasets/catalog/emnist
15. Официальный сайт онлайн-сообщества специалистов по данным и специалистов по машинному обучению Kaggle. [Электронный ресурс] – Режим доступа: https://www.kaggle.com/datasets
16. The Free Encyclopedia Wikipedia, “Kaggle” [Статья] – Режим доступа: https://ru.wikipedia.org/wiki/Kaggle
17. Официальный сайт онлайн-сообщества специалистов по данным и специалистов по машинному обучению Kaggle. [Электронный ресурс] – Режим доступа: https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols
18. Google Colaboratory [облачная среда для работы с кодом] https://colab.research.google.com/
19. Аналитическая платформа Loginom. [Статья] – Режим доступа: https://wiki.loginom.ru/articles/data-normalization.html
20. Официальная документация библиотеки OpenCV. [Статья] – Режим доступа: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
21. Открытый информационный портал «Русские блоги», ”Обработка порога изображения функцией cv2.threshold () (python)” [Статья] – Режим доступа: https://russianblogs.com/article/2776397280/

22. Открытый информационный портал “OpenCV-Python Tutorials”, “Morphological Transformations”. [Статья] – Режим доступа: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html
23. Информационный обучающий портал по обнаружению объектов и глубокому обучению “Pyimagesearch”, “OpenCV Contour Approximation” [Статья] – Режим доступа: https://pyimagesearch.com/2021/10/06/opencv-contour-approximation/
24. Обучающий информационный портал “Proproprogs”, «Структура и принцип работы полносвязных нейронных сетей» [Статья] – Режим доступа: https://proproprogs.ru/neural_network/struktura-i-princip-raboty-polnosvyaznyh-neyronnyh-setey
25. Обучающий информационный портал “Proproprogs”, «Структура и принцип работы полносвязных нейронных сетей» [Статья] – Режим доступа: https://proproprogs.ru/neural_network/struktura-i-princip-raboty-polnosvyaznyh-neyronnyh-setey
26. “Deeper Networks for Pavement Crack Detection”, Leo Paulya, Harriet Peela, Shan Luoa, David Hoggb and Raul Fuentesa, стр 3, [Электронный документ] – Режим доступа: https://www.researchgate.net/publication/319235847_Deeper_Networks_for_Pavement_Crack_Detection#pf3
27. The Free Encyclopedia Wikipedia, “SoftMax” [Статья] – Режим доступа: https://ru.wikipedia.org/wiki/Softmax
28. The Free Encyclopedia Wikipedia, “Метод обратного распространения ошибки” [Статья] – Режим доступа: https://ru.wikipedia.org/wiki/Метод_обратного_распространения_ошибки

29. Открытый информационный портал “Medium”, “Fully Connected vs Convolutional Neural Networks. Implementation using Keras”. [Статья] – Режим доступа: https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5
30. Открытый информационный портал “Medium”, “Сверточные нейронные сети в TensorFlow”. [Статья] – Режим доступа: https://medium.com/@bigdataschool/сверточные-нейронные-сети-в-tensorflow-ecb13171e001
31. The Free Encyclopedia Wikipedia, “LeNet” [Статья] – Режим доступа: https://en.wikipedia.org/wiki/LeNet#Structure
32. Открытый информационный портал “TowardsDatascience”, “Implementing AlexNet CNN Architecture Using TensorFlow 2.0+ and Keras”, [Статья] – Режим доступа: https://towardsdatascience.com/implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98
33. “ImageNet Classification with Deep Convolutional Neural Networks”, Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. Стр 5. [Электронный документ] – Режим доступа: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
34. Официальный справочник по API Keras. “Инициализаторы веса слоя” [Статья] – Режим доступа: https://keras.io/api/layers/initializers/
35. Добавлено из статьи “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.
Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(56):1929−1958, 2014.”

36. Обучающий информационный ресурс “MachineLearningMastery”, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning”. [Статья] – Режим доступа: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
37. Онлайн редактор LaTeX “Overleaf”. [Электронный ресурс] – Режим доступа: https://www.overleaf.com/

ПРИЛОЖЕНИЕ А.

Код функции photo_to_latex

Ниже приведен код алгоритма на Python

```python
#перевод в язык latex:
def photo_to_latex(im_path,model):
    to_latex=[[0, '!'], [1, '('], [2, ')'], [3, '+'], [4, ','], [5, '-'], [6, '0'],
              [7, '1'], [8, '2'], [9, '3'], [10, '4'], [11, '5'], [12, '6'], [13, '7'],
              [14, '8'], [15, '9'], [16, '='], [17, 'A'], [18, 'C'], [19, '\Delta'], [20, 'G'],
              [21, 'H'], [22, 'M'], [23, 'N'], [24, 'R'], [25, 'S'], [26, 'T'], [27, 'X'],
              [28, '['], [29, ']'], [30, '\alpha'], [31, '\char124'], [32, 'b'], [33, '\beta'],
              [34, '\cos'], [35, 'd'], [36, '\div'], [37, 'e'], [38, '\exists'], [39, 'f'],
              [40, '\forall'], [41, '\backslash'], [42, '\gamma'], [43, '\geq'], [44, '>'],
              [45, 'i'], [46, '\in'], [47, '\infty'], [48, '\int'], [49, 'j'], [50, 'k'],
              [51, 'l'], [52, '\lambda'], [53, '...'], [54, '\leq'], [55, '\lim'], [56, '\log'],
              [57, '<'], [58, '\mu'], [59, '\neq'], [60, 'o'], [61, 'p'], [62, '\phi'], [63, '\pi'],
              [64, '\pm'], [65, '\prime'], [66, 'q'], [67, '\Rightarrow'], [68, '\sigma'], [69, '\sin'],
              [70, '\sqrt'], [71, '\sum'], [72, '\tan'], [73, '\theta'], [74, '\\times'],
              [75, 'u'], [76, 'v'], [77, 'w'], [78, 'y'], [79, 'z'], [80, '{'], [81, '}']]
    res = ""
    for letter in letters_extract(im_path):
        res += str(emnist_predict_img(model,letter[2],to_latex))+" "
    return res
```

ПРИЛОЖЕНИЕ Б.

Полный исходный код

https://colab.research.google.com/drive/11pBRPWIE99x2xlTOlHCux_4yXTuCWeNf?usp=sharing