Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский Государственный Морской Технический Университет

Факультет цифровых промышленных технологий

**Курсовой проект**

**На тему**: Распознавание количество пальцев (CNN)

**По предмету:** Методы и средства глубокого обучения и нейросети

Работу выполнил студент группы

№ 20126 Е.К. Борисов

(подпись)

Работу принял преподаватель П.П. Поделенюк

(подпись)

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Санкт-Петербург

2024

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ 3](#_Toc167588363)

[ВВЕДЕНИЕ 4](#_Toc167588364)

[Глава 1. Набор данных 5](#_Toc167588365)

[Глава 2. Подготовка данных 6](#_Toc167588366)

[Глава 3. Создание и обучение модели 10](#_Toc167588367)

[Глава 4. Результаты работы сети 13](#_Toc167588368)

[ЗАКЛЮЧЕНИЕ 14](#_Toc167588369)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 15](#_Toc167588370)

# ПЕРЕЧЕНЬ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

|  |  |
| --- | --- |
| Датасет | Набор данных |
| CNN | Cверточная нейронная сеть (Convolutional Neural Network) |
| Tensorflow | Библиотека для работы с нейронными сетями |

ВВЕДЕНИЕ

В современном мире компьютерное зрение и автоматическое распознавание образов играют ключевую роль в различных сферах, начиная от медицинской диагностики и биометрической аутентификации до автоматического управления производственными процессами и обнаружения объектов на дорогах. Одной из важных задач в области компьютерного зрения является распознавание и классификация различных объектов на изображениях. В этом контексте распознавание пальцев на изображениях имеет большое практическое значение, особенно в системах биометрической аутентификации и контроля доступа.

Сверточные нейронные сети (CNN) представляют собой мощный класс алгоритмов глубокого обучения, который широко применяется в задачах компьютерного зрения, включая распознавание объектов на изображениях. Используя методику изучения признаков иерархическим образом, CNN способны автоматически извлекать и анализировать различные характеристики объектов на изображениях, что делает их идеальным выбором для задач распознавания пальцев.

Цель данной курсовой работы состоит в разработке и реализации системы автоматического распознавания количества пальцев на изображениях с использованием сверточных нейронных сетей.

1. Набор данных

В данной работе я буду использовать готовый датасет черно-белых картинок. <https://www.kaggle.com/datasets/koryakinp/fingers/data>. Данный датасет содержит 21600 изображений пальцев правой и левой рук. Метки классов находятся в двух последних символах имени файла. L/R обозначает левую/правую руку; 0,1,2,3,4,5 обозначает количество пальцев. Примеры изображений показаны на рисунке 1.



Рисунок 1 – Примеры изображений.

Датасет содержит 3600 тестовых изображений и 18 тысяч изображений для тренировки.

1. Подготовка данных

Для начала необходимо выполнить предварительную обработку изображений. Для обработки изображений будем использовать ImageDataGenerator.

ImageDataGenerator — это класс из библиотеки TensorFlow/Keras, который предоставляет возможность автоматической предварительной обработки изображений во время обучения нейронной сети. Он позволяет гибко настраивать предварительную обработку, включая масштабирование, изменение размера, аугментацию данных и другие техники.

Код предварительной обработки изображений показан на рисунке 2.



Рисунок 2 – Предварительная обработка изображений.

Для начала были созданы переменные, отвечающие за пути к месторасположению обучающих и тренировочных данных, а также параметры размера изображения и размер пакета, определяющий количество обучающих примеров, которые обрабатываются моделью перед обновлением её параметров.

После чего была объявлена функция extract\_label, которая достает из имени файла название метки класса. Название картинок имеет следующий формат “0a8c5a58-d75f-4e48-9f06-ac8f8f722ae6\_2R.png”, мы разбиваем строку на массив строк, а затем достаем номер класса изображения по индексу.

Далее создаются датафреймы тестовых и тренировочных данных при помощи функции create\_dataframe. Функция create\_dataframe принимает путь к директории с изображениями в качестве аргумента. Внутри функции происходит обход всех файлов в указанной директории с помощью os.walk(). Для каждого файла с расширением .png или .jpg извлекается метка класса из его названия с помощью функции extract\_label, а путь к файлу добавляется в список filename. После обработки всех файлов создается DataFrame, в котором каждая строка содержит путь к изображению и соответствующую метку класса. Эти датафреймы затем используются для генератора тестовых, тренировочных и валидационных данных при помощи ImageDataGenerator.

Создаются два объекта ImageDataGenerator: train\_datagen и test\_datagen. Параметр rescale=1./255 применяет масштабирование пиксельных значений изображений, чтобы они находились в диапазоне от 0 до 1.

В аргументы генератора передаются:

* Соответвующий датафрейм. ( Тестовый или тренировочный )
* Ключ столбца датафрейма содержащего изображения. (x\_col)
* Ключ столбца датафрейма содержащего метки классов.(y\_col)
* Размер изображений. (target\_size)
* Размер пакета. (batch\_size)
* Это режим класса, определяющий формат меток классов. Categorical указывает, что метки классов представлены в форме one-hot encoded. (class\_mode)
* Параметр, указывающий на то, нужно ли перемешивать данные перед каждой эпохой (shuffle)
* Параметр указывающий, какую часть данных использовать. (subset)
* Режим цветов изображения, так как в нашем случае изображения черно-белые, указывается знание grayscale.

train-generator - используется для обучения модели нейронной сети.

validation\_generator - используется для оценки производительности модели на отложенной валидационной выборке во время обучения.

test\_generator - используется для тестирования обученной модели на отдельном тестовом наборе данных.

1. Создание и обучение модели

Для начала создадим опишем архитектуру модели сети. Код и архитектура модели показаны на рисунке 3.

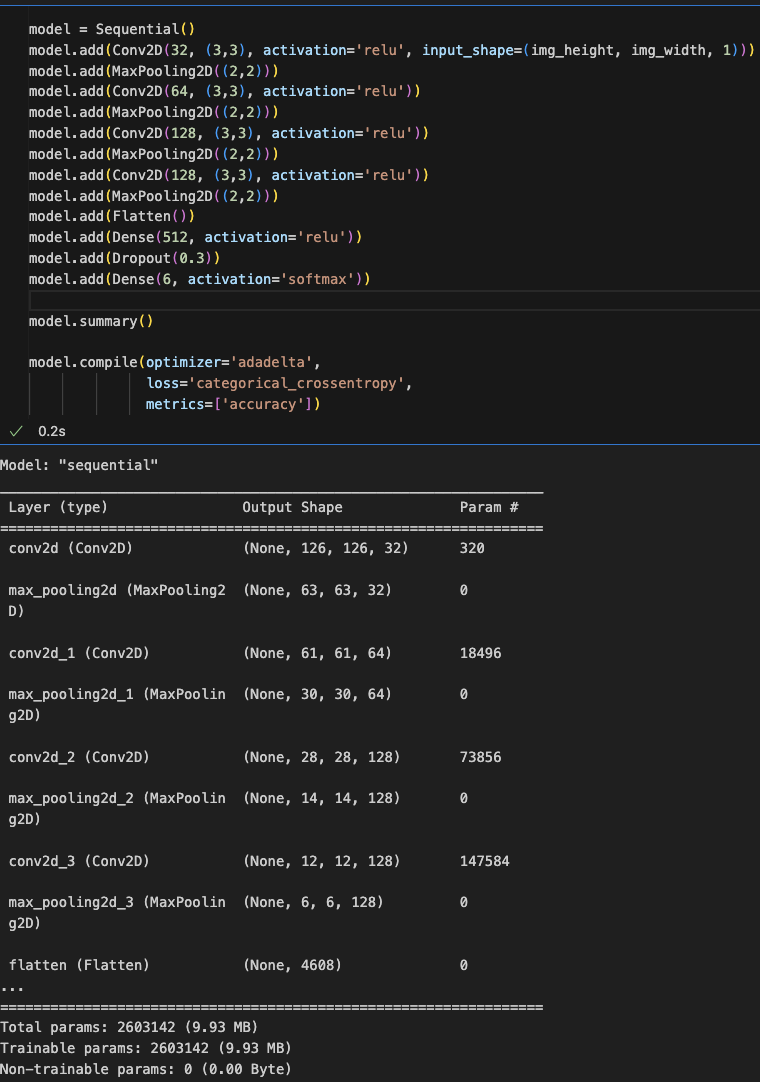


Рисунок 3 – Создание модели.

Мы используем CNN модель для классификации изображений. Вначале идут 4 сверточных слоя для извлечения признаков из изображений. Каждый сверточный слой применяет ядро размером 3x3 с активацией ReLU, чтобы выделить важные характеристики изображений. Первый сверточный слой имеет 32 фильтра для выделения основных признаков, второй слой увеличивает количество фильтров до 64 для извлечения более сложных признаков, а третий и четвертый слои содержат по 128 фильтров для более глубокого анализа изображений. После каждого сверточного слоя следует слой пулинга MaxPooling с ядром размером 2x2 для уменьшения размерности данных. После последнего сверточного слоя следует слой сглаживания Flatten, который преобразует многомерные данные в одномерный вектор. Затем идут два полносвязных слоя. Первый слой содержит 512 нейронов с активацией ReLU, а второй слой имеет 6 нейронов с активацией softmax для классификации изображений на 6 классов. Для уменьшения переобучения в модели применяется слой Dropout, который случайным образом отключает часть нейронов во время обучения. Модель компилируется с использованием оптимизатора Adadelta, функции потерь категориальной кросс-энтропии и используется метрика точности.

Далее при помощи функции fit мы обучаем модель. В этом методе мы передаем генераторы данных train\_generator и validation\_generator, которые будут предоставлять обучающие и валидационные данные соответственно. Количество шагов на каждой эпохе определяется как количество образцов в обучающем наборе данных, деленное на размер пакета. Таким образом, модель будет обновлять свои веса после каждого шага, с учетом предоставленных данных.

После нескольких тестов было принято решение использовать 30 эпох, так как далее модель, по сути, перестает обучаться. Процесс обучения показан на рисунке 4.

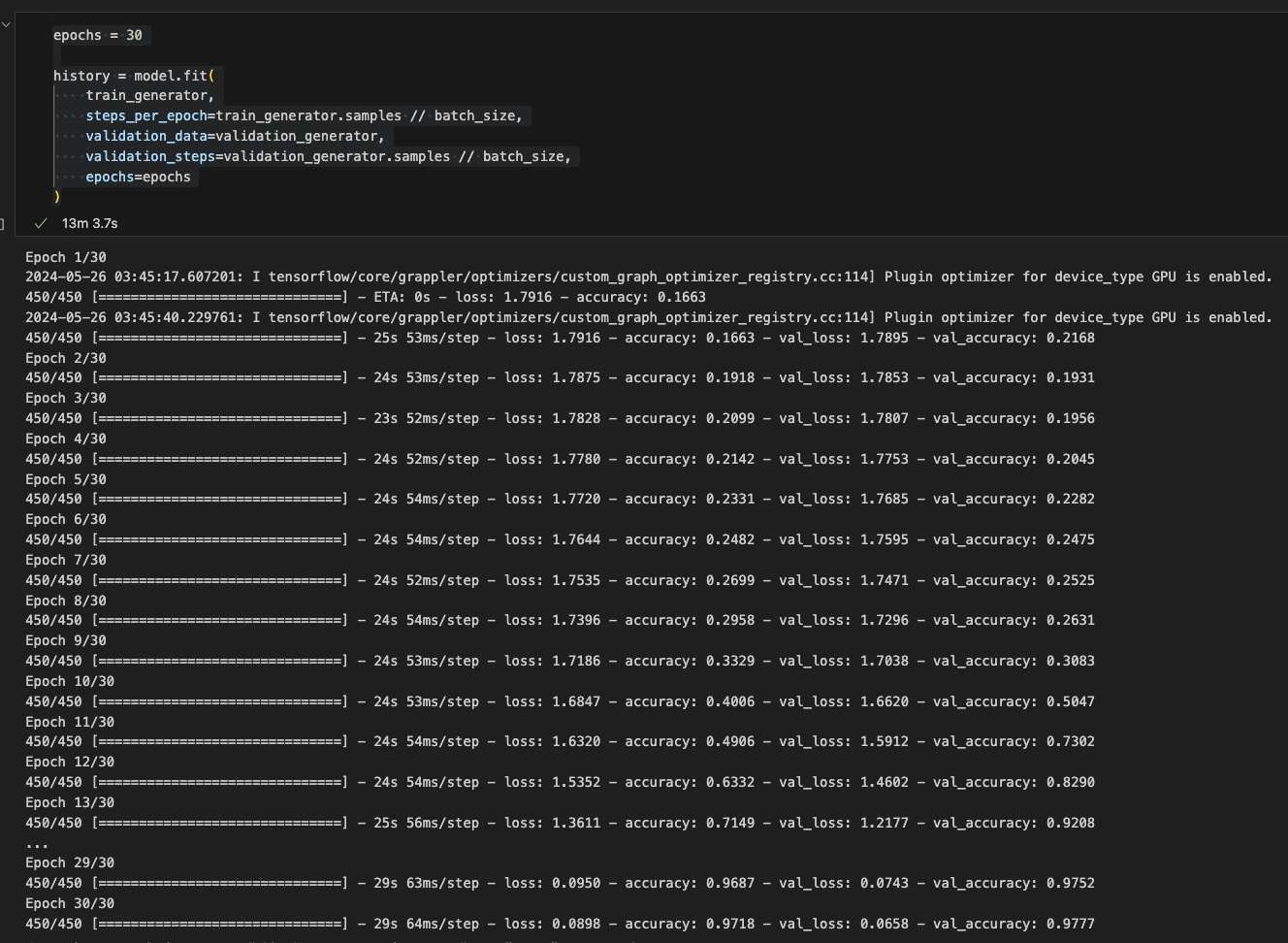


Рисунок 4 – Процесс обучения сети.

На протяжении всех эпох видно, как значения функции потерь уменьшаются, а точность увеличивается как на обучающем, так и на валидационном наборах данных. Это свидетельствует о том, что модель постепенно улучшает свои предсказательные способности и способна лучше обобщать данные. В целом, результаты обучения указывают на то, что модель хорошо справляется с задачей классификации и успешно обучается на предоставленных данных.

1. Результаты работы сети

Проверим точность модели на тестовых данных. Результат показан на рисунке 5.

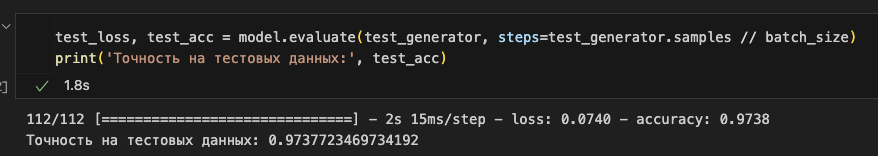


Рисунок 5 – Точность модели на тестовых данных.

Как видно из рисунка, точность модели составила больше 97 процентов, что говорит о хорошем результате обучения. На рисунке 6 показана демонстрация работы сети.

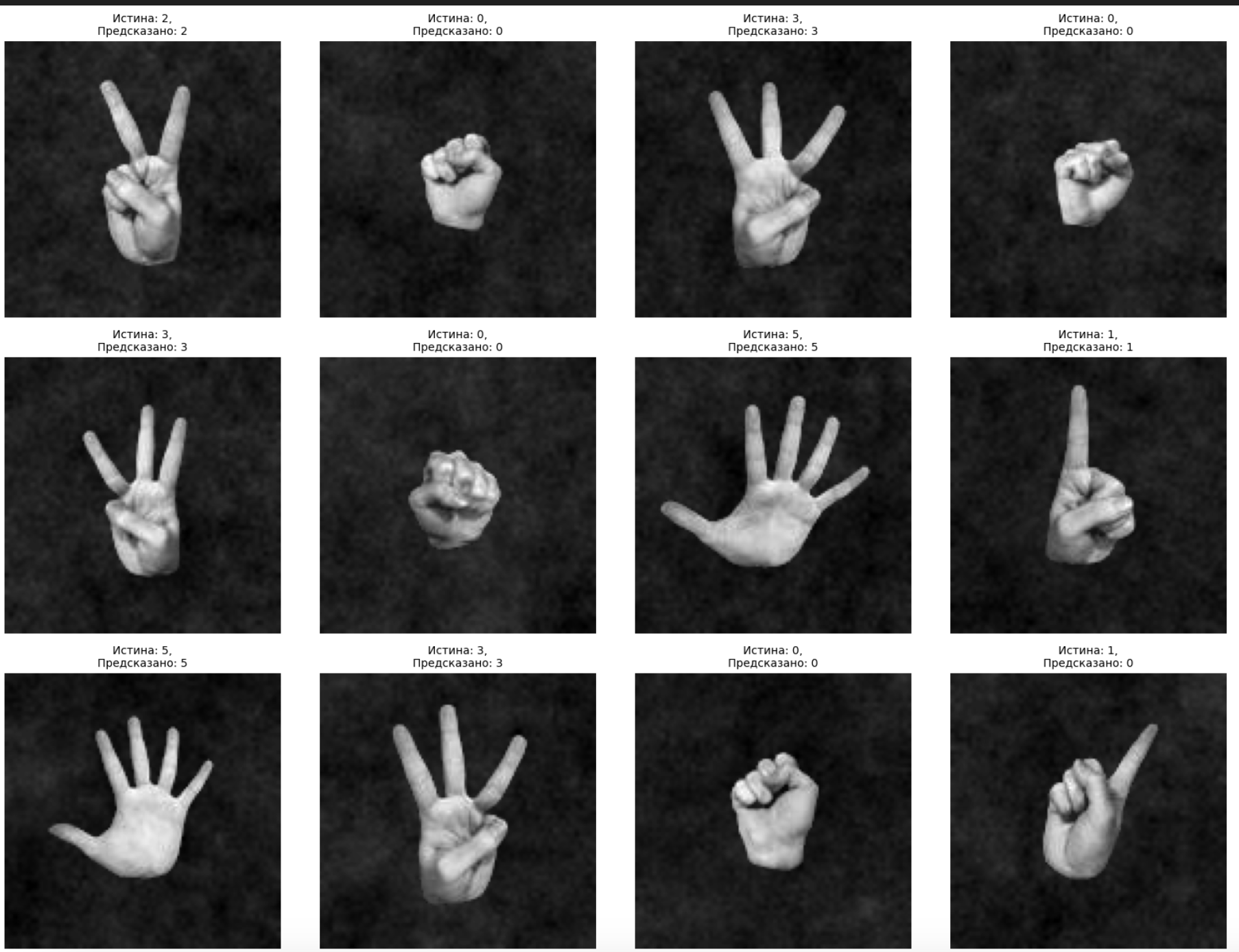


Рисунок 6 – Демонстрация работы сети.

# ЗАКЛЮЧЕНИЕ

В ходе этой работы была разработана и обучена сверточная нейронная сеть для классификации изображений. Исходные изображения были подготовлены путем создания датафреймов, в которых каждому изображению была присвоена соответствующая метка. Для обработки данных и создания генераторов был использован ImageDataGenerator. Модель состояла из четырех сверточных слоев с активацией ReLU, слоев подвыборки (MaxPooling), слоя сглаживания (Flatten), полносвязных слоев (Dense) и слоя Dropout для предотвращения переобучения.

Для обучения модели были использованы данные из генераторов. Модель была скомпилирована с использованием оптимизатора Adadelta и функции потерь категориальной кросс-энтропии. Обучение проводилось в течение 30 эпох.

В процессе обучения наблюдалось последовательное уменьшение значения функции потерь и увеличение точности как на обучающем, так и на валидационном наборах данных.

Разработанная модель успешно справилась с задачей классификации изображений, продемонстрировав хорошие результаты на тестовом наборе данных без явных признаков переобучения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Нейронные сети для начинающих // HABR URL: <https://habr.com/ru/articles/312450/>
2. Сверточная нейронная сеть, часть 1. // HABR URL: <https://habr.com/ru/articles/348000/>
3. Выбор слоя активации в нейронных сетях: как правильно выбрать для вашей задачи // HABR URL: <https://habr.com/ru/articles/727506/>

**ЛИСТИНГИ**

Листинг 1. Исходный код.

|  |
| --- |
| import tensorflow as tf  from tensorflow.keras.preprocessing.image import ImageDataGenerator, load\_img, img\_to\_array  from tensorflow.keras.models import Sequential, load\_model  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout  import os  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import cv2  train\_dir = './archive2/fingers/train'  test\_dir = './archive2/fingers/test'  img\_height, img\_width = 128, 128  batch\_size = 32  def extract\_label(filename):  return filename.split('\_')[1][0]  def create\_dataframe(directory):  data = {'filename': [], 'label': []}  for root, dirs, files in os.walk(directory):  for file in files:  if file.endswith('.png') or file.endswith('.jpg'):  filepath = os.path.join(root, file)  label = extract\_label(file)  data['filename'].append(filepath)  data['label'].append(label)  return pd.DataFrame(data)  train\_df = create\_dataframe(train\_dir)  test\_df = create\_dataframe(test\_dir)  train\_datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.2)  test\_datagen = ImageDataGenerator(rescale=1./255)  train\_generator = train\_datagen.flow\_from\_dataframe(  train\_df,  x\_col='filename',  y\_col='label',  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=True,  subset='training',  color\_mode='grayscale'  )  validation\_generator = train\_datagen.flow\_from\_dataframe(  train\_df,  x\_col='filename',  y\_col='label',  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=True,  subset='validation',  color\_mode='grayscale'  )  test\_generator = test\_datagen.flow\_from\_dataframe(  test\_df,  x\_col='filename',  y\_col='label',  target\_size=(img\_height, img\_width),  batch\_size=batch\_size,  class\_mode='categorical',  shuffle=False,  color\_mode='grayscale'  )  model = Sequential()  model.add(Conv2D(32, (3,3), activation='relu', input\_shape=(img\_height, img\_width, 1)))  model.add(MaxPooling2D((2,2)))  model.add(Conv2D(64, (3,3), activation='relu'))  model.add(MaxPooling2D((2,2)))  model.add(Conv2D(128, (3,3), activation='relu'))  model.add(MaxPooling2D((2,2)))  model.add(Conv2D(128, (3,3), activation='relu'))  model.add(MaxPooling2D((2,2)))  model.add(Flatten())  model.add(Dense(512, activation='relu'))  model.add(Dropout(0.3))  model.add(Dense(6, activation='softmax'))  model.summary()  model.compile(optimizer='adadelta',  loss='categorical\_crossentropy',  metrics=['accuracy'])  epochs = 30  history = model.fit(  train\_generator,  steps\_per\_epoch=train\_generator.samples // batch\_size,  validation\_data=validation\_generator,  validation\_steps=validation\_generator.samples // batch\_size,  epochs=epochs  )  model.save('finger\_count\_model.h5')  test\_loss, test\_acc = model.evaluate(test\_generator, steps=test\_generator.samples // batch\_size)  print('Точность на тестовых данных:', test\_acc)  def plot\_images\_with\_predictions(generator, model, num\_images):  images, labels = next(generator)  predictions = model.predict(images)  rows = (num\_images // 5) + 1  cols = 5    plt.figure(figsize=(20, 4 \* rows))  for i in range(num\_images):  ax = plt.subplot(rows, cols, i + 1)  plt.imshow(images[i].squeeze(), cmap='gray')  real\_class = np.argmax(labels[i])  predicted\_class = np.argmax(predictions[i])  plt.title(f'Истина: {real\_class},\nПредсказано: {predicted\_class}', fontsize=10)  plt.axis('off')  plt.tight\_layout()  plt.show()  model = load\_model('finger\_count\_model.h5')  plot\_images\_with\_predictions(test\_generator, model, 30) |