# Лабораторная работа 4. Компьютерное зрение в биометрической аутентификации

С развитием компьютерных технологий растет и заинтересованность злоумышленников в овладении ценной информацией, в том числе открывающей доступ к реальным материальным ресурсам. И, таким образом, перед разработчиками биометрических систем защиты информации возникают разнообразные задачи, одна из которых – поддержка идентификации и аутентификации пользователя, то есть выяснения его личности и подтверждения подлинности его личности, соответственно, для дальнейшего предоставления ему определенных прав, или авторизации.

Следует отметить несколько особенностей при использовании биометрической аутентификации для обеспечения информационной безопасности. Во-первых, сам по себе процесс аутентификации по биометрическим показателям имеет сравнительно невысокую надёжность. В «перестраховочных» системах, где допускается значительная вероятность ложноотрицательного срабатывания, использование биометрической информации как единственного фактора может затруднить доступ реальным пользователям, поэтому в таких системах различные факторы разрешается делать взаимозаменяемыми – например, аутентифицировать пользователя либо по биометрическому показателю, либо по паролю. Такой подход улучшает пользовательский опыт, но он не является, в общем случае, многофакторной аутентификацией. Следует помнить, что объединение факторов аутентификации через логическое «ИЛИ» уменьшает защищенность системы в целом, поскольку это увеличивает возможную площадь атаки для потенциального злоумышленника, т.к. для доступа в систему достаточно провести успешную атаку на любой из используемых факторов. Во-вторых, в отличие от традиционных факторов аутентификации на основе секретной информации (паролей, ключей, токенов, пин-кодов и т.п.) биометрические показатели, как правило, не могут быть изменены. Это значительно снижает надёжность при использовании таких факторов в случае утечки информации – при хранении биометрических показателей в исходном виде они могут быть сфабрикованы, что, в свою очередь, компрометирует использование этого биометрического фактора конкретным пользователем во всех других системах, где он используется. В-третьих, показатели для многих биометрических факторов могут меняться с течением времени. Если система это учитывает и допускает некоторую вариативность, площадь атаки на такие системы также увеличивается, поскольку для успешной подделки показателя требуется меньшая степень схожести.

В основе функционирования биометрических систем лежит цепочка действий:

1. Запись – считываются биометрические данные пользователя;
2. Обработка данных – из представленных данных извлекается уникальная информация, например, изображение или вектор признаков;
3. Сравнение – сравнивается образ с эталонами из базы данных системы;
4. Принятие решения – решение об окончании процедуры идентификации, ее повторении или изменении условий ее проведения.

На данный момент существует и продолжает разрабатываться достаточно много подходов, основанных на получении различных биометрических образов:

- сканирование радужной оболочки глаза;
- анализ геометрии лица;
- снятие отпечатков пальцев;
- анализ геометрии руки;
- сканирование сетчатка глаза;
- получение отпечатка ладони;
- термограмма лица и руки;
- динамические:
- походка;
- получение характеристик речи;
- рукописная подпись;
- клавиатурный почерк;
- и т.д.

**Практические примеры**

Существует большое количество узкоспециализированных алгоритмов, направленных как на выделение признаков и построения эталонной модели для сравнения данных пользователя, так и на процесс самого сравнения и принятия решения об аутентификации личности. Однако в данной лабораторной работе предлагается рассмотреть подход, основанный на сверточных нейронных сетях, как наиболее универсальном средстве выделения признаков, применимом и адаптируемым под разные виды графических данных.

Рассмотрим пример аутентификации человека по отпечаткам пальцев, на основании которого реализуем простейший классификатор личностей по отпечаткам пальцев. Подходящий датасет можно скачать по ссылке: https://www.kaggle.com/peace1019/fingerprint-dataset-for-fvc2000-db4-b.

Датасет содержит 2 каталога: с обучающими данными и с тестовыми. Обучающий каталог содержит каталог 800 изображений 10 пальцев, называемых по определенному паттерну «номер пальца»_«номер изображения для этого пальца». Каталог с тестовыми данными содержит 10 изображений к каждому из классов. Пример изображения представлен на рисунке 2:

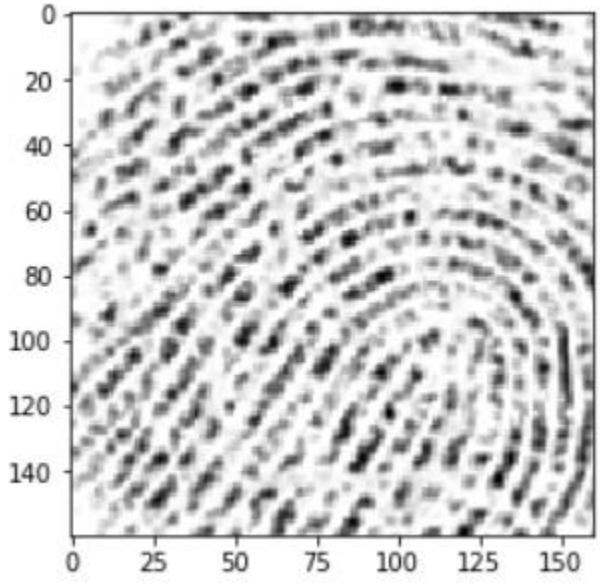



Рисунок 2. Пример изображения датасета

Для работы с изображениями можно использовать библиотеку OpenCV. Для установки библиотеки можно в командной строке использовать команду `pip install opencv-python`. Так как датасет в данном случае представлен набором изображений, а не сформирован в файл формата .csv, как было во всех предыдущих примерах, необходимо сначала корректно загрузить изображения в память. Основным этапом является получение списка всех возможных имен файлов. Сделать это можно с помощью функции `listdir` стандартного модуля python `os`:

```
from os import listdir

images_dir = 'D:/lab4/fingerprint-dataset-for-fvc2000-db4-b/dataset_FVC2000_DB4_B/dataset/train_data'
file_names = listdir(images_dir)
sample_count = len(file_names)
print(sample_count)
```

Если все работает правильно, то код, приведенный выше, должен вывести количество файлов в указанном каталоге. Вывести первое

изображение и получить данные о его размере можно с помощью кода, приведенного ниже:

```
import matplotlib.pyplot as plt

first_image = cv2.imread(images_dir + '/' + file_names[0],
cv2.IMREAD_GRAYSCALE)
plt.imshow(first_image, cmap='gray')
height, width = first_image.shape
```

Для дальнейшей работы с изображениями и для подачи их на вход модели, необходимо получить полный путь к файлу и загрузить их в переменные. Сделать это можно с помощью простейшей конкатенацией строк из пути до каталога и названий файлов:

```
from os.path import splitext
import pandas as pd

depth = 1
X = np.zeros((sample_count, width, height, depth))
y = np.zeros((sample_count, 1), dtype=int)
for index, file_name in enumerate(file_names):
    full_path = images_dir + '/' + file_name
    file_label_text, file_number_and_ext = file_name.split('_')
    image = cv2.imread(full_path, cv2.IMREAD_GRAYSCALE)
    label = int(file_label_text)
    X[index] = np.reshape(image, (width, height, depth))
    y[index] = label
```

Функция `enumerate` позволяет оборачивать любой перечислимый тип и возвращает для него элементы в паре с индексом, дополняя тем самым цикл `for-in`, который в python не осуществляет индексацию.

Выходное значение зададим в виде «one-hot» представления:

```
y_one_hot = np.zeros((sample_count, unique_classes))
for index, label in enumerate(y):
    y_one_hot[index, label] = 1
```

Модель сверточной сети отличается типом используемых в ней слоев. Слой, обозначающийся как `Conv2D`, является сверточным слоем. Обязательный первый параметр задаёт количество ядер свёртки, а параметр `kernel_size` -–размер каждого из ядра (в примере ниже используются ядра размером 3x3). Слой `MaxPooling` задаёт слой субдискретизации, параметр `pool_size` позволяет указывать размер области субдискретизации по одной оси. Слой `Flatten` позволяет преобразовать выход со сверточного слоя в виде

результата сверток по всем ядрам к векторному виду для подачи на следующий слой. Пример создания модели сверточной сети с помощью библиотеки Keras и класса `Sequential`:

```
from tensorflow.keras import layers, losses, metrics, optimizers
from tensorflow.keras.models import Sequential

classifier = Sequential([
    layers.Conv2D(32,      kernel_size=3,      activation='relu',
input_shape=(height, width, depth)),
    layers.MaxPooling2D(pool_size=2),
    layers.Conv2D(64, kernel_size=3, activation='relu'),
    layers.MaxPooling2D(pool_size=2),
    layers.Conv2D(128, kernel_size=3, activation='relu'),
    layers.MaxPooling2D(pool_size=2),
    layers.Flatten(),
    layers.Dense(256),
    layers.Dense(1, activation='sigmoid')
])

classifier.compile(loss='binary_crossentropy',
optimizer=optimizers.RMSprop(), metrics=['accuracy'])
classifier.summary()
```

При обучении моделей всегда важным и полезным является рассмотрение того, как ведет себя модель на обучающей и валидационной выборках. Обычно для этого выводят графики зависимостей точности и функции стоимости или потерь (Loss) от номера итерации. Вывести их можно с помощью библиотеки livelossplot, предварительно установив ее с помощью командной строки (`pip install livelossplot`).

```
from livelossplot import PlotLossesKeras
classifier.fit(X,      y_single,      verbose=0,      epochs=50,
validation_split=0.2, callbacks=[PlotLossesKeras()])
```

Для проверки корректности работы сети можно подать на вход обученной модели изображения из каталога «real_data».

Данный пример иллюстрирует возможность применения сверточной нейронной сети для анализа изображений. В данном случае каждый пользователь являлся отдельным классом для модели, и обученная модель оказалась способна провести классификацию пользователей по их отпечатку пальца. Теоретически, с помощью такой модели можно разграничить доступ в системе или вести логирование, задав, например, еще один класс гостевого пользователя (заранее не авторизованного в системе), для которого будет низкая активация на каждый из приведенных заранее классов. Однако, как

правило, задача в системах биометрической аутентификации отличается от классификации по известным классам. Практически у каждого сейчас есть современный смартфон с возможностью биометрической аутентификации по одному из рассмотренных выше факторов (чаще всего это отпечаток пальца или геометрия лица). При этом, когда пользователь добавляет свои данные, чтобы настроить доступ, переобучить систему на всех пользователях в мире и внести пользователя в список как один из распознаваемых классов просто невозможно. От системы ожидается, что она создаст некоторое представление или шаблон, на основании сравнения с которым будет ограничен доступ к устройству. Так что же в данном случае должно быть обучающей выборкой? Ведь если кто-то обучил нейронную сеть на некотором наборе людей, почему она должна правильно определять какого-либо другого человека? В данном случае необходимо каким-то образом обучить систему определять «схожесть с шаблоном».

Рассмотрим другой пример биометрической аутентификации - по почерку. Датасет к этому примеру можно скачать по ссылке: https://www.kaggle.com/robinreni/signature-verification-dataset.

Датасет содержит набор изображений с 69 подписями, при этом для каждой подписи есть «подлинные» экземпляры, расположенные в каталогах с названием «№» и «подделанные» – в каталогах с названием «№_forg». Обратите внимание, что для реализации распознавания «подлинная/подделанная» по подписи для одного пользователя примеров слишком мало (порядка 10 изображений для каждого класса). Задачей в данном случае является обучение модели на распознавание «схожести» между образцами. Для этого можно сформировать датасет из всех возможных пар подписей, в которых одна подпись используется как образец, а вторая – как оцениваемый пример. Таким образом, для пар, в которых оба изображения являются подлинными образцами одной и той же подписи, будем ожидать на выходе значение «1», а если образец является подлинным, а пример – подделкой, будем ожидать значение «0». Обратите внимание, что при формировании датасета в парах друг с другом рассматриваются только экземпляры одной и той же подписи – оригиналы с оригиналами и оригиналы с подделками. Таким образом, задача сводится к бинарной классификации. При этом модель не обучается отличать заранее подготовленные подписи друг от друга, а вместо этого обучается определять, является ли произвольная подпись подлинной относительно предоставленного подлинного образца.

Функция, преобразовывающая входные данные в такой вид, может быть следующей:

```
import numpy as np
```

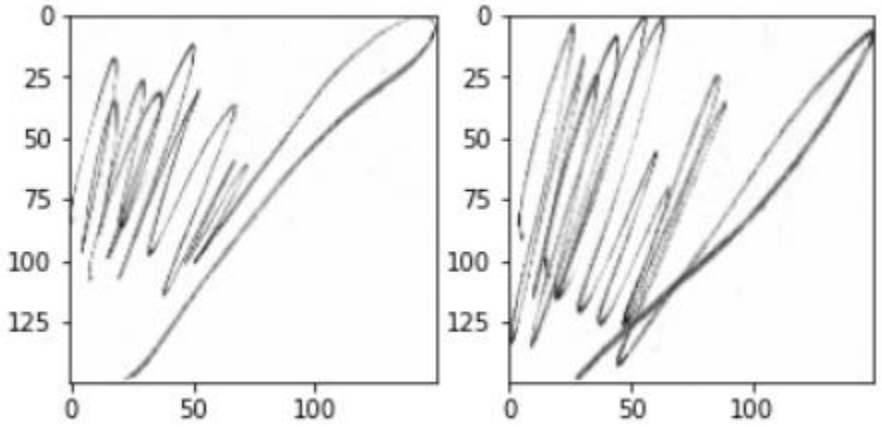
```
import cv2
from os import listdir

def load_data_from_folder(images_dir, height, width, depth):
    dir_names = listdir(images_dir)
    dir_count = len(dir_names)

    images_real = {}
    images_forgery = {}
    labels = set()

    for dir_name in dir_names:
        parts = dir_name.split('_')
        label_text = parts[0]
        is_forgery = len(parts) > 1

        if (label_text not in labels):
            labels.add(label_text)

        if label_text not in images_real:
            images_real[label_text] = []

        if label_text not in images_forgery:
            images_forgery[label_text] = []

        image_file_names = listdir(images_dir + '/' + dir_name)
        for image_file_name in image_file_names:
            image = cv2.imread(images_dir + '/' + dir_name + '/'
+ image_file_name, cv2.IMREAD_GRAYSCALE)
            if is_forgery:
                images_forgery[label_text].append(image)
            else:
                images_real[label_text].append(image)
    X_base_list = []
    X_comparison_list = []
    y_list = []

    for label in labels:
        real_samples = images_real[label]
        forged_samples = images_forgery[label]

        for real_img in real_samples:
            real_resized = cv2.resize(real_img, (width, height))
            for another_real in real_samples:
                another_resized = cv2.resize(another_real,
(width, height))
                X_base_list.append(real_resized.reshape((width,
height, depth)))

X_comparison_list.append(another_resized.reshape((width, height,
depth)))
                y_list.append(1)
```

```
                for another_fake in forged_samples:
                    fake_resized = cv2.resize(another_fake, (width,
height))
                    X_base_list.append(real_resized.reshape((width,
height, depth)))

X_comparison_list.append(fake_resized.reshape((width, height, depth)))
                    y_list.append(0)

        X_base = np.array(X_base_list)
        X_comparison = np.array(X_comparison_list)
        y = np.array(y_list)

        return X_base, X_comparison, y
```

Кроме формирования всевозможных пар изображений, в данной функции они также приводятся к одному размеру с помощью функции `resize` модуля `cv2` библиотеки OpenCV. Чтобы сформировать датасет с помощью этой функции, необходимо передать в нее путь к каталогу и размеры изображений (к которым необходимо привести входные изображения).

Вывести одну из пар изображений и ожидаемый результат можно при помощи следующих операций:

```
import matplotlib.pyplot as plt

index = 3
f, axes = plt.subplots(1,2)
axes[0].imshow(X_base[index, :, :, 0], cmap='gray')
axes[1].imshow(X_comparison[index, :, :, 0], cmap='gray')
print(f'Label: {y[real_index]}')
```



В отличие от большинства моделей, которые рассматривались до сих пор, задача сравнения двух изображений решается с использованием пары параллельных сверточных нейронных сетей с общими весами, результаты работы которых дополнительно подаются на слой активации, после чего для вычисления разницы между изображениями вектор активации с одной сети вычитается из вектора активации другой сети. Поскольку такая сеть имеет нелинейную структуру, она не может быть задана с помощью класса Sequential и вместо этого задаётся с использованием функционального API Keras на основании классов Input и Model. В функциональном API каждый слой модели может быть вызван как функция, аргументом которой является вход с предыдущего слоя. С помощью переменной feature в модели создаётся структура сверточной сети, которая затем формируется в 2 отдельные модели x1_net и x2_net с входами x1 и x2 соответственно, которые используют одну и ту же конфигурацию слоёв с общими весами. Результирующая сеть объединяет результаты работы параллельных моделей в слое Subtract, который вычисляет разность выходных векторов дублированной модели. Полученная разность дополнительно подаётся на сверточный слой и слой субдискретизации, результат работы которых подаётся на полносвязный слой из 512 нейронов с функцией активации ReLU и, впоследствии, на 1 выходной сигмоидальный нейрон, который формирует вывод модели.

```
from tensorflow.keras import layers, losses, metrics, optimizers
from tensorflow.keras.models import Model

x1 = layers.Input(shape=(width, height, depth))
x2 = layers.Input(shape=(width, height, depth))

# параллельная модель
inputs = layers.Input(shape=(width, height, depth))
feature = layers.Conv2D(32, 3, activation='relu')(inputs)
feature = layers.MaxPooling2D(2)(feature)
feature = layers.Conv2D(64, 3, activation='relu')(feature)
feature = layers.MaxPooling2D(2)(feature)
feature = layers.Conv2D(128, 3, activation='relu')(feature)
feature = layers.MaxPooling2D(2)(feature)
feature_model = Model(inputs=inputs, outputs=feature)

x1_net = feature_model(x1)
x2_net = feature_model(x2)

net = layers.Subtract()([x1_net, x2_net])
net = layers.Conv2D(128, 3, activation='relu')(net)
net = layers.MaxPooling2D(2)(net)
net = layers.Flatten()(net)
net = layers.Dense(512, activation='relu')(net)
```

```
    net = layers.Dense(1, activation='sigmoid')(net)

    classifier = Model(inputs=[x1, x2], outputs=net)
    classifier.compile(loss='binary_crossentropy',
optimizer=optimizers.RMSprop(), metrics=['accuracy'])
    classifier.summary()
```

После создания модели необходимо ее обучить с использованием функции fit на датасете, сформированном из пар изображений. Для проверки корректности работы модели можно подавать на вход любые 2 изображения с подписями – подлинную подпись и образец, с которым нужно её сравнить, и проанализировать выход сети. При этом на выходе сети будет 1 число – результат последнего слоя. При анализе большого числа экземпляров можно подобрать корректное пороговое значение для отнесения результата к одному или другому классу.

Существует архитектура сверточных нейронных сетей, называемая «сиамской сетью» (Siamese), которая основана на схожей идее – использовании двух копий одной и той же сверточной подсети с общими весами для формирования разности изображений. Архитектуру такой сети, а также особенности её обучения, рекомендуется изучить самостоятельно.

Использование сверточных слоёв может быть оптимизировано вычислениями на видеопроцессоре. В частности, TensorFlow предоставляет аппаратную поддержку работы с видеокартами NVIDIA. Если на вашем компьютере имеется видеопроцессор NVIDIA с поддержкой Cuda, можно установить соответствующие библиотеки и драйверы, как описано в инструкции: https://www.tensorflow.org/install/gpu.

**Задание к лабораторной 4.**

1. Создайте и обучите нейронную сеть для идентификации по отпечатку пальцев.
2. Создайте и обучите нейронную сеть для проверки подлинности произвольной рукописной подписи.
3. Сделайте образец своей подписи и попросите кого-нибудь (из одногруппников или родственников) сделать ложный ее образец.
4. Подайте полученные образцы на вход и проанализируйте результаты.
5. Найдите образцы дополнительные образцы подписей в интернете и на основании них подберите правильное пороговое значение для выхода сети.
6. Представьте ваше исследование в виде Блокнота JupyterLab, содержащего все пункты задания.