УДК 004.93

ПОИСК И РАСПОЗНАВАНИЕ ЛИЦ НА ИСТОРИЧЕСКИХ ФОТОГРАФИЯХ

Викторов Иван Юрьевич[1], Ершов Николай Михайлович[2]

[1]Студент;
Государственный университет «Дубна»;
141980, Московская область, г. Дубна, ул. Университетская, 19;
e-mail: ivor108@gmail.com.

[2]Доцент;
Государственный университет «Дубна»;
141980, Московская область, г. Дубна, ул. Университетская, 19;
e-mail: ershovnm@gmail.com.

Работа посвящена разработке системы распознавания и верификации лиц на исторических (дореволюционных и довоенных) фотографиях. В работе проводится обзор существующих методов распознавания лиц. Рассматривается модель сиамской нейронной сети в качестве основной архитектурой для разработки системы. Описывается программная реализация модели сиамской нейронной сети на языке программирования Python с использованием фреймворков TensorFlow и Keras. Рассматриваются три варианта обучения модели сиамской нейронной сети. Были проведена процедура подготовки данных. Из набора данных был собран размеченный набор обучающих примеров. Модель была протестирована и улучшена с помощью метода дообучения нейронных сетей. Максимальной точностью, которой удалось добиться в ходе выполнения работы, является 94%. Методы распознавания графических образов можно использовать для решения генеалогических задач, таких как поиск родственных связей, поиск информации о живых и погибших родственниках и т.д. Для жителей России и стран СНГ такие задачи актуальны связи с историческими событиями последних ста лет. Также приводятся результаты численных экспериментов по исследованию и сравнению предложенных методов.

Ключевые слова: распознавание лиц, верификация лиц, нейронные сети, сиамские нейронные сети.

Для цитирования:
Викторов И. Ю., Ершов Н. М. Поиск и распознавание лиц на исторических фотографиях // Системный анализ в науке и образовании: сетевое научное издание. 2021. № 2. С. 43–55. URL : http://sanse.ru/download/435.

FACE SEARCH AND RECOGNITION IN HISTORICAL PHOTOS

Viktorov Ivan Yu.[1], Ershov Nikolay M.[2]

[1]Student;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
e-mail: ivor108@gmail.com.

[2]Assistant professor;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
e-mail: ershovnm@gmail.com.

The work is devoted to the development of a system for recognizing and verifying faces in historical (pre-revolutionary and pre-war) photographs. The paper reviews the existing methods of face recognition. The Siamese neural network model is considered as the main architecture for system development. The software implementation of the Siamese neural network model in the Python programming language using the TensorFlow and Keras frameworks is described. Three variants of training the Siamese neural network model are considered. The data preparation procedure was carried out. A marked-up set of training examples was

collected from the data set. The model was tested and improved using the method of additional training of neural networks. The maximum accuracy that was achieved during the execution of the work is 94%. The methods of recognizing graphic images can be used to solve genealogical problems, such as searching for family ties, searching for information about living and deceased relatives, etc. For residents of Russia and the CIS countries, such tasks are relevant due to the historical events of the last hundred years. The results of numerical experiments on the study and comparison of the proposed methods are also presented.

Keywords: face recognition, face verification, neural networks, Siamese neural networks.

For citation:
Viktorov I., Ershov N. Search and recognition of the faces in historical photographs. System Analysis in Science and Education, 2021;(2):43–55(In Russ). Available from: http://sanse.ru/download/435.

Введение

В настоящее время тенденция использования алгоритмов глубокого обучения для анализа данных растёт. Это связано с увеличением производительности вычислительных систем и увеличением объёмов доступной информации в сети интернет. Поэтому, использование алгоритмов глубокого обучения в прикладных задачах сегодня как никогда актуально.

Методы распознавания графических образов можно использовать для решения генеалогических задач, таких как поиск родственных связей, поиск информации о живых и погибших родственниках и т.д. Для жителей России и стран СНГ такие задачи актуальны связи с историческими событиями последних ста лет.

В связи с широким распространением социальных сетей и других интернет-ресурсов, поиск данных для обучения моделей не является большой проблемой. Несмотря на это, при поиске старинных фотографий в необходимом качестве и в достаточном объёме могут возникнуть трудности.

Сегодня самой эффективной моделью для решения задач распознавания графических образов являются свёрточные нейронные сети. Данная модель позволяет распознавать образы с большой точностью. Свёрточные нейронные являются составляющей архитектуры сиамских нейронных сетей. Приведённая архитектура используется при ограниченном наборе данных в задачах распознавания и верификации.

Обзор метода распознавания лиц

Один из самых популярных методов распознавания лиц – это мультиклассовая классификация (см. рис. 1). Данный метод имеет следующие недостатки. Первым недостатком является распознавание большого количества классов, из-за этой особенности нейронная сеть сильно увеличивается в размерах и процесс обучения становится долгим. Следующим недостатком является большое количество данных для каждого класса. При решении поставленной задачи нет возможности сбора большого количества данных для каждого класса. Третьим недостатком является необходимость переобучения модели при добавлении новых классов. Данный недостаток помещает реализовать универсальную и гибкую систему.

Проанализировав особенности и недостатки данного метода был выбран метод сиамских нейронных сетей.

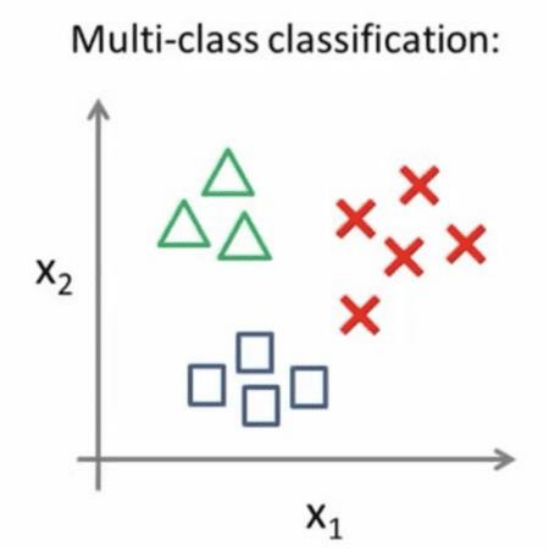

Рис. 1. Мультиклассовый классификатор

Сиамские нейронные сети

Сиамские нейронные сети – разновидность архитектур нейронных сетей. Их главная особенность состоит в том, что сиамские нейронные сети включают в себя две или более идентичные свёрточные нейронные сети.

Дочерние сети имеют одинаковую структуру, параметры и веса. Чаще всего сиамская нейронная сеть включает в себя две дочерние сети (см. рис. 2). Любые обновления параметров отражаются в обеих подсетях, то есть, если обновляются веса в одной сети, то веса и в другой тоже обновляются.

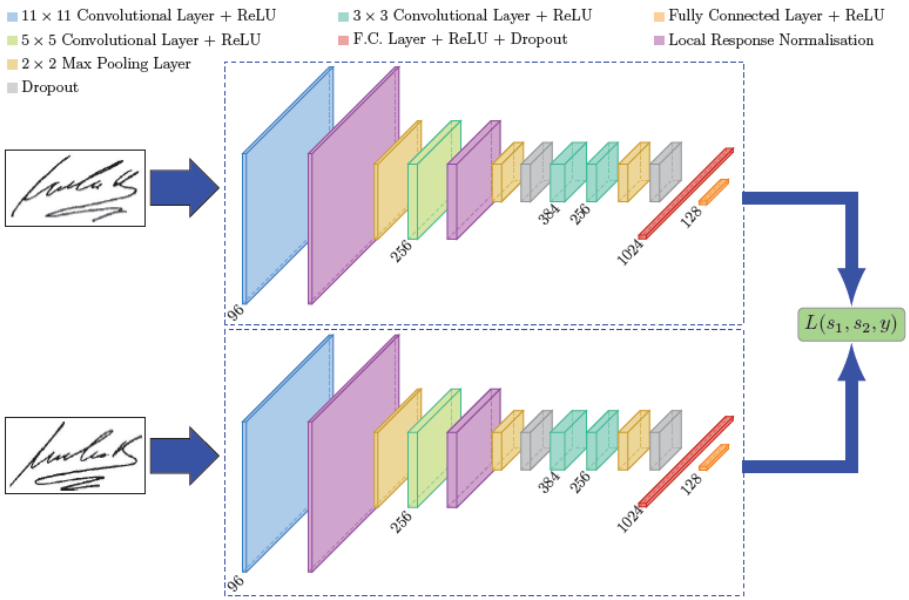

Рис. 2. Пример сиамской нейронной сети

На вход модели подаётся пара изображений. Каждая дочерняя сеть выделяет вектор свойств для каждого изображения. Далее вычисляется расстояние между векторами. Расстояние определяет сходство данных изображения. Такая архитектура нейронной сети имеет большое преимущество перед традиционными способами классификации изображений. Модель не требует повторного обучения при добавлении новых классов и объём данных для обучения, может быть, небольшим.

Для вычисления расстояния между векторами свойств используется формула Евклидова расстояния:

$$d(x, y) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2} \quad (1)$$

где x, y – вектора свойств, N – размерность вектора.

Для реализации сиамской нейронной сети можно использовать различные функции потерь. Одна из них – бинарная кросс энтропия, представленная в следующей формуле:

$$E = \frac{-1}{N}\sum_{i=1}^{N}[d_i log y_i + (1 - d_i) log(1 - y_i)]. \quad (2)$$

Для решения некоторых задач используются более сложные функции потерь такие как триплетная и контрастная функции потерь. Так как в данной работе модель сиамской нейронной сети будет использоваться для сопоставления двух фотографий, задачу можно свести к бинарной классификации. В таком случае, необходимо использовать в качестве функции потерь бинарную кросс энтропию [1].

Описание информационной системы

Для обучения модели, на входи системы подаётся набор изображений произвольного размера. Далее, с помощью системы распознавания лиц, на изображениях выделяются лица и обрезается лишние участки. Полученные изображении переводятся в градации серого и приводятся к одному размеру. Перед подачей на нейронную сеть, набор данных нормируется (см. рис. 3).

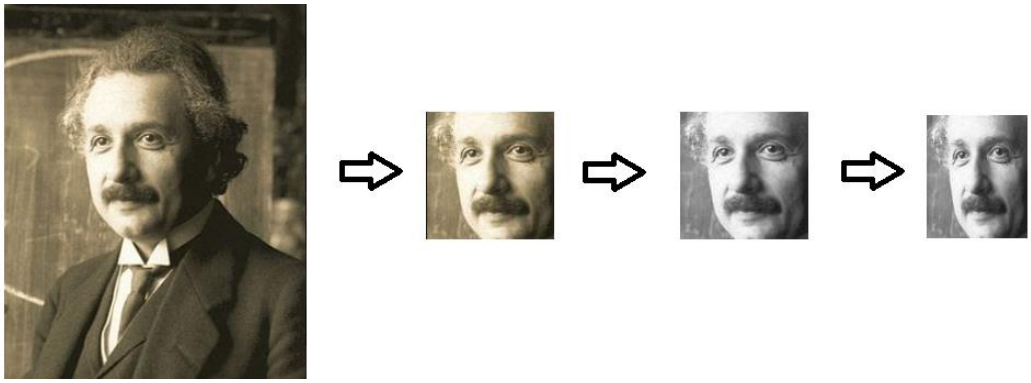

Рис. 3. Пример обработки данных

Важной особенностью модели сиамской нейронной сети, является то, что на вход необходимо подавать пары изображений двух классов: позитивные и негативные. Позитивные пары – две разные фотографии одного человека, негативные – две фотографии разных людей (см. рис. 4). Процесс обработки изображений и составления пар будет подробнее описан в разделе программной реализации.

Рис. 4. Пример пар изображений

После этапа обучения, на выходе будет получена модель нейронной сети, готовая для дальнейшего применения и тестирования.

Для использования системы в режиме предсказания входные данные подаются в таком же формате, как и при обучении. Выходными данными будет массив коэффициентов схожести двух изображения. Коэффициент схожести представляет из себя число с плавающей точкой в диапазоне от 0 до 1. Поэтому коэффициент можно интерпретировать как вероятность.

Подготовка данных

Ниже представлена реализация функция *crop_faces*. Функция получает на вход изображение, а на выходе возвращает список обрезанных лиц, распознанных на изображении (см. рис. 5).

```
def crop_faces(img):
  face_loc = fr.face_locations(img)
  pil_image = PIL.Image.fromarray(img)
  croped_images = []
  for face_location in face_loc:
    top, right, bottom, left = face_location
    croped = pil_image.crop((left, top, right, bottom))
    croped_images.append(croped)
  return croped_images
```

Рис. 5. Функция для обрезки лиц на изображении

После определения функции необходимо применить её ко всему набору данных. Для этого произведём обход всех папок и преобразуем каждое изображение в каждой папке (см. рис. 6).

```
path = '/content/drive/MyDrive/Colab Notebooks/Faces/data/'
uncrop = []

for i in range(1, 21):
  for j in range(1, 11):
    img = PIL.Image.open(path + str(i) + '/' + str(j) + '.jpg')
    crop_img = crop_faces(fr.load_image_file(path + str(i) + '/' + str(j) + '.jpg'))
    if len(crop_img) != 0:
      crop_img[0].save('/content/drive/MyDrive/Colab Notebooks/Faces/data_cropped/'
                       + str(i) + '/' + str(j) + '.jpg')
    else:
      img.save('/content/drive/MyDrive/Colab Notebooks/Faces/data_cropped/'
               + str(i) + '/' + str(j) + '.jpg')
      uncrop.append(str(i) +  str(j))
```

Рис. 6. Обработка всех изображений с помощью функции crop_faces

В переменной *path* хранится путь к директории с необработанными данными. В строках 9 и 12 происходит запись обрезанной фотографии в необходимую директорию.

Далее произведём преобразование изображений к градациям серого и приведение к необходимому размеру (см. рис. 7).

```
1 path = '/content/drive/MyDrive/Colab Notebooks/Faces/data_cropped/'
2
3 for i in range(1, 21):
4   for j in range(1, 11):
5     img = cv2.imread(path + str(i) + '/' + str(j) + '.jpg')
6     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7     cv2.imwrite('/content/drive/MyDrive/Colab Notebooks/Faces/data_grey/'
8                 + str(i) + '/' + str(j) + '.jpg', img)
```

```
1 path = '/content/drive/MyDrive/Colab Notebooks/Faces/data_grey/'
2
3 for i in range(1, 21):
4   for j in range(1, 11):
5     img = cv2.imread(path + str(i) + '/' + str(j) + '.jpg')
6     img = cv2.resize(img, (92, 112), interpolation = cv2.INTER_AREA)
7     cv2.imwrite('/content/drive/MyDrive/Colab Notebooks/Faces/data_resized/'
8                 + str(i) + '/' + str(j) + '.jpg', img)
```

Рис. 7. Преобразование изображений

Теперь необходимо сформировать общую выборку данных в виде пар изображений. Как было упомянуто ранее, пары должны быть двух видов, позитивные и негативные. Для составления такого набора пар были составлены списки содержащие номера изображений, которые нужно соединить в пары (см. рис. 8). Список *same* содержит пары различных фотографий одного человека, *diff* – фотографии разных людей.

```
6  same = []
7  diff = []
8
9  for i in range(1, 21):
10   for j in range(1, 11):
11     for k in range(j + 1, 11):
12       same.append(str(i)+ " " + str(j) + " " + str(k))
13
14 same += same[:]
15
16 for i in range(1, 21):
17   for j in range(i + 1, 21):
18     for k in range(1, 11):
19       for m in range(1, 11):
20         diff.append(str(i) + " " + str(k) + " " + str(j)+ " " + str(m))
```

Рис. 8. Составление списков пар

Далее при составлении пар, список *diff* будет перемешен и из него случайным образом будут взяты столько же пар сколько и в списке *same*. Это необходимо для того, чтобы в обучающей выборке пар обоих видов было равное количество.

Далее, исходя из номеров изображений из списков *same* и *diff* формируем *NumPy* массив с позитивными и негативными парами (см. рис. 9) [2].

```
count = 0
x_geuine_pair = np.zeros([total_sample_size, 2, 1, dim1, dim2]) # 2 is for pairs
y_genuine = np.zeros([total_sample_size, 1])
for i in same:
  ind = same[count].split(' ')
  img1 = read_image('/content/drive/MyDrive/Colab Notebooks/Faces/data_resized/'
                    + str(ind[0]) + '/' + str(ind[1]) + '.jpg')
  img2 = read_image('/content/drive/MyDrive/Colab Notebooks/Faces/data_resized/'
                    + str(ind[0]) + '/' + str(ind[2]) + '.jpg')
  img1 = img1[::size, ::size]
  img2 = img2[::size, ::size]
  x_geuine_pair[count, 0, 0, :, :] = img1
  x_geuine_pair[count, 1, 0, :, :] = img2
  y_genuine[count] = 1
  count += 1

count = 0
x_imposite_pair = np.zeros([total_sample_size, 2, 1, dim1, dim2])
y_imposite = np.zeros([total_sample_size, 1])
for i in diff[:1800]:

  ind = diff[count].split(' ')
  img1 = read_image('/content/drive/MyDrive/Colab Notebooks/Faces/data_resized/'
                    + str(ind[0]) + '/' + str(ind[1]) + '.jpg')
  img2 = read_image('/content/drive/MyDrive/Colab Notebooks/Faces/data_resized/'
                    + str(ind[2]) + '/' + str(ind[3]) + '.jpg')
  img1 = img1[::size, ::size]
  img2 = img2[::size, ::size]
  x_imposite_pair[count, 0, 0, :, :] = img1
  x_imposite_pair[count, 1, 0, :, :] = img2
  y_imposite[count] = 0
  count += 1
```

Рис. 9. Составление позитивных пар

Во время составления пар формируем массив меток. У позитивных пар будет метка 1, у негативных 0. После формирования пар и меток объединяем массив позитивных пар с массивом негативных пар. Ту же операцию проделываем с массивами меток и нормируем данные путём деления каждого значения массива пар на 255 (см. рис. 10).

```
X = np.concatenate([x_geuine_pair, x_imposite_pair], axis=0)/255
Y = np.concatenate([y_genuine, y_imposite], axis=0)
```

Рис. 10. Объединение массивов пар и меток

Последним этапом подготовки данных – это разделение общей выборки на тренировочную выборку и тестовую. Для разделения используем функцию *train_test_split* из модуля *Sklearn*. Выполним разбиение и выведем на экран форматы общей выборки, обучающей и тестовой (см. рис. 11).

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=.10)
print(X.shape)
print(Y.shape)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3600, 2, 1, 56, 46)
(3600, 1)
(3240, 2, 1, 56, 46)
(360, 2, 1, 56, 46)
(3240, 1)
(360, 1)
```

Рис. 11. Разбиение общей выборки

Таким образом процесс обработки данных завершён. Были сформированы обучающая и тестовая выборка. Выборки состоят из пар позитивных и негативных пар изображений и метками [3].

Программная реализация модели

После того как данные были подготовлены, прейдём к непосредственной реализации модели сиамской нейронной сети. Для этого была реализована функция построения дочерней нейронной сети. Функция принимает два аргумента, форма входного массива и размерность выходного массива. В строке 2 инициализируем тензор необходимой формы (см. рис. 12). В нашем случае форма будет (1, 56, 46).

```
1  def build_siamese_model(inputShape, embeddingDim=48):
2    inputs = Input(inputShape)
3
4    x = Conv2D(64, (3, 3), padding="same", activation="relu", data_format= 'channels_first')(inputs)
5    x = MaxPooling2D(pool_size=(2, 2), data_format= 'channels_first')(x)
6    x = Dropout(0.1)(x)
7
8    x = Conv2D(64, (3, 3), padding="same", activation="relu", data_format= 'channels_first')(x)
9    x = MaxPooling2D(pool_size=2,  data_format= 'channels_first')(x)
10   x = Dropout(0.1)(x)
11
12   pooledOutput = GlobalAveragePooling2D()(x)
13   outputs = Dense(embeddingDim)(pooledOutput)
14
15   model = Model(inputs, outputs)
16
17   return model
```

Рис. 12. Функция построения дочерней нейронной сети

В строке 4 объявляется первый сверхточный слой с помощью функции *Conv2d*. В качестве параметров задаётся количество фильтров, размерность ядра свёртки. Присваиваем параметру *padding* значение “*same*”, это означает что будет создана рамка для сохранения исходной размерности после проведения операции свёртки. Зададим функцию активации *relu.* Параметр *data_format* определяет на какой позиции во входном тензоре находится канал цвета. В нашем случае канал цвета находится на первой позиции. В строке 5 добавляется операция *MaxPooling* определяем размер ядра и позицию канала цвета. В строке 6 добавляем операцию Dropout и задаёт коэффициент. В нашем случае коэффициент достаточно мал, т.к. во время обучения не появлялся эффект переобучения. Далее повторим все предыдущие действия. В строке 12 добавим ещё одну операцию *Pooling*, но на этот раз *GloablAverage-Pooling*. С помощью этой операции произойдёт преобразование тензора к векторной форме. В 13 строке зададим выходной полносвязный слой [4].

Далее создаётся модель по указанным настройкам и подаётся на выход функции. В результате инициализируется следующая архитектура (см. рис. 13).

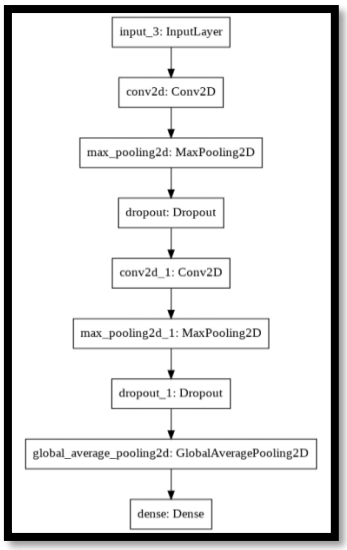

Рис. 13. Архитектура дочерней нейронной сети

Перед тем как инициализировать непосредственно саму модель сиамской нейронной сети необходимо определить функцию Евклидова расстояния. Для проведения математических расчётов был использован модуль *backend* который был переименован в *K* (см. рис. 14).

```
1 def euclidean_distance(vectors):
2   (featsA, featsB) = vectors
3   sumSquared = K.sum(K.square(featsA - featsB), axis=1, keepdims=True)
4   return K.sqrt(K.maximum(sumSquared, K.epsilon()))
```

Рис. 14. Функция расчёта Евклидова расстояния

Теперь всё готово для реализации основной модели. В строках 1-5 создадим дочернюю нейронную сеть *featureExtractor* и создадим две её копии *featsA* и *featsB*. В строке 7 определим *Lambda* слой, вычисляющий Евклидово расстояние. На вход слою подаём две копии дочерней сети. В строке 8 определяем выходной полносвязный слой с функцией активации *sigmoid*. И в строке 9 создаём модель (см. рис. 15).

```
1 imgA = Input(shape=IMG_SHAPE)
2 imgB = Input(shape=IMG_SHAPE)
3 featureExtractor = build_siamese_model(IMG_SHAPE)
4 featsA = featureExtractor(imgA)
5 featsB = featureExtractor(imgB)
6
7 distance = Lambda(euclidean_distance)([featsA, featsB])
8 outputs = Dense(1, activation="sigmoid")(distance)
9 model = Model(inputs=[imgA, imgB], outputs=outputs)
```

Рис. 15. Построение модели сиамской нейронной сети

В итоге была получена следующая архитектура (см. рис. 16).

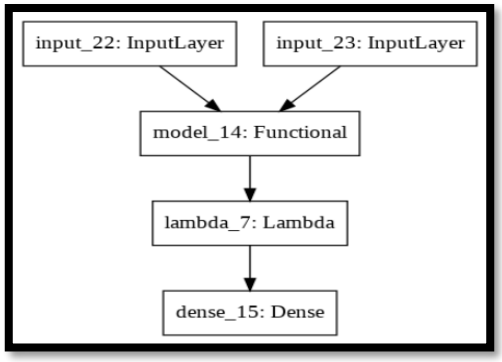

Рис. 16. Архитектура сиамской нейронной сети

Далее необходимо скомпилировать разработанную модель и запустить процесс обучения. Для компиляции вызовем метод compile указав бинарную кросс энтропию как функцию потерь, оптимизатор *adam* и метрику *accurancy*. Вызовем метод *fit* для запуска процесса обучения. Передадим тренировочную выборку. Зададим процент разбиения тренировочной выборки на вариационную, в данном случае задаётся 25%. Параметры *bath_size* и *epochs* определяют размер пакета данных и количество эпох обучения (см. рис. 17) [5].

```
1 model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

```
1 history = model.fit(
2   [x_train[:, 0], x_train[:, 1]], y_train,
3   validation_split=.25,
4   batch_size=64,
5   epochs=300)
```

Рис. 17. Компиляция и обучение модели

Во время обучения промежуточные результаты сохранялись в переменную *history*. Построим графики потерь и точности для тренировочной и результирующей выборок (см. рис. 18).

Рис. 18. Графики функции потерь и точности во время обучения сиамской нейронной сети

В итоге была получена точность 90% и потери 25%. Точность предсказаний модели можно увеличить с помощью методов дообучения которые будут реализованы в следующем разделе.

Дообучение – метод увеличения точности модели. Идея метода заключается в следующем. Модель обучается на созданном другими людьми наборе данных. Существует большее количество уже готовых наборов данных собранных специально для обучения нейронных сетей [6]. Далее нижние слои обученной модели замораживаются. Модель снова обучается, только уже на данных собранными нами для решения конкретной задачи. Таким образом, слои выделяющие общие признаки не изменяются, а слои выделяющие более узкие признаки настраиваются под наши данные.

Для реализации данного метода будет использоваться набор данных *AT&T Database of Faces*. Обучим модель, разработанную в прошлом разделе на этом наборе данных. После обучения были получены следующие результаты (см. рис. 19).

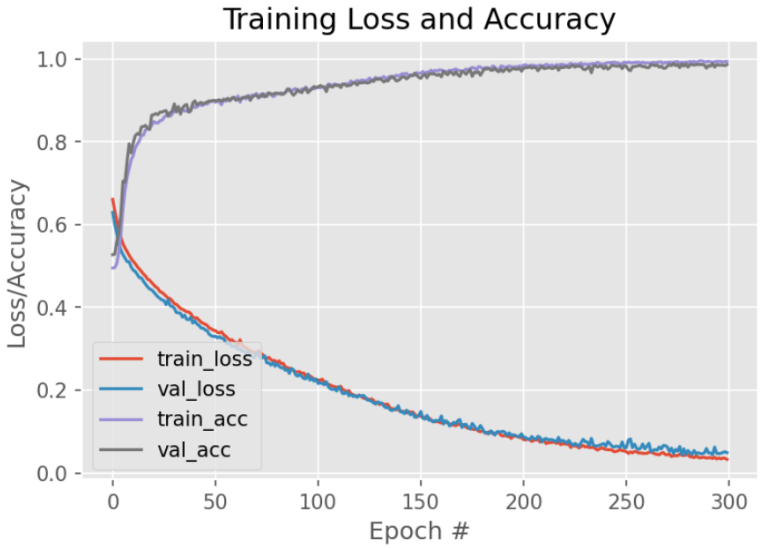

Рис. 19. Графики функции потерь и точности во время обучения на AT&T Database of Faces

При обучении на заранее подготовленных данных точность модели намного больше, чем в предыдущем варианте, но специально подготовленные данные часто отличаются от реальных данных, поэтому необходимо произвести дообучение полученной модели. Для пройдём циклом по всем слоям и заморозим все кроме слоя dense_1. Этот слой является выходным слоем модели (см. рис. 20).

```
for layer in model.layers:
  if layer.name == 'dense_1':
    layer.trainable = True
  else:
    layer.trainable = False
```

Рис. 20. Заморозка слоёв нейронной сети

После заморозки слоев необходимо запустить процесс обучения для наших данных. После обучения был получен следующий результат (см. рис. 21).

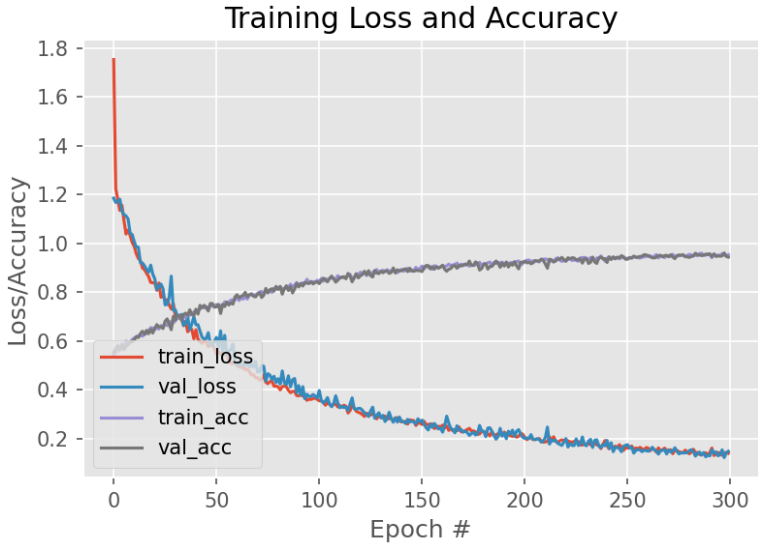

Рис. 21. Графики функции потерь и точности во время дообучения модели

Точность полученной модели составляет 94% и потери 15%. Точность модели была увеличена. Таким образом было получено три различных модели, в следующем разделе будет проведен сравнительный анализ этих моделей и проверка работы моделей на практике.

Численное исследование

В предыдущем разделе была реализована модель сиамской нейронной сети. Три экземпляра модели были обучены на различных данных и различными способами. В этой главе будет проведено сравнение трёх реализация и проверена работа моделей на практике. Первый экземпляр модели был обучен на данных представляющий из себя набор фотографий исторических личностей 20-го века. Для удобства, назовём данный экземпляр *my_model*. Второй экземпляр модели был обучен на готовом наборе данных *AT&T Database of Faces* [7]. При тестировании, данной модели будем называть её *clear_model*. При реализации третьего экземпляра был использован метод увеличения точности модели – дообучение. Данный метод называется *fine tuning* поэтому назовём третий экземпляр *ft_model*.

Загрузим тестовые данные и проведём оценку точностей моделей на наборе фотографий исторических личностей 20-го века. Полученные результаты представим в виде таблицы (см. Таблица 1).

Таблица 1. Результаты тестирования

Название модели	Точность	Потери
my_model	0,9333	0,2062
clear_model	0,4806	4,6157
ft_model	0,9444	0,1577

Необходимо обратить внимание, что тест для модели clear_model проводился на наборе фотографий исторических личностей. Поэтому были получены результаты, сильно отличающиеся от результатов, полученных после обучения.

Построим столбчатую диаграмму для визуализации полученных данных. Красным цветом изображены показатели модели *my_model*. Синим *clear_model*, а фиолетовым *ft_model* (см. рис. 22).

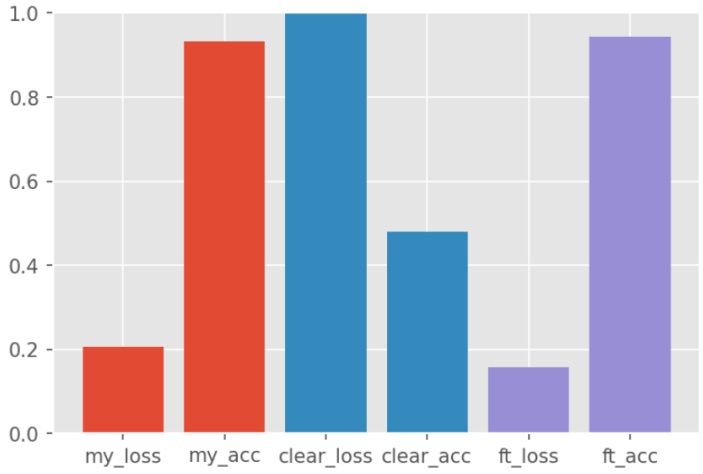

Рис. 22. Точность и потери, полученные при тестировании моделей

В ходе тестирования было произведено сравнение точности моделей. Было выяснено, что моделью с самой большой точностью является *ft_model* т.е. дообученная модель. Следовательно, эту модель необходимо использовать для решения практических задач.

После тестирования проведём анализ предсказаний модели. Рассмотрим количество верных предсказаний и ошибочных. Так же проверим для каких пар изображений чаще всего модели допускают ошибки. Все предсказания будут производиться на наборе данных фотографий исторических личностей 20-го века. Сначала рассмотрим общее количество верных и ошибочных предсказаний (см. Таблица 2).

Таблица 2. Верные и не верные предсказания моделей

Название модели	Общее количество фотографий	Верные предсказания	Не верные предсказания	Точность
my_model	360	336	24	0,933
clear_model	360	173	187	0,481
ft_model	360	340	20	0,944

Проверим удачные предсказания всех трёх моделей (см. Таблица 3).

Таблица 3. Верные предсказания моделей

Название модели	Общее количество пар	Верные предсказания		
		Общее количество верных	Позитивные пары	Негативные пары
my_model	360	336	192	144
clear_model	360	173	31	142
ft_model	360	340	175	165

Теперь рассмотрим ошибочные предсказания моделей (см. Таблица 4).

Таблица 4. Не верные предсказания моделей

Название модели	Общее количество пар	Не верные предсказания		
		Общее количество не верных	Позитивные пары	Негативные пары
my_model	360	24	0	24
clear_model	360	187	161	26
ft_model	360	20	4	16

По полученным данным видно, что модели *my_model* и *ft_model* примерно одинаково верно предсказывают позитивные и негативные пары. При предсказании негативных пар ошибки делаются чаще. Дообученная модель *ft_model* точнее модели без дообучения *my_model*, но модель без дообучения не сделала ни одной ошибки при предсказании позитивных пар, в отличии от дообученной модели.

Заключение

В ходе проведения работы были исследованы методы распознавания и верификации лиц, более детально была изучена модель сиамской нейронной сети. Была реализована модель сиамской нейронной сети и обучена на различных наборах данных, для этого был собран набор фотографий исторических личностей 20-го века. Были проведена процедура подготовки данных. Из набора данных был собран размеченный набор обуча.щих примеров. Модель была протестирована и улучшена с помощью метода дообучения нейронных сетей. Максимальной точностью, которой удалось добиться в ходе выполнения работы, является 94%.

Список литературы

1. Sean Benhur J. A friendly introduction to Siamese Networks // Torwads Data Science. 2010 September 2. URL: https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942 (дата обращения: 15.05.2021).
2. NumPy Documentation. NumPy, 2008–2020. URL: https://numpy.org/doc/ (дата обращения: 20.05.2021).
3. Князев Н. Распознавание лиц с помощью сиамских сетей // Хабр. 28 августа 2019. URL: https://habr.com/ru/company/jetinfosystems/blog/465279/ (дата обращения: 25.04.2021).
4. Keras: библиотека глубокого обучение на Python. URL: https://ru-keras.com/home/ (дата обращения: 19.05.2021).
5. Rosebrock A. Siamese networks with Keras, TensorFlow, and Deep Learning // PySmageSearch. 2020 November 30. URL: https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/ (дата обращения: 20.04.2021).
6. Шолле Ф. Глубокое обучение на Python. — СПб. : Питер, 2018. — 400 с.
7. AT&T The Database of Faces // Kaggle. URL: https://www.kaggle.com/kasikrit/att-database-of-faces (дата обращения: 20.05.2021).