**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**
**МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ**

**Отчет по курсовой работе**
**Дисциплины «Программная инженерия»**
**По теме: «Реализация приложения для решения задачи классификации данных на основе нейронных сетей»**

**Выполнил студент группы А-13а-19**
**Горина Анастасия**
**Научный руководитель: Варшавский П.Р.**
**Преподаватель:**
**Маран М.М.**

**Москва 2022**

# Оглавление

## Введение

В данной работе решается задача классификации людей на изображении по признаку соблюдения масочного режима. Целью работы является реализация приложения, способного по загруженной фотографии людей во время пандемии обнаружить людей, и определить соблюдение масочного режима.

Мотивацией к выбору данной задачи послужила эпидемиологическая ситуация, в которой находятся большинство стран мира на протяжении последних двух лет в связи с пандемией SARS-CoV-2. Данные условия требуют мер социальных ограничений, в том числе соблюдения масочного режима при нахождении в общественных местах. Контроль данных ограничений может требовать большого количества людей, что заставляет задумываться о возможности оптимизации данного процесса с помощью доступных технологий.

## Задача классификации

С увеличением объёма информации её анализ значительно усложняется, что приводит к невозможности её анализа человеком и необходимости использования интеллектуального анализа данных (ИАД) (в англоязычной литературе используется термин Data mining). ИАД заключается в исследовании и автоматическом обнаружении в данных знаний и закономерностей, не известных ранее, нетривиальных, практически полезных и доступных для интерпретации человеком (ссылка на анализ данных и процессов). Для построения моделей ИАД существуют различные подходы, к которым относятся, например, дерево решений, прецедентные системы и искусственные нейронные сети. Кроме того, несмотря на то, что статистические методы требуют изначальных предположений, они в некоторых случаях так же применяются среди методов ИАД.

- Дерево решений – это метод, который позволяет представить набор решающих правил в виде иерархической структуры (Quinlan, 1986). Основными элементами дерева решений являются узлы, которые

определяют условие, и листья, которые представляют собой множества, соответствующие или нет условию. Процесс построения дерева решений заключается в разбиении решающего множества на подмножества использую решающие правила.

- Прецедентные системы (Yuan Guo, 2011), они же Case-Based Reasoning – это метод построения рассуждений, основанных на подобных ситуациях. Для ИАД этот подход реализуется как использование предыдущего опыта для решения новых задач. Для автоматизации решения проблем с помощью данного подхода необходима накопленная база прецедентов.
- Нейронные сети – математическая модель, основанная на модели биологического нейрона. При объединении искусственных нейронов в достаточно большую систему, она способна обучиться решать сложные задачи. В процессе обучения нейронная сеть способна выявлять неочевидные зависимости в данных, поэтому она является одним из ключевых методов ИАД.

ИАД включает в себя шесть наиболее распространенных задач (Usama Fayyad, 1996):

1. Выявление аномалий.
2. Поиск ассоциативных правил.
3. Задача кластеризация.
4. Задача классификации.
5. Регрессионный анализ.
6. Автоматическое реферирование.

Наиболее интересной для нас в рамках данной работы является задача классификации. Задача классификации заключается в определении класса объекта по его характеристикам. Эта задача является достаточно простой для человека, однако сложной для компьютера из-за наличия скрытых признаков которые он не в состоянии обработать (Honglak Lee, 2009).

## Нейронная сеть

Искусственная нейронная сеть (ИНС) –это система, состоящая из связанных между собой искусственных нейронов, способная решать сложные задачи, которые требуют аналитических вычислений подобных тем, которые делает человеческий мозг.

- Искусственный нейрон – это упрощенная модель биологического нейрона. Искусственный нейрон представляет собой нелинейную функцию (3) от комбинации (2) поступающих на него сигналов (1 –

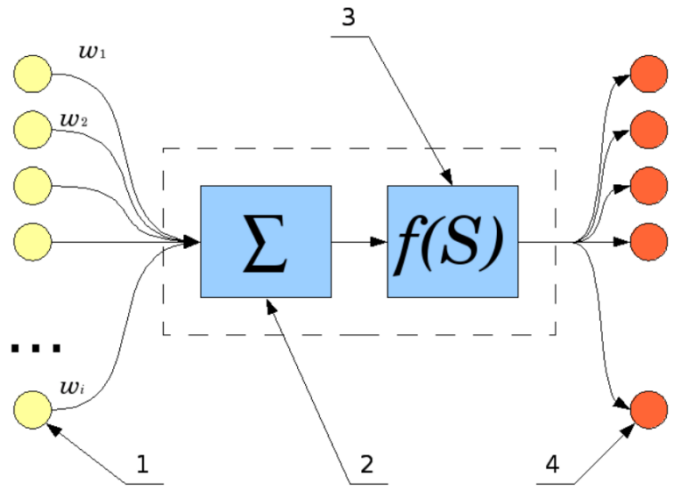


*Рисунок 1 - Модель искусственного нейрона*

нейроны предыдущего слоя и веса входных сигналов). Упомянутая функция является функцией активации. Результат посылается на выход нейрона и передается на вход нейронов следующего слоя (4).

В настоящее время нейронные сети активно используются для выполнения задач, связанных с распознаванием и классификацией образов, принятием решений, прогнозированием, оптимизацией, а также в задачах анализа данных.

Одной из наиболее востребованных сфер является распознавание изображений. Задача распознавания заключается в автоматическом выделении признаков в данных и на основе их определения положения и класса объектов. Чаще всего задачи распознавания решаются с использованием сверточных нейронных сетей.

Особенностью сверточных нейронных сетей является использование сравнительно небольшой матрицы (ядра свертки) к входному набору параметров, вместо большого числа отдельных параметров. Параметры ядер свертки на разных слоях ИНС устанавливаются в процессе обучения. После сверточного слоя как правило следует слой пулинга, который уплотняет изображение и позволяет перейти к наиболее абстрактному представлению признаков. Данные преобразования позволяют модели лучше работать при небольшом изменении положения признаков на изображении.

Однако модель, до того, как она была обучена, не имеет практического применения. Обучение модели нейронной сети заключается в процессе подбора значений коэффициентов связи между нейронами, весов. Подбор значений осуществляется, основываясь на обработке данных сетью и оценке значений ошибки.

Качество обучения нейронной сети значительно зависит от качества используемого набора. Это формирует определенные требования к выборке, несоблюдение которых ведет к неработоспособности модели при анализе новых данных, не встречавшихся ранее в обучающей выборке.

К таким требованиям относятся (А. А. Барсегян, 2009):

1. Представительность – выборка содержит данные всех классов.
2. Достаточность – большее количество данных позволяет построить более точную модель.
3. Неповторяемость – отсутствие одинаковых данных.
4. Различимость – отсутствие слишком схожих изображений разных классов.
5. Отсутствие аномалий.
6. Сбалансированность – равенство числа экземпляров в каждом классе.

## Подготовка данных для обучения ИНС

Первым этапом работы послужит анализ выбранного набора и преобразование его к формату, который позволит эффективно обучать модель.

Для построения решения был выбран набор бытовых фотографий с людьми, соблюдающими масочный режим и несоблюдающими, а также неправильно носящими медицинскую маску. Поскольку была выбрана задача классификации, изначальный формат изображений не является подходящим для построения модели. Перед построением модели необходимо собрать новый набор, содержащий только изображения лиц и значения класса, к которому оно относится.

Начальные данные представлены в виде пар: каждому фото соответствует файл с описанием расположения лиц и класса, к которому относится лицо. Из описаний, содержащихся в XML файлах, получим информацию о каждом лице, подсчитаем количество изображений каждого класса, размеры изображений лиц.

Примеры начальных изображений представлены на рисунке 2.

*Рисунок 2 - Примеры начальных изображений*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 28 | 55 | 46 | 71 | 0 | maksssksksss737 | 400 | 226 | maksssksksss737.xml | maksssksksss737.png |
| 98 | 62 | 111 | 78 | 0 | maksssksksss737 | 400 | 226 | maksssksksss737.xml | maksssksksss737.png |
| 159 | 50 | 193 | 90 | 2 | maksssksksss737 | 400 | 226 | maksssksksss737.xml | maksssksksss737.png |
| 293 | 59 | 313 | 80 | 0 | maksssksksss737 | 400 | 226 | maksssksksss737.xml | maksssksksss737.png |
| 352 | 51 | 372 | 72 | 0 | maksssksksss737 | 400 | 226 | maksssksksss737.xml | maksssksksss737.png |
| 228 | 53 | 241 | 73 | 0 | maksssksksss737 | 400 | 226 | maksssksksss737.xml | maksssksksss737.png |
| 34 | 153 | 62 | 176 | 0 | maksssksksss410 | 400 | 267 | maksssksksss410.xml | maksssksksss410.png |
| 88 | 138 | 111 | 163 | 0 | maksssksksss410 | 400 | 267 | maksssksksss410.xml | maksssksksss410.png |
| 139 | 207 | 168 | 234 | 0 | maksssksksss410 | 400 | 267 | maksssksksss410.xml | maksssksksss410.png |
| 116 | 135 | 135 | 154 | 0 | maksssksksss410 | 400 | 267 | maksssksksss410.xml | maksssksksss410.png |

*Рисунок 3 - Примеры данных о лицах*

При подсчете числа лиц в каждом классе мы получили, что в классе 0 (маска надета) значительно больше изображений (3232), чем в оставшихся двух: нет маски – 717 изображений, неверно надетая маска – 123 изображения.

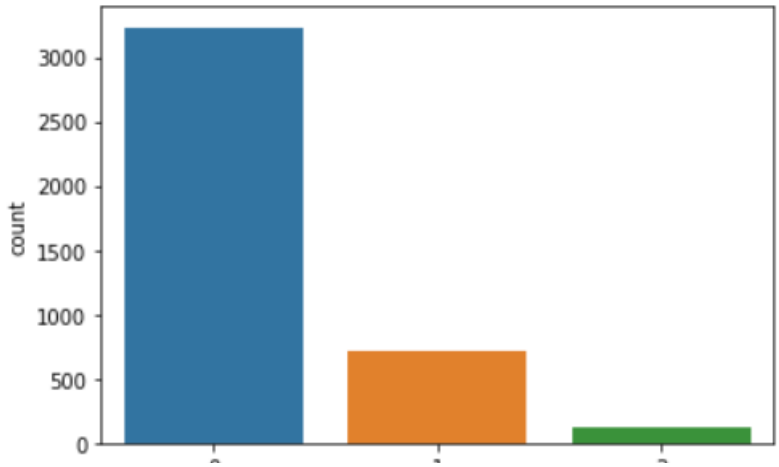


*Рисунок 4 - Изначальное распределение по классам*

Проанализируем размеры изображений, содержащих лица. Сохраним размеры и выведем их с помощью гистограммы, чтобы визуально продемонстрировать количество изображений каждого размера. Из полученных графиков можно заметить, что имеется большое число картинок слишком малого для анализа размера. Вероятно, данные изображения - это лица, находящиеся слишком далеко, однако для обучения они будут иметь слишком плохое качество. Будет целесообразным ограничить размер изображений, которые будут включены в итоговый набор.

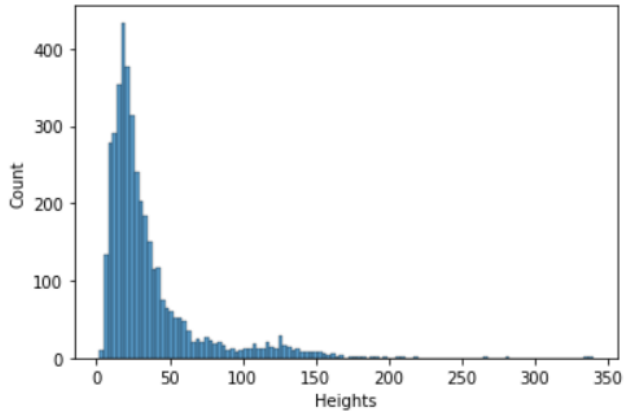


*Рисунок 6 - График распределения высоты изображений*

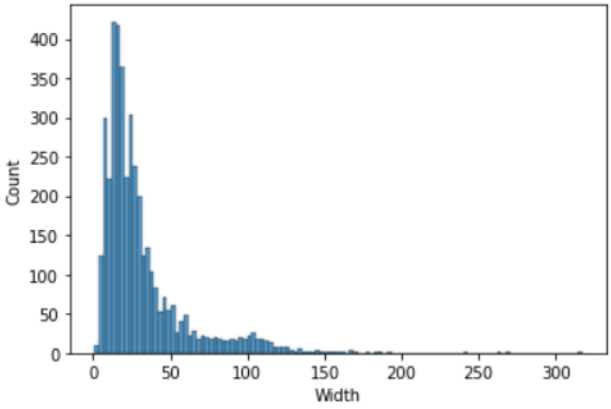


*Рисунок 5 - График распределения ширины изображений*

Выведем несколько лиц в изначальном качестве и без преобразований:

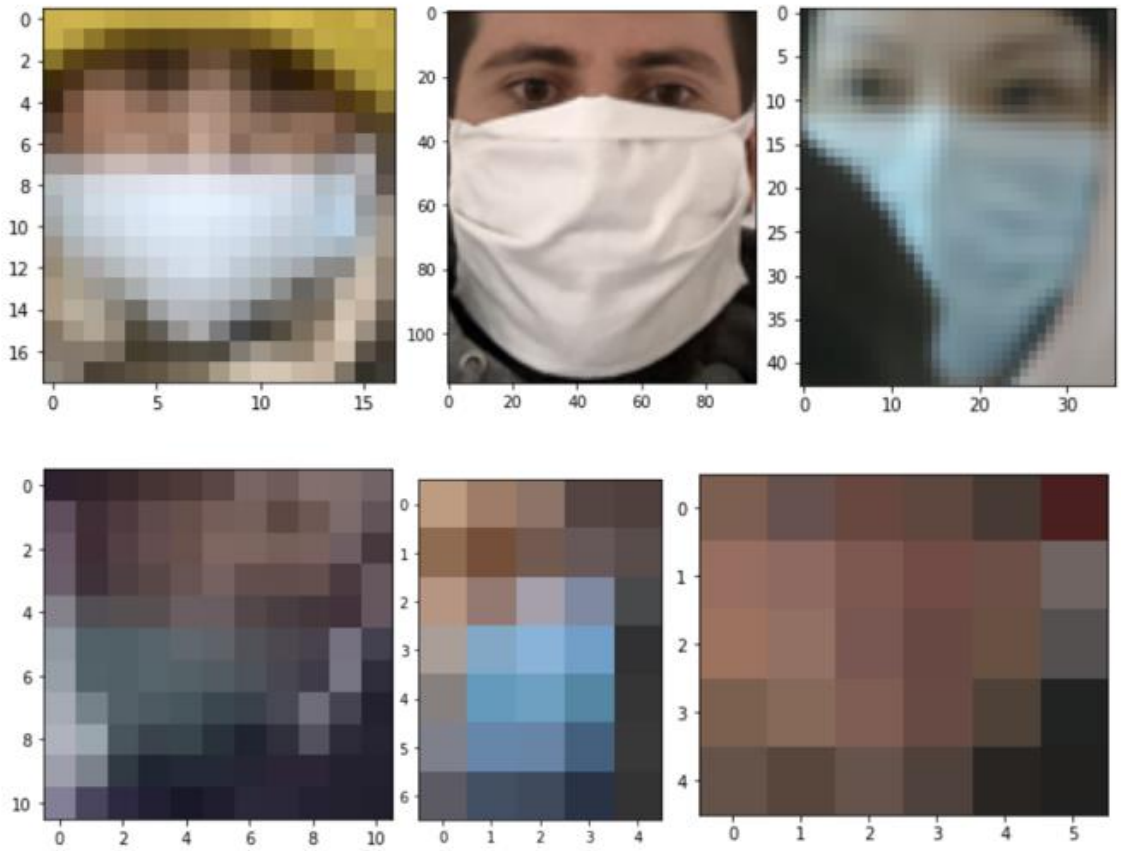

*Рисунок 7 - Примеры изначально полученных лиц*

Для ограничения размера выберем размер в 15 пикселей в ширину и длину, данные размерности всё ещё достаточно малы, однако уже не являются неразличимыми. Чтобы минимизировать преобразование размеров изображений, выберем средний размер как 40 на 40 пикселей, чтобы избежать значительного преобразования размеров и при этом иметь достаточное качество для выделения признаков.

Подсчитаем, какое количество изображений будет получено при применении вышеперечисленных ограничений: 2394 в первом классе и 444 во втором классе.

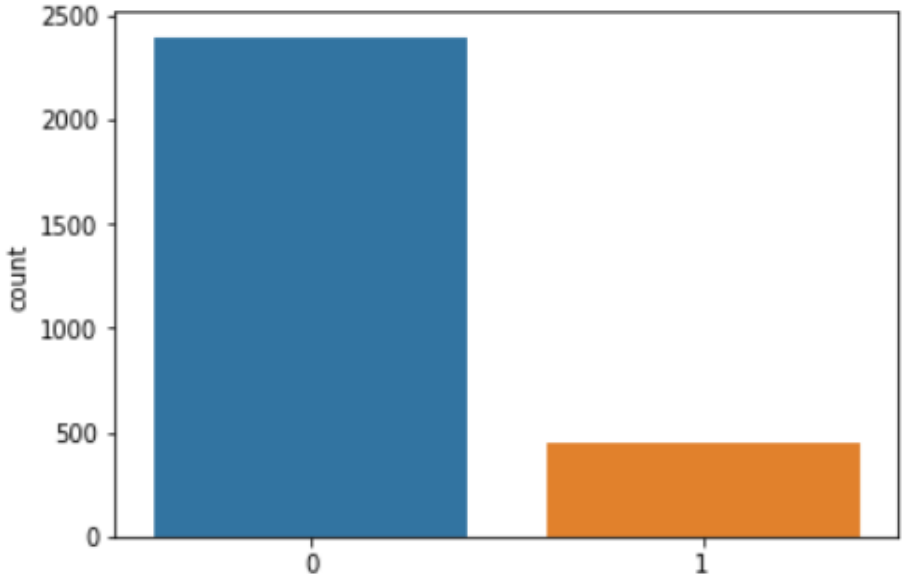


*Рисунок 8- График распределения по классам при ограничении размера изображений*

Используем дополнение множества с помощью библиотеки Keras (KerasAPIreference). Используем нейронную сеть с двумя слоями, один из которых случайно отражает изображение по горизонтали, а второй случайно поворачивает на малый угол. Для выравнивания числа изображений второго класса добавим для каждого изображения 4 его варианта, полученные с помощью упомянутой модели.

Реализация модели:

```
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal"),
  layers.RandomRotation(0.15),
])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
random_flip_1 (RandomFlip)   (40, 40, 3)               0
_________________________________________________________________
random_rotation_1 (RandomRot (40, 40, 3)               0
=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
None
```

*Рисунок 9 - Модель, используемая для дополнения набора*

Пример работы на одном из оригинальных изображений. Первое без изменений, следующие три получены как результат его обработки.

*Рисунок 10 - Пример дополнительных изображений*

Распределение по классам после дополнения данных: первый класс содержит 2394 изображений, второй 2220.

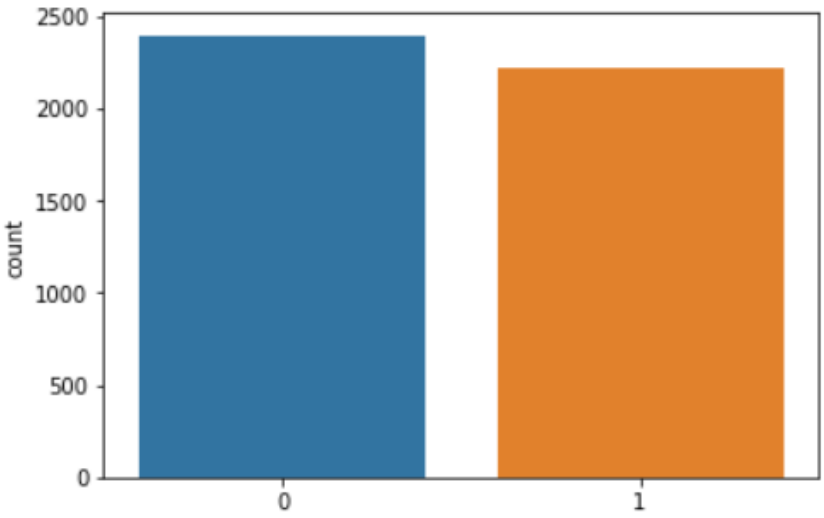


*Рисунок 11 - График распределения по классам в итоговом наборе*

Для формирования обучающего и тестового множеств полученный набор случайно перемешиваем и делим в пропорции 80% на обучающее множество и 20% на тестовое множество. В итоговом наборе имеем 3691 изображение в обучающем и 923 в тестовом множествах.

*Рисунок 12 - Примеры изображений обучающего и тестового множеств*

## Проектирование ИНС для решения выбранной задачи классификации

Основная задача заключается в построении модели классификации изображений по двум классам. Реализуем модель с помощью Python 3.9 (Python, 2021) с использованием библиотеки PyTorch v1.10.0 (pytorch). Разработку и обучение модели проведем с использованием облачного сервиса kaggle (Kaggle, 2021) в связи с большей эффективностью и поддержкой необходимых инструментов.

Для решения задачи будем использовать модель, содержащую два сверточных слоя, так как они выделяют абстрактные общие признаки изображений. Совместно с ними используем пакетную нормализацию.

- Пакетная нормализация (BatchNorm2D) дает два преимущества в процессе использования. Во-первых, мы получаем значение на слое в меньшем диапазоне, что увеличивает скорость обучения. Во-вторых, применение пакетной нормализации уменьшает величину ковариантного сдвига. Ковариантный сдвиг - это ситуация, в которой распределения значений признаков в обучающем и тестовом наборе имеют разные параметры. В нашем случае заметно, что количество изображений с масками светлого цвета значительно больше, что может привести к ковариантному сдвигу в виде неверного распознавания черных масок.

Так как мы имеем относительно небольшой набор, то предполагаем, что для обучения будет достаточно малого количества эпох.

Модель

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.conv3 = nn.Conv2d(32, 32, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.batchnorm32 = nn.BatchNorm2d(16)
        self.batchnorm64 = nn.BatchNorm2d(32)
        self.fc1 = nn.Linear(32*6*6, 64)
        self.fc2 = nn.Linear(64, 2)
        self.dropout = nn.Dropout(p = 0.3)
    def forward(self, x):
        x = self.conv1(x)
        x= self.batchnorm32(x)
        x = self.pool(F.relu(x))
        x = self.conv2(x)
        x= self.batchnorm64(x)
        x = self.pool(F.relu(x))
        x = F.relu(self.conv3(x))
        x = x.view(-1, 32*6*6)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

```
Model(
  (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (batchnorm32): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batchnorm64): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=1152, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=2, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
)
```

*Рисунок 13- Слои модели*

Полученные графики обучения говорят о достижении относительно высокой точности классификации. Для вычисления более точных данных о результатах обучения модели создадим classification report на основе тестового множества.

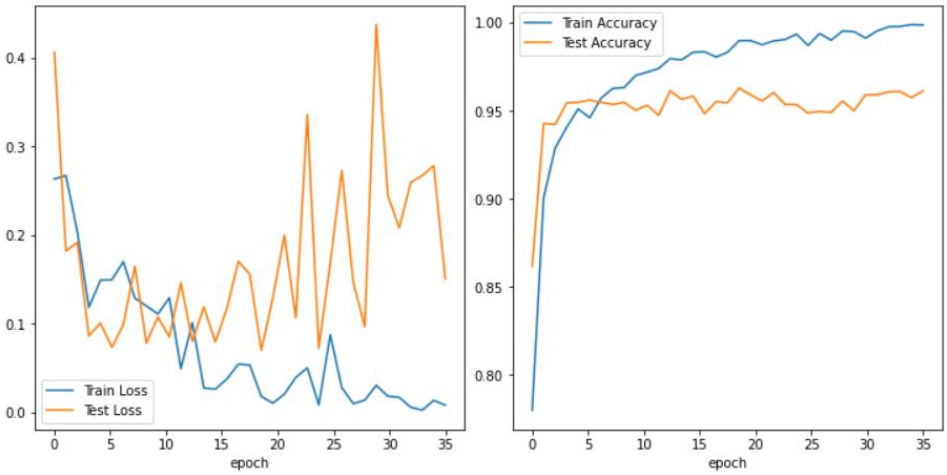


*Рисунок 14 - Графики обучения*

Исходя из подробного отчета, точность на тестовом наборе достигает 96%, что говорит о хороших результатах обучения и эффективности действий по дополнению исходных данных.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.98 | 0.95 | 0.96 | 510 |
| 1 | 0.94 | 0.97 | 0.96 | 413 |
| accuracy | | | 0.96 | 923 |
| macro avg | 0.96 | 0.96 | 0.96 | 923 |
| weighted avg | 0.96 | 0.96 | 0.96 | 923 |

*Рисунок 15 - Таблица classification_report*

Поскольку результаты обучения говорят об относительно высокой эффективности модели, сохраним её для дальнейшего использования приложением. В приложении понадобится воспользоваться обученной ранее моделью, однако в связи с ограниченным числом платформ, поддерживающих использование библиотеки Pytorch и некоторыми сложностями установки, встречающимися у многих пользователей, будет более выгодно перейти на использование библиотеки OpenCV v4.5 (OpenCV, 2021). Для этого потребуется сохранить модель в формате onnx, который упрощает перенос моделей между различными платформами.

- ONNX - Open Neural Network Exchange – это формат моделей с открытым исходным кодом. Данный формат задает расширенную модель графа вычислений, определения операторов и типов данных. Данный формат упрощает экспорт моделей для использования разными библиотеками.

Перевод к формату onnx требует дополнительно один раз запустить существующую модель, для чего выделим случайный пример изображения из тестового множества.

```
onnx_path = "./model.onnx"
model.eval()
rand_inp = torch.as_tensor(train[5][0].reshape(1, 1, 40, 40)).float()
torch.onnx.export(model, rand_inp, onnx_path, verbose=True)
```

## Описание реализации приложения

Для практического применения модели была реализована небольшая программа на языке Python 3.9 (Python, 2021), позволяющая достаточно просто выбирать изображение и наглядно выводить результат пользователю.

Для разработки оконного приложения были использованы инструменты из библиотеки tkinter (tkinter, 2021).

Так как подготовленная модель ИНС способна работать только с изображением лица, то первым этапом следует загрузить предварительно обученную модель для распознавания лиц на изображении. Для этого будем использовать доступную в OpenCV модель Haar Cascades (OpenVC).

- Haar Cascades – это алгоритм, разработанный в 2001, который строится на обнаружении специфических особенностей или «Хаар подобных» особенностей. Данные особенности представляют собой ядра свертки, по которым вычисляется разность в интенсивности цветов в разных областях, что дает признаки для выделения результатов анализа области.

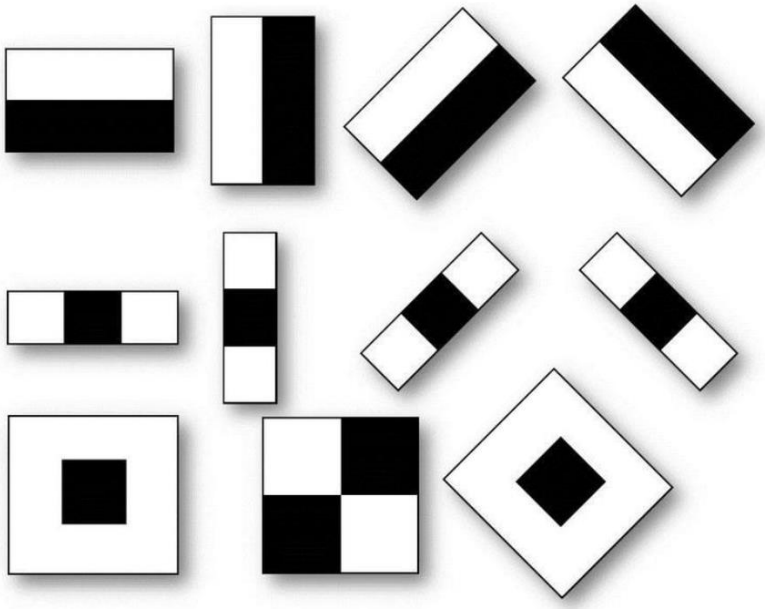

*Рисунок 16 - Примеры примитивов Хаара*

Загрузка модели для распознавания лиц:

```
detector = cv.CascadeClassifier(cv.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

Загрузка классификатора:

```
model_path = "D:\Programming\PyChanProjects\Mask test\model.onnx"
net = cv.dnn.readNetFromONNX(model_path)
```

Так как распознавание лиц может быть неточным, следует реализовать возможность добавить произвольную область к списку лиц.

## Тестирование реализованного приложения

Загрузим некоторые произвольные изображения, которые не встречались среди выбранного для обучения набора, обработаем их и оценим полученные результаты.


*Рисунок 17 - Тестовое изображение*

Распознанные лица:

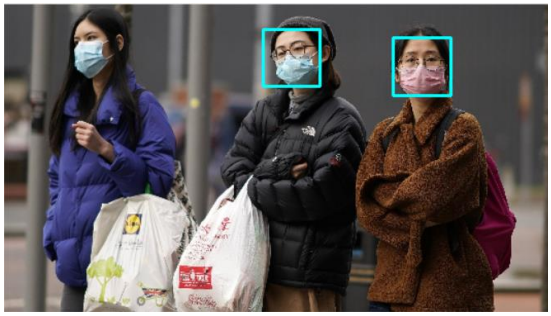

*Рисунок 18 - Распознанные лица на тестовом изображении*

Как видим из изображения-результата, наличие масок было распознано верно. Теперь добавим вручную положение третьего лица (рис. 19). Оно, хотя и находится под углом к нам, так же классифицируется верно.

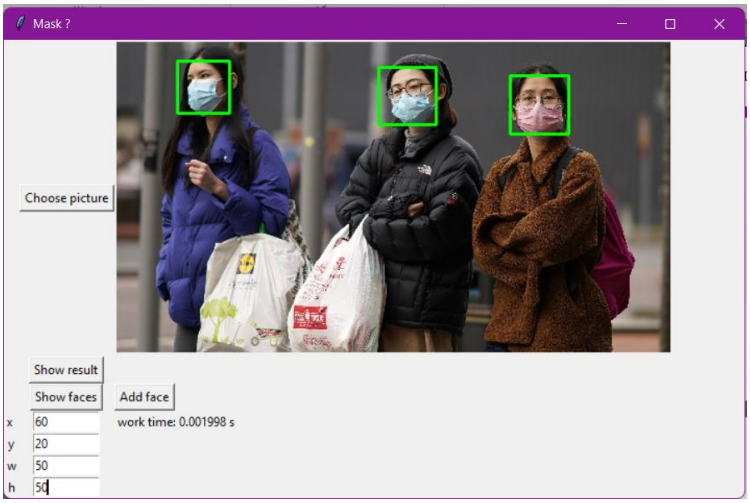


*Рисунок 19 - Результат классификации с дополненным списком лиц*

*Рисунок 20 - Распознанные лица*

Дополним список лиц и классифицируем их. В результате наблюдаем, что модель верно классифицирует в том числе и черные маски.

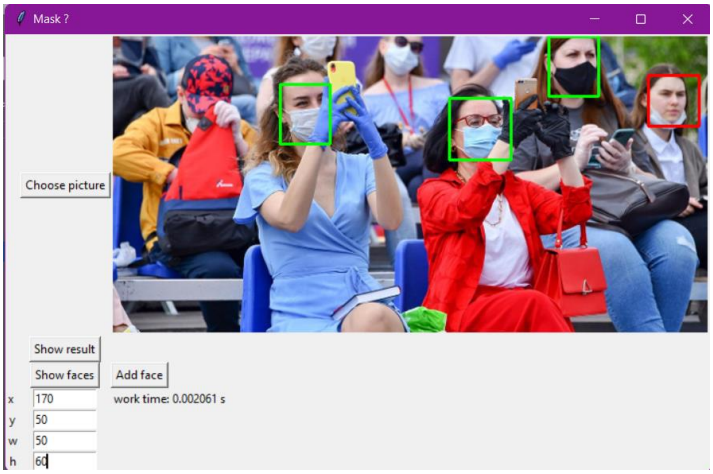


*Рисунок 21 - Результат классификации после дополнения списка лиц*

*Рисунок 22 - Распознанные лица на тестовом изображении*

Дополним список и классифицируем лица. Можно заметить, что на одном из изображений, классифицируемых как «без маски», маска есть, но надета неверно.

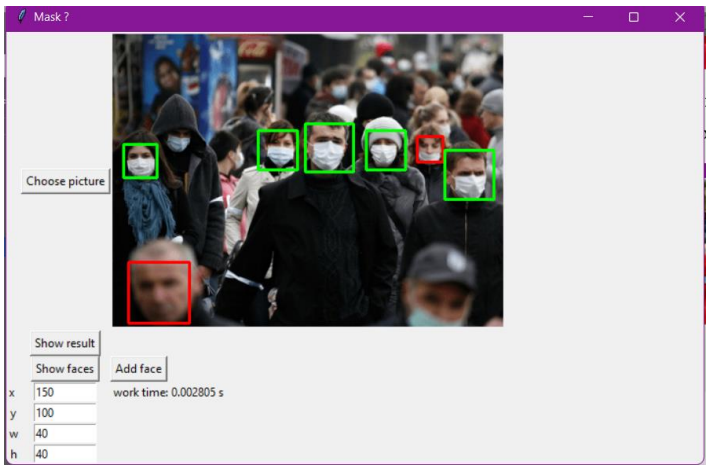


*Рисунок 23 - Результат классификации*

## Заключение

Для решения задачи классификации людей на фотографии по признаку соблюдения масочного режима было реализовано приложение, выполняющее задачу классификации с помощью обученной ИНС. Для обучения ИНС был выбран датасет, для которого был проведен анализ и преобразование исходных данных, и на их основе было получено обучающее множество, соответствующее поставленной задаче. Обученная на данном множестве модель достигает достаточно высокой точности при малом количестве нейронов и высокой скорости обучения.

Для обнаружения лиц на изображении использовалась сторонняя модель Haar Cascades, которая способна определять людей на фотографии при высоком качестве изображения. Однако, на многих изображениях имеются различные помехи для распознавания (угол наклона, низкое качество, размытие и др.) Поэтому для повышения качества обработки изображения был также реализован механизм ручного выделения лиц на фотографии.

В результате работы мы имеем программу, позволяющую достаточно быстро обрабатывать изображения и получать визуально информативный результат обработки.

## Список литературы


1. **Quinlan J. R.** Induction of decision trees [Книга]. - Boston : Kluwer Academic Publisher, 1986.
2. **Yuan Guo Jie Hu, Ying hong Peng** Research on CBR system based on data mining [Журнал] // Applied Soft Computing. - 2011 г.. - стр. 5006-5014.
3. **Usama Fayyad Gregory Piatetsky-Shapiro, Padhraic Smyth** From Data Mining to Knowledge Discovery in Databases [Журнал] // AI Magazine. - 1996 г.. - стр. 37-54.
4. **Honglak Lee Roger B. Grosse, Rajesh Ranganath, Andrew Y. Ng** Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations [Конференция] // Proceedings of the 26th Annual International Conference on Machine Learning, {ICML} 2009, Montreal, Quebec, Canada, June 14-18, 2009. - Quebec, Canada : ACM, 2009. - стр. 609--616.
5. **А. А. Барсегян М. С. Куприянов, И. И. Холод, М. Д. Тесс, С. И. Елизаров.** Анализ данных и процессов [Книга]. - Санкт-Петербург : БХВ-Петербург, 2009.
6. Keras API reference [В Интернете]. - https://keras.io/api/.
7. tkinter Documentation Python Standard Library [В Интернете]. - 2021 г.. - https://docs.python.org/3/library/tkinter.html.
8. Python [В Интернете]. - 2021 г.. - https://www.python.org.
9. PyTorch документация [В Интернете]. - https://pytorch.org/docs/stable/index.html.
10. Kaggle [В Интернете]. - 2021 г.. - https://www.kaggle.com.

11. OpenCV документация [В Интернете]. - 2021 г.. - https://docs.opencv.org/4.5.5/.

12. OpenVC: Cascade Classifier [В Интернете]. - https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

## Приложение 1: Код программы

```
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv
import pandas as pd
import tkinter as tk
from PIL import Image, ImageTk
from tkinter import filedialog, DISABLED, NORMAL
from tkinter.filedialog import askopenfilename, asksaveasfilename
import time

model_path = "D:\Programming\PyChanProjects\Mask test\model.onnx"
net = cv.dnn.readNetFromONNX(model_path)
detector = cv.CascadeClassifier(cv.data.haarcascades +
'haarcascade_frontalface_default.xml')

picture = []

class App:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Mask ?")
        # создаем рабочую область
        self.frame = tk.Frame(self.root)
        self.frame.grid()
        self.but_pick = tk.Button(self.frame, text="Choose picture", command =
self.get_picture).grid(row =1, column = 2)
        self.but_operate = tk.Button(self.frame, text="Show result",
command=self.process_picture)
        self.but_operate.grid(row=2, column=2)
        self.but_operate["state"] = DISABLED
        self.but_showf = tk.Button(self.frame, text= "Show faces", command =
self.show_faces)
        self.but_showf.grid(row = 3, column = 2)
        self.but_showf["state"] = DISABLED
```

```
        self.lb_x = tk.Label(self.frame, text = "x").grid(row = 4, column = 1,
sticky="w")
        self.lb_y = tk.Label(self.frame, text="y").grid(row=5, column=1)
        self.lb_w = tk.Label(self.frame, text="w").grid(row=6, column=1)
        self.lb_h = tk.Label(self.frame, text="h").grid(row=7, column=1)
        self.en_x = tk.Entry(self.frame, width = 10)
        self.en_x.grid(row = 4, column = 2)
        self.en_y = tk.Entry(self.frame, width  =10)
        self.en_y .grid(row = 5, column = 2)
        self.en_w = tk.Entry(self.frame, width =10)
        self.en_w.grid(row=6, column=2)
        self.en_h = tk.Entry(self.frame, width =10)
        self.en_h.grid(row=7, column=2)
        self.but_addf =  tk.Button(self.frame, text="Add face", command =
self.add_face)
        self.but_addf.grid(row = 3, column = 3, sticky="w")
        self.but_addf["state"] = DISABLED
        self.lb_time = tk.Label(self.frame, text = "no time")
        self.lb_time.grid(row = 4, column = 3, sticky="w")
        self.canvas = tk.Canvas(self.frame, height=300, width=600)
        self.canvas.grid(row =1, column = 3)
        self.root.mainloop()
    def add_face(self):
        if (self.h != '' and self.w != ''):
            self.faces = np.concatenate((self.faces, [np.array((self.x, self.y, self.w,
self.h))]))
    def process_picture(self):
        predictions = []
        info_sample = {'xm': [], 'xM': [], 'ym': [], 'yM': []}
        time_s = time.time()
        self.image = self.photo.copy()
        for result in self.faces:
            x, y, w, h = result
            face_cut = self.photo[y:y + h, x:x + w]
            info_sample['xm'] += [x]
            info_sample['xM'] += [x + w]
            info_sample['ym'] += [y]
            info_sample['yM'] += [y + h]
            res = cv.resize(face_cut, dsize=(40, 40), interpolation=cv.INTER_CUBIC)
            res = cv.cvtColor(res, cv.COLOR_BGR2GRAY)
            input_blob = cv.dnn.blobFromImage(
```

```
                image=res,
                size=(40, 40),  # img target size
                crop=False  # center crop
            )
            net.setInput(input_blob)
            preds = net.forward()
            prediction = np.array(preds)[0].argmax()
            predictions.append(prediction)
        boundaries = pd.DataFrame(info_sample)
        time_e = time.time()
        for i in range(len(self.faces)):
            color = (255, 255, 0)
            if (predictions[i] == 0):
                color = (0, 255, 0)
            else:
                color = (255, 0, 0)
            cv.rectangle(self.image, (boundaries['xm'].iloc[i], boundaries['ym'].iloc[i]),
                         (boundaries['xM'].iloc[i], boundaries['yM'].iloc[i]), color, 2)
        self.image = ImageTk.PhotoImage(image=Image.fromarray(self.image))
        self.c_image = self.canvas.create_image(0, 0, anchor='nw', image=self.image)
        self.canvas.grid(row=1, column=3)
        self.lb_time.config(text = "work time: " + str('%.6f' % (time_e - time_s)) + "
s")
    def get_picture(self):
        filepath = askopenfilename(filetypes=[("png pictures", "*.png")])
        if not filepath:
            return
        global picture
        self.photo = cv.imread(filepath)
        x = 300
        y = int(self.photo.shape[1] * x / self.photo.shape[0])
        self.photo = cv.resize(self.photo, dsize=(y, x),
interpolation=cv.INTER_CUBIC)
        self.photo = cv.cvtColor(self.photo, cv.COLOR_BGR2RGB)
        self.image = ImageTk.PhotoImage(image=Image.fromarray(self.photo))
        self.c_image = self.canvas.create_image(0, 0, anchor='nw', image=self.image)
        self.faces = detector.detectMultiScale(self.photo, scaleFactor=1.1,
minNeighbors=5,
                                               minSize=(5, 5))
        self.canvas.grid(row=1, column=3)
        self.but_operate["state"] = NORMAL
```

```
        self.but_showf["state"] = NORMAL
    def show_faces(self):
        self.image = self.photo.copy()
        color = (0, 255, 255)
        for (x, y, w, h) in self.faces:
            cv.rectangle(self.image, (x, y), (x + w, y+h), color, 2)
        self.x = self.en_x.get()
        self.y = self.en_y.get()
        self.w = self.en_w.get()
        self.h = self.en_h.get()
        if (self.w != '' and self.h != '' and self.x != '' and self.y != 0) :
            self.x = int(self.x)
            self.y = int(self.y)
            self.w = int(self.w)
            self.h = int(self.h)
            cv.rectangle(self.image, (self.x, self.y), (self.x + self.w, self.y + self.h),
color, 2)
        self.image = ImageTk.PhotoImage(image=Image.fromarray(self.image))
        self.c_image = self.canvas.create_image(0, 0, anchor='nw', image=self.image)
        self.canvas.grid(row=1, column=3)
        self.but_addf["state"] = NORMAL
app= App()
```