Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики, математики и электроники

Факультет информатики  
Кафедра технической кибернетики

**Отчёт о научно-исследовательской работе**

**бакалавра**

**Семестр 7**

Направление подготовки: 01.03.02 Прикладная математика и информатика:   
Профиль "Компьютерные науки"

Студент Гусев Илья Александрович группы 6308-010302D факультета информатики

Тема научно исследовательской работы: «Обнаружение следов применения методов реконструкции цифровых изображений».

Научный руководитель: Мясников Владислав Валерьевич д.ф – м.н, доцент, профессор каф. ГИ и ИБ Самарского национального исследовательского университета имени академика С.П. Королева

Научный руководитель

\_\_\_\_\_\_\_\_\_ Мясников А.А

*(подпись)*

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Студент

\_\_\_\_\_\_\_\_\_ Гусев И.А.

*(подпись)*

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

**Самара 2023РЕФЕРАТ**

**Отчет об учебной практике:** 24страницы, 15 рисунков, 12 источников.

РЕКОНСТРУКЦИЯ ИЗОБРАЖЕНИЙ, РЕТУШИРОВАНИЕ ИЗОБРАЖЕНИЙ, ЭКСПЕРТИЗА ИЗОБРАЖЕНИЙ, ОБНАРУЖЕНИЕ ОКРАШИВАНИЯ ИЗОБРАЖЕНИЙ

Объектом исследования являются алгоритмы обнаружения следов реконструкции цифровых изображений

Цель работы – создание БД ретушированных изображений, с использованием методов ретуширования, оценка качества работы готовых решений по обнаружению реконструкций на созданной БД. Разработка метода обнаружения ретуширования, с использованием стандартных методов машинного обучения. Оценка эффективности.

Содержание

[ВВЕДЕНИЕ 4](#_Toc105686634)

[1 Создание БД для хранения изображений 5](#_Toc105686635)

[1.1 Описание созданной БД для хранения изображений 5](#_Toc105686636)

[1.2 Проверка работоспособности готовых решений 7](#_Toc105686637)

[2 Теоретическое исследование методов разработки алгоритмов обнаружения ретуширования 13](#_Toc105686638)

[2.1 Обработка изображений 13](#_Toc105686639)

[2.2 Сверточные нейронные сети – СНС 15](#_Toc105686640)

[2.3 Обнаружение подделок цифровых изображений с использованием СНС 15](#_Toc105686641)

[ЗАКЛЮЧЕНИЕ 19](#_Toc105686642)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc105686643)

[ПРИЛОЖЕНИЯ 22](#_Toc105686644)

# ВВЕДЕНИЕ

В современном мире с развитием цифровых технологий подделка изображений является очень распространённым явлением, что ставит под сомнение надёжность цифровой информации.

Исключить вероятность подделок невозможно, но учёные уделяют пристальное внимание возможности отслеживания модификаций изображений. Эта проблема является крайне актуальной, так как модификации изображений могут кардинально искажать информацию, представленную на нём, что может ввести в заблуждение большие массы людей, также государственные органы.

В данной работе создаётся БД ретушированных изображений, берётся набор чистых изображений, вырезаются дыры в изображениях и методами ретуширования происходит заполнение этих дыр. Также сохраняем маски изображений. БД в последующем используется при обучении модели для распознавания ретуширования, исходный набор данных [1].

Приведены результаты работы готовых решений по обнаружению реконструкций на созданной БД и результаты разработанного алгоритма.

Проведено теоретическое исследование методов разработки алгоритмов обнаружения ретуширования.

## 1 Создание БД ретушированных изображений

## 1.1 Описание и тестирование примеров из созданной БД изображений

Чистые изображения взяты из исходного источника [1]. Сначала в этих изображениях вырезались дыры, в виде геометрических фигур, и сохранялись, также чёрно-белые маски изображений, смотрите рисунки 1 и 2. После дыры на изображениях заполнялись с использованием готовых методов ретуширования [2] и [3]. Далее закрашенные изображения тестировались на готовых методах обнаружения ретуширования

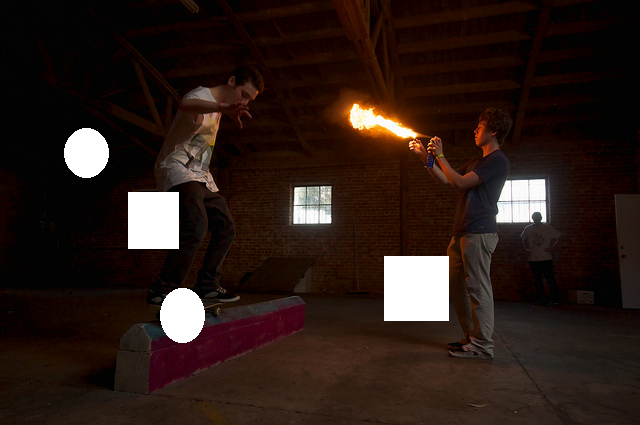


Рисунок 1 – Пример изображения с вырезанными областями

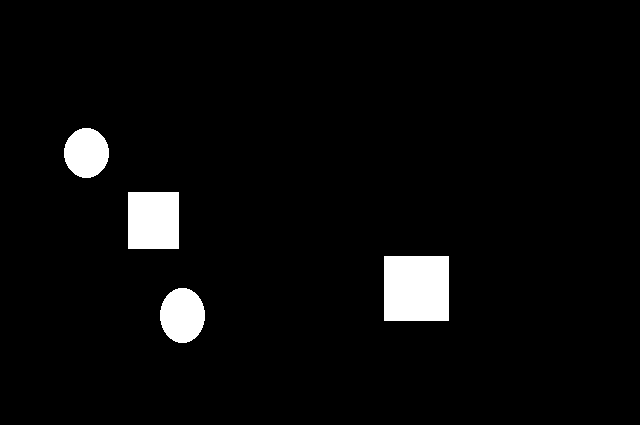


Рисунок 2 – Пример маски изображения с вырезанными областями

После дыры на изображениях заполнялись с использованием методов ретуширования [2]. Смотрите рисунок 3.



Рисунок 3 – Пример закрашивания дыр на изображении

Далее закрашенные изображения тестировались на готовых методах обнаружения ретуширования [4] смотрите рисунки 4 и 5.



Рисунок 4 – Пример обнаружения ретуширования изображения



Рисунок 5 – Пример обнаружения ретуширования изображения

## 1.2 Разработка метода обнаружения ретуширования с использованием стандартных методов машинного обучения

Для обучения моделей были использована информация о текстурные характеристики изображения, в закрашенных регионах и чистых. Задача состояла в бинарной классификации соответствующего фрагмента на изображении. С помощью методов ретуширования были раскрашены более 3000 изображений в разных областях, далее изображения нарезалось на фрагменты по пикселя, и после извлекались текстурные признаки Харалика, с помощью скрипта из источника [5]. В сравнении с другими текстурными признаками текстурные признаки Харалика показали лучшие результаты обучаемости моделей. Тренировочная выборка включает свыше 320 000 фрагментов изображений пикселей. В данной выборке примерно 75 000 фрагментов ретушированных областей, остальная часть – оригинальные фрагменты изображений. Решая задачу бинарной классификации, ретушированный фрагмент был обозначен классом – 1, оригинальный классом – 0.

В качестве фреймворка для машинного обучения использовалась библиотека машинного обучения Scikit-learn. В процессе обучения на тренировочной выборке были использованы стандартные деревья решений, также их ансамблевые варианты [6], было обучено 6 моделей:

– Модель, использующая стандартный классификатор деревьев решений;

– Модель, использующая классификатор случайного леса, работает по следующему принципу: выбираются различные подмножества признаков для повышения точности прогнозирования и контроля переобучения;

– Также схожий алгоритм – чрезвычайно рандомизированные деревья;

– Модель, использующая классификатор Бэггинга, обученный при участии классификатора, реализующего метод K-ближайших соседей;

– Также использованы ансамблевые методы, такие как AdaBoost, для стандартного классификатора деревьев решений по алгоритму SAMME и случайного леса по алгоритму SAMME.R [7].

В тестовом скрипте происходит разбиение изображений на фрагменты , далее извлечение следующих 13-ти текстурных признаков Харалика [8], таких как:

– второй угловой момент, который высок, когда изображение имеет очень схожие пиксели;

– функция контрастности, является мерой контраста или количества локальных вариаций, присутствующих на изображении;

– корреляция, измеряет линейную зависимость уровней серого соседних пикселей;

– сумма квадратов, дисперсия;

– обратный разностный момент, который высок, когда уровень серого однороден;

– сумма среднего, сумма отклонений, энтропия и суммарная энтропия;

– разностная дисперсия;

– информационные меры корреляции.

После извлечения текстурных признаков, они записываются в csv файл построчно, для каждого фрагмента. Далее, происходит получения прогноза от предварительно обученной модели по всем фрагментам изображения. Далее, фрагменты, отнесённые к классу – 1, полностью закрашиваются, для выделения предполагаемой области. Результирующее изображение собирается в целое из закрашенных и чистых фрагментов. Для просмотра результата смотрите рисунки 5 и 6. В таблице 1 приведена точность прогноза моделей для учебной и тестовой выборки.

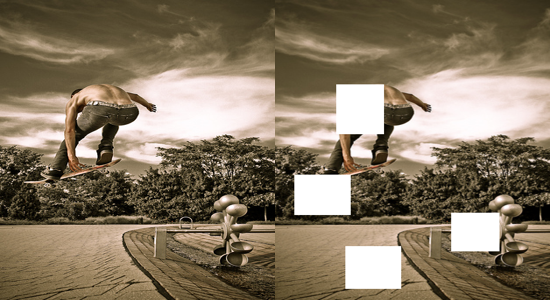


Рисунок 6 – Изображение с вырезанными фрагментами



Рисунок 7 – Закрашенное изображение



Рисунок 8 – Пример обнаружения ретуширования изображения с помощью обученной модели машинного обучения



Рисунок 9 – Пример обнаружения ретуширования изображения с помощью обученной модели машинного обучения



Рисунок 10 – Пример обнаружения ретуширования изображения с помощью обученной модели машинного обучения

Лучшие результаты обнаружения, на данных не участвующих при обучении, показали модели Ансамбль Light Gradient Boosted Machine и Random Forest (случайные леса). Результаты обнаружения на цельном изображении варьируются от изображения к изображению, текстурные признаки для этой задачи подходят плохо, что выражается в ложно положительных и ложно отрицательных предсказаниях. В целом, результаты обнаружения схожи с результатами алгоритма [4]. В таблице 1 приведена точность для каждой модели, достигнутая при обучении моделей на тренировочной выборке и на тестовой выборке – 15% от числа всей выборки.

Таблица 1 – Точность обнаружения обученных моделей на тестовом наборе данных

|  |  |  |
| --- | --- | --- |
| Алгоритм | Тренировочная выборка | Тестовая выборка |
| Стандартные деревья решений с AdaBoost | 99,9964% | 80,3722% |
| Бэггинг-классификатор | 82,3047% | 78,5400% |
| Ансамбль Light Gradient Boosted Machine | 99,9964% | 85,7436% |
| Случайные леса | 99,9949% | 86,6768% |
| Чрезвычайно рандомизированные деревья | 99,9964% | 86,6201% |
| Случайные леса с AdaBoost | 98,7788% | 80,0465% |

## 

## 2 Теоретическое исследование методов разработки алгоритмов обнаружения ретуширования с использование ИНС

## 2.1 Сверточные нейронные сети – СНС

**Сверточная нейронная сеть** (СНС) – это одна из прогрессивных форм **искусственных нейронных сетей (ИНС)**. СНС преуспели в решении задач анализа изображений. Принцип работы СНС состоит в том, что анализируемая область изображения проходит через череду свёрточных слоёв, каждый из которых является фильтром, отслеживающим определённые характеристики изображения, например фильтр, покажет, что на изображении кривая и т. д, после прохождения очередного слоя, на вход следующему, более глубокого уровня поступает, информация от предыдущего[9].

## 2.2 Обнаружение подделок цифровых изображений с использованием СНС

Для решения проблемы обнаружения ретуширования с использованием СНС, в работе [10] используется VGG16[9]

Архитектура VGG16 (смотрите рисунок 13):

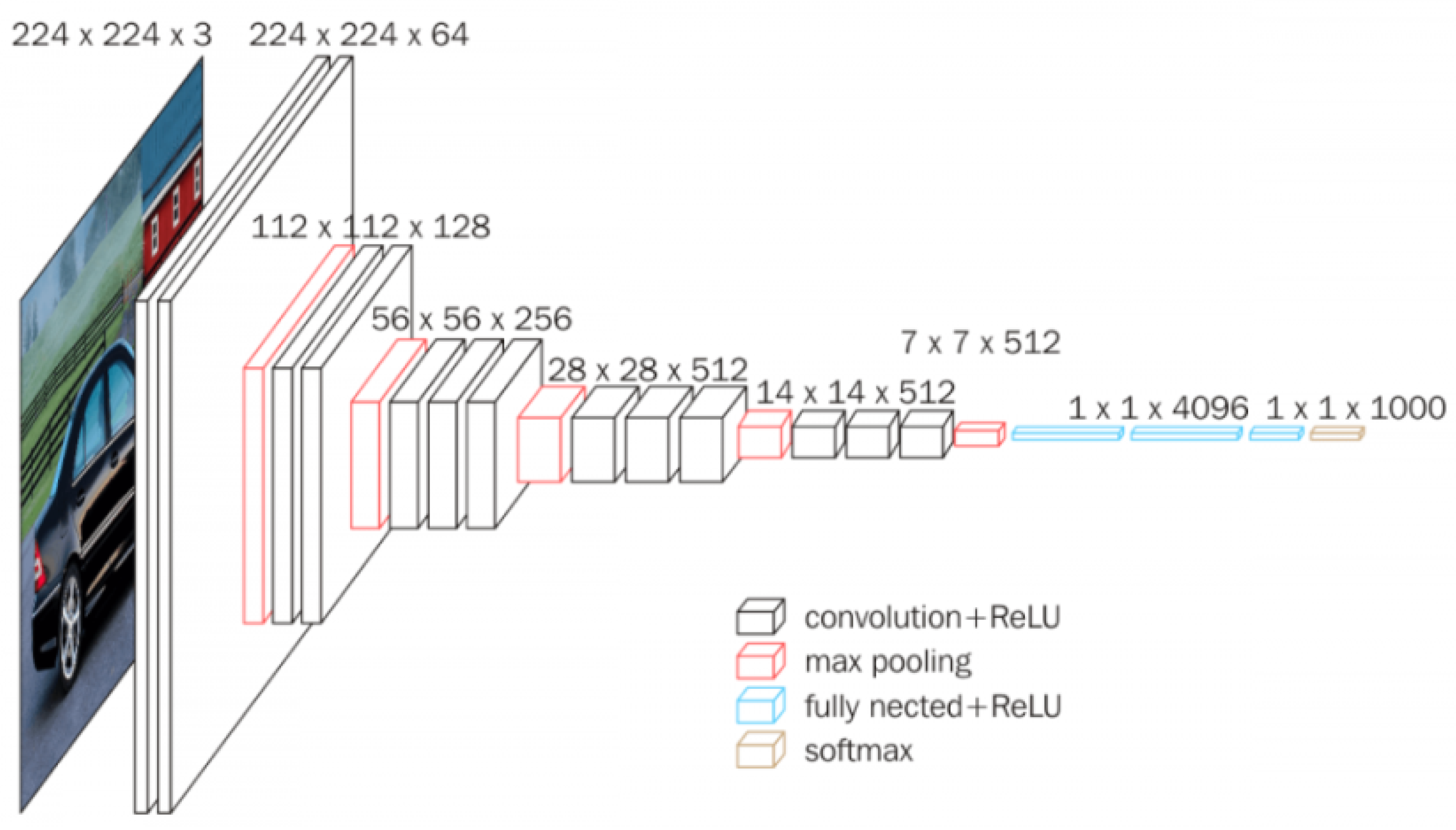


Рисунок 11 – Архитектура VGG16 [10]

Количество слоёв сети VGG16 равно 16. Входные данные – изображение , затем идут два сверточных слоя с функцией активации ReLU, каждый имеет по 64 фильтра, размеры ядер . Два подряд сверточных слоя позволяют анализировать изображение ядром , что повышает эффективность обучения нейронной сети. После идёт слой max pooling, который уменьшает линейные размеры карт признаков в два раза, количество фильтров же удваивается, таким образом, два следующих сверточных слоя имеют по 128 фильтров и так далее. В итоге получаем выходные данные , которые подаются на вход полно-связной нейронной сети, имеющей функцию активации ReLU, два первых слоя полносвязной нейронной сети имеют 4096 нейронов, кроме последнего слоя, имеющего 1000 нейронов, т.к сеть обучалась классифицировать данные на 1000 разных классов, далее идёт функция активации softmax [10].

В работе [1] происходит тренировка и тестирование моделей VGG16 с использованием графического интерфейса скриптом main\_final\_window.py.

В работе [10], рассмотрен способ совмещения VGG16 и UNet предварительно обученной на поддельных изображениях и их масках в чёрно-белых пикселях. Архитектура UNet (смотрите рисунок 14):

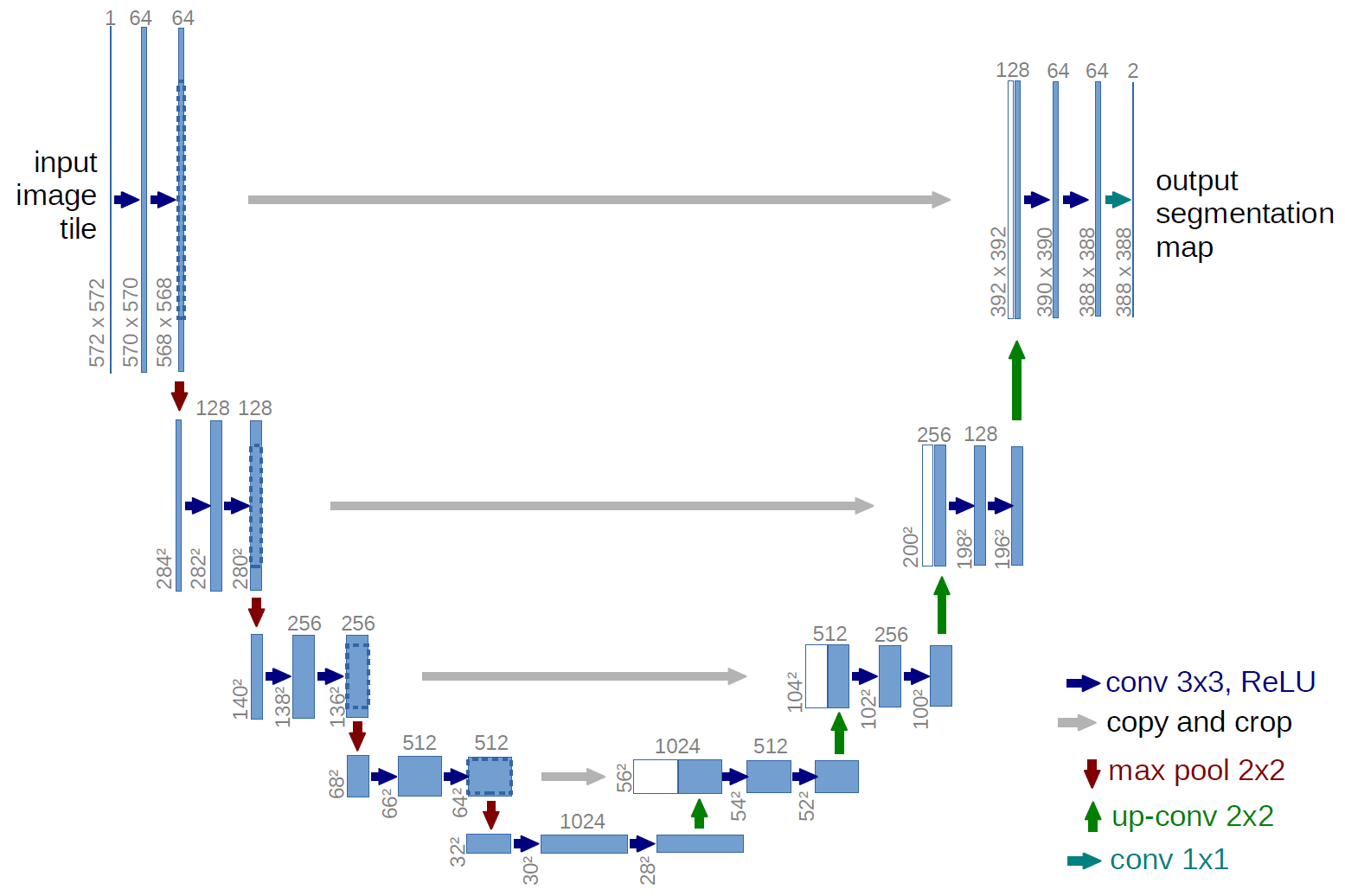


Рисунок 12 – Архитектура UNet [11]

Архитектура UNet состоит из сужающейся части слева, аналогичной типичной СНС, расширяющейся части справа, выходными данными получаем карту сегментации [11].

VGG16, предварительно обученный на данных ImageNET, кодирует входное изображение, перед отправкой его в сеть сегментации UNet. (смотрите рисунок 15).

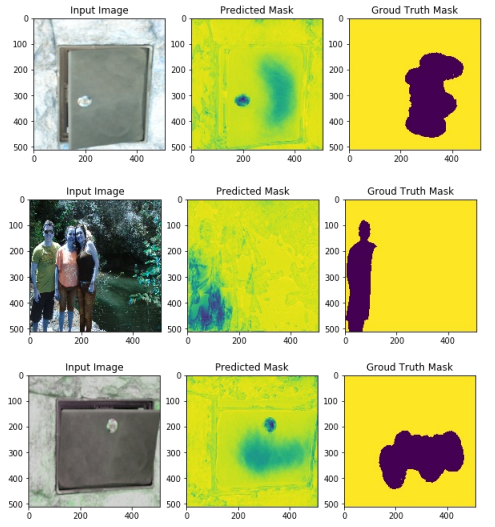


Рисунок 13 – Сравнение исходных и прогнозируемых масок VGG16+UNet [10]

Для обучения и тестирования UNet также используются исходные маски, ретушированных областей на изображении.

# ЗАКЛЮЧЕНИЕ

Подводя итог, хочется повторно отметить актуальность проблемы и важность создания и разработки методов обнаружения следов реконструкции изображения, мне самому приходилось часто сталкиваться в интернете с поддельными изображениями, это путает и вводит в замешательство.

В ходе работы мною были найдены различные источники с помощью поисковых систем Google и Google Scolar, Github, Inc. Была создана база данных ретушированных изображений, произведён запуск готовых решений для изображений из БД. Также был разработан алгоритм обнаружения ретуширования с использованием стандартных методов машинного обучения, с обучением моделей на текстурных признаках фрагментов изображений. Проведено теоретическое исследование методов обработки изображений для обнаружения подделок.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 50 free Machine Learning Datasets: Image Datasets/ cambridgespark. – Электрон. Дан. – [Б. м]. – URL: https://blog.cambridgespark.com/50-free-machine-learning-datasets-image-datasets-241852b03b49 (дата обращения 12.01.2023).

2 AOT-GAN-for-Inpainting [Электронный ресурс] / Github, Inc. – Электрон. Дан. – [Б. м.], 2021. – URL: https://github.com/researchmm/AOT-GAN-for-Inpainting (дата обращения 13.01.2023).

3 Hypergraphs-Image-Inpainting [Электронный ресурс] / Github, Inc. – Электрон. Дан. – [Б. м.], 2021. – URL: https://github.com/GouravWadhwa/Hypergraphs-Image-Inpainting (дата обращения 08.01.2023).

4 PAL4Inpaint [Электронный ресурс] / Github, Inc. – Электрон. Дан. – [Б. м.], 2021. – URL: https://github.com/owenzlz/PAL4Inpaint (дата обращения 06.01.2023).

5 Traditional-Feature-Extraction [Электронный ресурс]: Github, Inc. – Электрон. Дан. – [Б. м.], 2021. – URL: https://github.com/Rohit-Kundu/Traditional-Feature-Extraction (дата обращения 07.01.23).

6 Ансамблевые методы [Электронный ресурс] // scikit-learn. Машинное обучение в Python. – URL: https://scikit-learn.ru/1-11-ensemble-methods/?ysclid=ldt5tpmp8j631309522 (дата обращения 07.06.22)

7 Discrete versus Read AdaBoost [Электронный ресурс] // scikit-learn. Машинное обучение в Python. – URL: https://scikit-learn.ru/example/discrete-versus-real-adaboost// (дата обращения 10.01.23).

8 efg`s R Notes: EBImage: Haralick Textural Features Details [Электронный ресурс] // Хабр. 2015 .– URL: http://earlglynn.github.io/RNotes/package/EBImage/Haralick-Textural-Features.html (дата обращения 12.01.23).

9 Что такое свёрточная нейронная сеть [Электронный ресурс] // Хабр. 2016 .– URL: https://habr.com/ru/post/309508/ (дата обращения 05.01.23).

10 VGG16 – сверточная сеть для выделения признаков изображений [Электронный ресурс] // Neurohive. 2018. – URL: https://neurohive.io/ru/vidy-nejrosetej/vgg16-model/ (дата обращения 09.01.23).

11 Image forgery localization(IFL) using UNET architecture[Электронный ресурс] // Medium.com. 2020. – URL: https://medium.com/analytics-vidhya/image-forgery-localization-ifl-using-unet-architecture-772ba1b15a2d (дата обращения 09.01.23).

12 U-Net – нейросесть для сегментации изображений [Электронный ресурс] // Neurohive. 2018. – URL: https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/ (дата обращения 09.01.23).

# ПРИЛОЖЕНИЯ

**Код алгоритма обнаружения ретуширования**

import os

import shutil

from itertools import product

import natsort

import pandas as pd

import mahotas as mt

import glob

import csv

import time

import os

import skops.io as sio

import cv2

from PIL import Image, ImageDraw

tic = time.time()

def tile(filename, dir\_in, dir\_out, d):

name, ext = os.path.splitext(filename)

img = Image.open(os.path.join(dir\_in, filename))

w, h = img.size

grid = product(range(0, h - h % d, d), range(0, w - w % d, d))

for i, j in grid:

box = (j, i, j + d, i + d)

out = os.path.join(dir\_out, f'{name}\_{i}\_{j}{ext}')

img.crop(box).save(out)

wh = (w, h)

return wh

def concat(dir\_out,dir\_in,listdir, wh):

total\_width, max\_height = wh

total\_width = total\_width - total\_width % 32 - 32

max\_height = max\_height - max\_height % 32 - 32

result = Image.new('RGBA', (total\_width, max\_height)) # common canvas

y\_offset = 0

x\_offset = 0

for image in listdir:

img = Image.open(os.path.join(dir\_in, image))

if x\_offset < total\_width:

result.paste(img, (x\_offset, y\_offset))

x\_offset += 32

else:

y\_offset += 32

x\_offset = 0

img.close()

result.save(os.path.join(dir\_out, 'result.png'))

result.close()

print("[STATUS] Image predict saved : test/output/result.png")

# function to extract haralick textures from an image

def extract\_features(image):

# calculate haralick texture features for 4 types of adjacency

textures = mt.features.haralick(image)

# take the mean of it and return it

ht\_mean = textures.mean(axis=0)

return ht\_mean

if \_\_name\_\_ == '\_\_main\_\_':

wh = (640, 424)

dir\_in = r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\test\_image'

os.mkdir(r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\cropp\_test\_image')

dir\_out = r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\cropp\_test\_image'

for filename in os.listdir(dir\_in):

wh = tile(filename, dir\_in, dir\_out, 32)

# load the training dataset

train\_path = r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\cropp\_test\_image' # Enter the directory where all the images are stored

train\_names = os.listdir(train\_path)

# empty list to hold feature vectors and train labels

train\_features = []

train\_labels = []

# loop over the training dataset

print("[STATUS] Started extracting haralick textures..")

cur\_path = os.path.join(train\_path, '\*g')

cur\_label = train\_names

i = 0

with open('test\_fragments\_img.csv', 'a+', newline='') as obj:

writer = csv.writer(obj)

if i == 0:

writer.writerow(

['Haralick1', 'Haralick2', 'Haralick3', 'Haralick4', 'Haralick5', 'Haralick6', 'Haralick7', 'Haralick8',

'Haralick9',

'Haralick10', 'Haralick11', 'Haralick12', 'Haralick13'])

for file in glob.glob(cur\_path):

# print("Processing Image - {} in {}".format(i, cur\_label[i]))

# read the training image

image = cv2.imread(file)

# convert the image to grayscale

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# extract haralick texture from image

features = extract\_features(gray)

# print(features)

# append the feature vector and label

train\_features.append(features)

train\_labels.append(cur\_label[i])

writer.writerow(features)

# show loop update

i += 1

test = pd.read\_csv('test\_fragments\_img.csv', sep=',', header=0)

# LOAD MODELs

with open('models/Haralick\_LGBM\_model.pkl', 'rb') as f:

model = sio.load(f, trusted=True)

model\_prediction = list(model.predict(test))

list = []

m = 0

for i in model\_prediction:

m += 1

if i == 1:

list.append(m)

i = 0

TINT\_COLOR = (148, 0, 211)

TRANSPARENCY = .80 # Degree of transparency, 0-100%

OPACITY = int(255 \* TRANSPARENCY)

os.mkdir(r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\predict')

listdir = natsort.natsorted(os.listdir(r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\cropp\_test\_image'), reverse=False)

print("[STATUS] Started predict textures..")

for images in listdir:

with Image.open(os.path.join('C:/Users/Ilya/PycharmProjects/NIRB\_ML7/test/cropp\_test\_image/', images)) as im:

height, width = im.size

im = im.convert("RGBA")

i += 1

if i in list:

overlay = Image.new('RGBA', im.size, TINT\_COLOR + (0,))

draw = ImageDraw.Draw(overlay) # Create a context for drawing things on it.

draw.rectangle(((0, 0), (32, 32)), fill=TINT\_COLOR + (OPACITY,))

# Alpha composite these two images together to obtain the desired result.

im = Image.alpha\_composite(im, overlay)

im = im.convert("RGB") # Remove alpha for saving in jpg format.

im.save(r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\predict\predict\_image\_' + str(i) + '\_.png')

dir\_in = r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\predict'

dir\_path1 = 'C:/Users/Ilya/PycharmProjects/NIRB\_ML7/test/predict'

dir\_path2 = 'C:/Users/Ilya/PycharmProjects/NIRB\_ML7/test/cropp\_test\_image'

dir\_out = r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\output\_image'

files = os.listdir(dir\_in)

files = natsort.natsorted(files, reverse=False)

concat(dir\_out, dir\_in, files, wh)

try:

shutil.rmtree(dir\_path1)

except OSError as e:

print("Ошибка: %s : %s" %(dir\_path1, e.strerror))

dir\_path2 = 'C:/Users/Ilya/PycharmProjects/NIRB\_ML7/test/cropp\_test\_image'

try:

shutil.rmtree(dir\_path2)

except OSError as e:

print("Ошибка: %s : %s" % (dir\_path2, e.strerror))

os.remove(r'C:\Users\Ilya\PycharmProjects\NIRB\_ML7\test\_fragments\_img.csv')