Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки

«Анализ данных и машинное обучение»

**Анализ уровней сжатия пикселей на изображении для выявления фотошопа**

Разработчики проекта:

Ракитин Арсений Дмитриевич

Золотарёв Данил Сергеевич

Парамонов Арсений Сергеевич

Вершинин Никита Александрович

Моздакова Анастасия Александровна

Пермь, 2024

**ПАСПОРТ ПРОЕКТА**

**Название проекта:**Анализ уровней сжатия пикселей на изображении для выявления фотошопа.

**Сведения об авторах:**Ракитин Арсений Дмитриевич, Золотарёв Данил Сергеевич, Парамонов Арсений Сергеевич, Вершинин Никита Александрович, Моздакова Анастасия Александровна.

**Цель:**разработка программного комплекса с интерфейсом для логического анализа метаданных и нейросетевого анализа артефактов сжатия изображений, которым сможет воспользоваться пользователь без специальных навыков

**Задачи:**

1. Анализ проблемы, обоснование ее актуальности.

2. Сбор изображений для обучения нейросети.

3. Обучение нейросети.

4. Разработка и распространение приложения.

**Краткое описание проекта:**

Требуется обучить нейросеть, которая будет анализировать набор изображений на следы редактирования с помощью алгоритмов ELA и HFN и на основе результатов классифицировать каждое изображение от 0 до 1 в зависимости от того, поддельное фото или нет.

**Конкретные ожидаемые результаты:**

Обученная нейронная сеть.

**СОДЕРЖАНИЕ ПРОЕКТА**

**Анализ проблемы исследования**

В современном мире информационные технологии распространены повсеместно. В каждом доме есть вычислительные машины и инструменты для обработки информации. Стоит уделить особое внимание цифровым изображениям, т.к. данный вид информации может быть доказательством какого-либо важного факта. Из-за того, что в открытом доступе существуют инструменты для редактирования изображений, каждый человек, имеющий персональный компьютер, может редактировать изображение по своему усмотрению. Изображения, которые были отредактированы, являются фейковыми и могут быть использованы для доказательства ложных фактов. Чтобы заметить следы редактирования на изображении невооруженным глазом необходимо обладать навыками визуального анализа изображений. Но даже обладая такими навыками есть вероятность не определить подделку, если изображение было качественно отредактировано профессионалом. Для решения проблемы необходим такой инструмент, который поможет квалифицированным людям анализировать изображения на следы редактирования, а обычным пользователям без навыков самостоятельно проведет анализ.

Объектом исследования является признак артефактов сжатия и параметры метаданных цифрового изображения.

Предметом исследования является выявление фейковых изображений посредством анализа метаданных и артефактов сжатия.

**Исходные данные**

С сайтов paperswithcode.com и kaggle.com были найдены несколько датасетов:

* Casia V2;
* Coverage;
* ArtiFact.

В найденых датасетах есть выборки настоящих и подделанных изображений, но для анализа уровня ошибок подходит только Casia, из-за метода подделки изображений. В Coverage изображения подделаны методом копирования областей уже существующих пикселей изображения, а значит уровень сжатия будет таким же. В ArtiFact изображения подделаны методом генерацией нейросетью, а значит уровень сжатия на всем изображении будет одинаков. В Casia изображения подделаны методом вставки из одного изображения в другое, из-за чего появится разница в уровне сжатия, а следовательно, нейросеть сможет выявить признак подделки.

**Обзор используемых методов**

Программный комплекс будет основан на использовании трёх разных методов анализа изображений.

1. Метод HFN

HFN (high frequency noises) - это метод анализа высокочастотных шумов на изображении такого формата, который сохраняется с потерями в качестве. Смысл заключается в том, что в тех областях изображения, где частота шума выше, там наиболее вероятно было выполнено редактирование.

1. Метод ELA

ELA (Error Level Analysis) — это метод анализа цифровых изображений, который позволяет обнаружить изменения в уровнях сжатия изображения, которые могут указывать на манипуляции или редактирование изображения. ELA использует различные уровни сжатия JPEG для выявления различий в качестве изображения, которые могут возникать при редактировании или изменении содержимого фотографии.

Принцип ELA заключается в том, что при повторном сжатии изображения после редактирования области с изменениями будут иметь более высокий уровень ошибок (потерь качества) по сравнению с остальными частями изображения. Это может помочь выявить места, где произошли изменения, такие как удаление или добавление объектов, ретуширование или другие виды фотошопа.

1. Метод CNN

Нейросеть (Искусственная нейронная сеть) – это математическая модель в виде графа узлов и связей. Нейросеть состоит из слоев нейронов, имеющих связи (веса) с другими нейронами и функцию активации для передачи данных следующему слою. Такая организация позволяет выполнять множество задач: аппроксимировать любую функцию, распознать образы на изображении, прогнозировать курсы валют. Самое главное, чтобы в данных содержалась закономерность – коррелирующие параметры или признаки.

Для того, чтобы нейронная сеть выполняла задачи, необходимо настроить веса связей между нейронами посредством процесса обучения на обучающих данных.

CNN (Convolutional Neural Network) – это специализированный тип нейронной сети, который используется для анализа изображений. Тип CNN был разработан для распознавания признаков на изображении и классификации.

Суть модели CNN заключается в том, что используются сверточные слои для извлечения признаков из изображений. Эти сверточные слои применяют фильтры к различным областям, чтобы выделить признаки, в нашем случае это разница уровней ошибок обработки изображения методом ELA. Затем следуют слои пулинга, которые уменьшают размер матрицы и сохраняют наиболее важные признаки. Слой пулинга представляет собой нелинейную функцию уплотнения группы пикселей до одного пикселя.

**Структура проекта**

Проект был реализован в репозитории на платформе Github. Структура проекта представлена на рисунке 1.

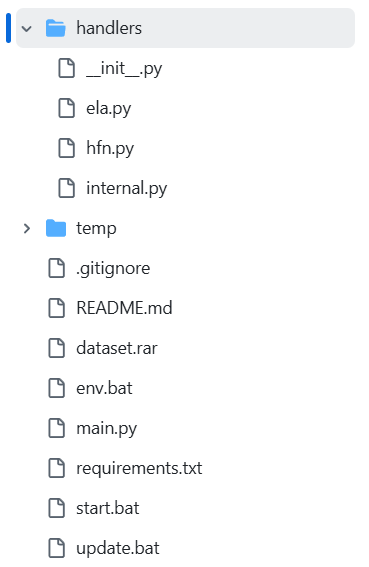
****

Рисунок 1 - Структура проекта

Файл main.py - это основной скрипт проекта. Его запуск запускает весь процесс обработки изображений, обучения модели и вывода результатов.

Файл requirements.txt - это файл, содержащий список библиотек Python, которые требуются для работы проекта.

Архив dataset.rar содержит данные для обучения. Он включается в себя папки с изображениями (fake и real), которые используются в скрипте main.py.

Файл README.md содержит инструкцию для запуска всего проекта.

Файл .gitignore используется в системах контроля версий, чтобы указать, какие файлы или папки нужно игнорировать при отслеживании изменений.

Файл env.bat создает виртуальную среду Python для работы с проектом.

Файл start.bat нужен для запуска проекта. Он запускает файл main.py.

Файл update.bat активирует виртуальную среду, устанавливает или обновляет библиотеки, перечисленные в requirements.txt.

Папка temp. Эта папка используется для временных файлов. Она содержит промежуточные результаты обработки данных или изображений.

Папка handlers содержит модули Python, которые отвечают за обработку изображений и извлечение признаков.

Файл \_\_init\_\_.py делает папку модулем Python. Это позволяет импортировать папку как модуль.

Файл ela.py реализует метод ELA (Error Level Analysis) для анализа ошибок сжатия изображений.

Файл hfn.py реализует метод HFN (High-Frequency Noise) для анализа высокочастотного шума в изображениях.

Файл internal.py предоставляет инструменты для предварительной обработки изображений перед их анализов или подачей в модель машинного обучения.

**Подготовка данных к анализу**

Подключаем необходимые библиотеки, получаем путь к папке с данными для обучения и к папкам, в которых находятся поддельные и реальные изображения, а также создаем списки для хранения признаков изображений, которые будут извлекаться с помощью ELA и HFN.

Создавать нейросеть с нуля не придется, так как существуют разные библиотеки, облегчающие разработку. Например, Keras – мощная библиотека, предоставляющая инструменты для создания и обучения нейронных сетей. К преимуществам библиотеки можно отнести то, что она имеет простой и понятный интерфейс, а также поддерживает разные виды нейросетей, в том числе и CNN.

import handlers, os

from keras.utils import to\_categorical

from keras.layers import Input, Convolution2D, Flatten, Dense, Concatenate, MaxPool2D, BatchNormalization, Dropout

from keras.models import Model

from sklearn.model\_selection import train\_test\_split

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

import numpy as np

DATASET\_TRAIN\_PATH = os.path.join('dataset', 'train')

FAKE\_FOLDER = os.path.join(DATASET\_TRAIN\_PATH, 'fake')

REAL\_FOLDER = os.path.join(DATASET\_TRAIN\_PATH, 'real')

X\_ELA = []

X\_HFN = []

Y = []

Рисунок 2 - Подключение библиотек и датафрейма

**Функционал для проведения Error Level Analysis (ELA)**

Файл ela.py реализует фукнционал для проведения анализа уровня ошибок изображения. Этот метод используется для выполнения манипуляций с изображением, таких как его редактирование, и работает путем анализа разницы между оригиналом изображения и его повторно сохраненной версией (например, в формате JPEG).

Сначала импортируем необходимые модули.

from PIL import Image

import numpy as np

from . import internal

Рисунок 3 - Импорт библиотек

Функция Prepare готовит изображение для анализа и подачи в модель машинного обучения. Внутри сначала вызывается функция ELA, чтобы получить изображение, обработанное методом Error Level Analysis. Затем функция изменяет размер полученного изображения до заданного (переменная size). Преобразует изображение в массив numpy и нормализует значения пикселей, деля их на 255 (приводя в диапазон [0, 1]). В конце функция возвращает этот массив.

def Prepare(pathToImage: str, quality: int, size: tuple[int, int]) -> np.ndarray:

image = ELA(pathToImage, quality)

image = image.resize(size)

data = np.array(image).flatten() / 255

return data

Рисунок 4 - Функция Prepare

Функция ELA открывает изображение по указанному пути и преобразует его в формат RGB. Затем вызывает функцию internal.Difference для вычисления разницы между оригиналом изображения и его повторно сохраненной версией с заданным качеством JPEG. Затем результат передается функции Enhance для усиления видимости артефактов и возвращает обработанное изображение.

def ELA(pathToImage: str, quality: int) -> Image:

image = Image.open(pathToImage).convert("RGB")

image = internal.Difference(image, quality)

image = Enhance(image)

return image

Рисунок 5 - Функция ELA

Функция Enhance усиливает видимость деталей в изображении, созданном на этапе вычисления разницы. Сначала определяется максимальное значение яркости (интенсивности пикселей) на изображении с помощью image.getextrema()[0]. Это диапазон яркости пикселей. Если диапазон яркости равен нулю (т.е. Изображение полностью черное), устанавливается extrema равным 1, чтобы избежать деления на ноль. Далее вызывается функция internal.Enhance, чтобы нормализовать яркость изображения, растягивая диапазон до [0, 255]. В конце возвращает усиленное изображение.

def Enhance(image: Image) -> Image:

extrema = max(image.getextrema()[0])

if extrema == 0:

extrema = 1

image = internal.Enhance(image, extrema)

return image

Рисунок 6 - Функция Enhance

**Функционал для проведения High Frequency Noise (HFN)**

Файл hfn.py реализует функционал для анализа высокочастотного шума в изображениях. Высокочастотный шум часто используется для выявления манипуляций с изображениями, поскольку в местах редактирования или подделки структура шума может измениться. Основная идея заключается в том, чтобы выделить и усилить высокочастотные компоненты изображения для анализа.

Сначала импортируем необходимые библиотеки.

from PIL import Image

import numpy as np

from . import internal

Рисунок 7 - Импорт библиотек

Функция Prepare подготавливает данные для использования в анализе или подаче в модель машинного обучения. Сначала вызывается функция HFN, которая извлекает высокочастотный шум из изображения. Затем функция изменяет размер изображения на указанный размер (size). Далее функция преобразует изображение в массив в numpy, нормализует пиксели (делит значениея на 255, чтобы они были в диапазоне [0, 1], и делает массив одномерным). В конце функция возвращает подготовленный массив.

def Prepare(pathToImage: str, quality: int, size: tuple[int, int]) -> np.ndarray:

image = HFN(pathToImage, quality)

image = image.resize(size)

data = np.array(image).flatten() / 255

return data

Рисунок 8 - Функция Prepare

Далее функция HFN должна извлечь высокочастотный шум из изображения. Сначала функция открывает изображение по указанному пути. Затем конвертирует изображение в формат L (градации серого), чтобы фокусироваться только на интенсивности пикселей, исключая цветовую информацию. Затем вызывается функция internal.Difference для вычисления разницы между оригиналом изображения и его повторно сохраненной версией с заданным качеством JPEG. В конце функция передает результат в функцию Enhance для усиления видимости высокочастотного шума и возвращает изображение.

def HFN(pathToImage: str, quality: int) -> Image:

image = Image.open(pathToImage).convert("L")

image = internal.Difference(image, quality)

image = Enhance(image)

return image

Рисунок 9 - Функция HFN

Функция Enhance усиливает высокочастотный шум, чтобы он стал более заметным для визуального анализа. Сначала функция вычисляет максимальное значение интенсивности пикселей на изображении (extrema). Если все пиксели чёрные (интенсивность равна 0), устанавливает extrema равным 1, чтобы избежать деления на ноль. Передает изображение и максимальное значение интенсивности в функцию internal.Enhance, которая масштабирует яркость изображения, усиливая шум. В конце возвращает усиленное изображение.

def Enhance(image: Image) -> Image:

extrema = max(image.getextrema())

if extrema == 0:

extrema = 1

image = internal.Enhance(image, extrema)

return image

Рисунок 10 - Функция Enhance

**Основная часть проекта**

Функция main() - это основная фукнция, которая выполняет подготовку данных, создание и обучение модели, а затем выводит график результатов.

def main():

prepare\_images()

shuffle\_data()

model = get\_model()

hist = train(model)

show\_history\_plot(hist)

Рисунок 3 - Функция main

Далее в функции prepare\_images() загружаются изображения из папок REAL\_FOLDER и FAKE\_FOLDER. Затем для каждого изображения:

* Извлекаются два типа признаков: ELA и HFN
* Признаки добавляются в соответствующие списки (X\_ELA и X\_HFN), а метка 0 или 1 добавляется в список Y (0 - реальное, 1 - поддельное)
* После этого изображения преобразуются в массивы NumPy для дальнейшей обработки

def prepare\_images():

global X\_ELA, X\_HFN, Y, FAKE\_FOLDER, REAL\_FOLDER

quality = 90

size = (128, 128)

print('Loading...')

amount = 0

for \_, \_, fileNames in os.walk(REAL\_FOLDER):

for fileName in fileNames:

amount += 1

for \_, \_, fileNames in os.walk(FAKE\_FOLDER):

for fileName in fileNames:

amount += 1

count = 0

for dirName, \_, fileNames in os.walk(REAL\_FOLDER):

for fileName in fileNames:

count += 1

fullPath = os.path.join(dirName, fileName)

X\_ELA.append(handlers.ELA(fullPath, quality, size))

X\_HFN.append(handlers.HFN(fullPath, quality, size))

Y.append(0)

if(count%50==0):

print ("\033[A \033[A")

print(f'{count}/{amount}')

for dirName, \_, fileNames in os.walk(FAKE\_FOLDER):

for fileName in fileNames:

count += 1

fullPath = os.path.join(dirName, fileName)

X\_ELA.append(handlers.ELA(fullPath, quality, size))

X\_HFN.append(handlers.HFN(fullPath, quality, size))

Y.append(1)

if(count%50==0):

print ("\033[A \033[A")

print(f'{count}/{amount}')

X\_ELA = np.array(X\_ELA).reshape(-1, 128, 128, 3)

X\_HFN = np.array(X\_HFN).reshape(-1, 128, 128, 1)

Y = to\_categorical(Y, 2)

Рисунок 4 - Функция prepare\_images()

**Моделирование**

При помощи фукнции shuffle\_data() перемешиваем данные в случайном порядке, чтобы улучшить обучение модели и избежать зависимости от порядка данных

def shuffle\_data():

global X\_ELA, X\_HFN, Y

datas = [[X\_ELA[i], X\_HFN[i], Y[i]] for i in range(0, len(X\_ELA))]

datas = shuffle(datas, random\_state=47)

X\_ELA = [data[0] for data in datas]

X\_HFN = [data[1] for data in datas]

Y = [data[2] for data in datas]

Рисунок 5 - Функция shuffle\_data()

Затем функция split\_data() разделяет данные на обучающую и тестовую выборки. Для этого используетсся метод train\_test\_split из библиотеки sklearn. Данные разделяются на две группы:

* X\_ELA и X\_HFN для признаков.
* Y для меток классов.

def split\_data():

global X\_ELA, X\_HFN, Y

x\_ela\_train, x\_ela\_test, y\_train, y\_test = train\_test\_split(X\_ELA, Y, test\_size = 0.2, random\_state=47)

x\_hfn\_train, x\_hfn\_test, \_, \_ = train\_test\_split(X\_HFN, Y, test\_size = 0.2, random\_state=47)

return np.array(x\_ela\_train), np.array(x\_hfn\_train), np.array(y\_train), np.array(x\_ela\_test), np.array(x\_hfn\_test), np.array(y\_test)

Рисунок 6 - Функция split\_data()

Далее функция get\_cnn\_model() создает и возвращает простую сверточную нейронную сеть (CNN), которая принимает изображения на входе:

* Слой свертки с 32 фильтрами размером 5х5, активацией ReLU.
* Слой подвыборки (MaxPool) для уменьшения размеров изображения.
* Dropout для предотвращения переобучения.
* Преобразование в одномерный вектор с помощью слоя Flattern.
* Полносвязный слой с 128 нейронами и активацией ReLU.

def get\_cnn\_model(input\_shape):

input\_layer = Input(shape=input\_shape)

conv\_layer = Convolution2D(32, (5, 5), activation='relu')(input\_layer)

pool\_layer = MaxPool2D(pool\_size = (2, 2))(conv\_layer)

drop\_layer = Dropout(0.25)(pool\_layer)

flat\_layer = Flatten()(drop\_layer)

dense\_layer = Dense(128, activation='relu')(flat\_layer)

return [input\_layer, dense\_layer]

Рисунок 7 - Функция get\_cnn\_model()

Затем внутри функции get\_model() создаем основную модель, которая комбинирует два CNN: Один для изображений с тремя каналами (RGB), другой - для изображений с одним каналом (например, для обработки признаков HFN). Признаки из обеих сетей объединяются с помощью слоя Concatenate. Затем идет еще один полносвязный слой с 128 нейронами и dropout. На выходе модель использует softmax для классификации на два класса (реальное или поддельное).

def get\_model():

cnn\_model1 = get\_cnn\_model((128, 128, 3))

cnn\_model2 = get\_cnn\_model((128, 128, 1))

conc\_layer = Concatenate(axis=1)([cnn\_model1[-1], cnn\_model2[-1]])

dense\_layer = Dense(128, activation='relu')(conc\_layer)

drop\_layer = Dropout(0.5)(dense\_layer)

output\_layer = Dense(2, activation='softmax')(drop\_layer)

model = Model(inputs=[cnn\_model1[0], cnn\_model2[0]], outputs=output\_layer)

model.summary()

return model

Рисунок 8 - Функция get\_model()

**Обучение модели**

Фукнция train() обучает модель, делает компиляцию модели с оптимизатором Adam и функцией потерь binary\_crossentropy. Обучение происходит на 15 эпохах с размером батча 50. После обучения модель сохраняется в файл model.keras. В конце функция возвращает историю обучения (объект hint), чтобы можно было вывести график.

**def train(model):**

**x\_ela\_train, x\_hfn\_train, y\_train, x\_ela\_test, x\_hfn\_test, y\_test = split\_data()**

**epochs = 15**

**batchSize = 50**

**model.compile(**

**optimizer = 'adam',**

**loss = 'binary\_crossentropy',**

**metrics = ['accuracy'])**

**hist = model.fit(**

**[x\_ela\_train, x\_hfn\_train],**

**y\_train,**

**batch\_size = batchSize,**

**epochs = epochs,**

**validation\_data = ([x\_ela\_test, x\_hfn\_test], y\_test))**

**model.save('model.keras')**

**return hist**

Рисунок 9 - Функция train()

**Визуализация**

Функция show\_history\_plot() показывает график обучения. График потерь (loss) и точности (accuracy) для обучающей и валидационной выборок. Используется библиотека matplotlib для визуализации.

**Заключение**

Нейросеть была обучена. Она может выявлять реальное и фейковое изображение с точностью 86%. Отсюда можно сделать вывод, что цель работы была достигнута.

**Список использованных источников и литературы**

1. Wang W., Dong J., Tan T. Tampered Region Localization of Digital Color Images Based on JPEG Compression Noise // Digital Watermarking: 9th International Workshop, IWDW 2010. Seoul, Korea. pp. 120-133.

2. Поддубный А.С., Полупанов А.А. Нейросетевой алгоритм обнаружения фейковых изображений // Прикладная математика: современные проблемы математики, информатики и моделирования: Материалы V всеросс. научно-практ. конф., молодых ученых. Краснодар, 2023. С. 454-456.

3. Casia dataset, Kaggle [Электронный ресурс], URL:<https://www.kaggle.com/datasets/sophatvathana/casia-dataset> (Дата обращения 04.11.2024)

**Приложение A**

**Программный код**

Файл main.py

import handlers, os

from keras.utils import to\_categorical

from keras.layers import Input, Convolution2D, Flatten, Dense, Concatenate, MaxPool2D, BatchNormalization, Dropout

from keras.models import Model

from sklearn.model\_selection import train\_test\_split

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.regularizers import l2

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

import tensorflow as tf

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.initializers import HeNormal

from tensorflow.keras.metrics import AUC

from keras.models import load\_model

from PIL import Image

ROOT\_DATASET\_PATH = 'CASIA2'

AU\_FOLDER = os.path.join(ROOT\_DATASET\_PATH, 'Au')

TP\_FOLDER = os.path.join(ROOT\_DATASET\_PATH, 'Tp')

X\_ELA = []

X\_HFN = []

Y = []

class\_names = ['real', 'fake']

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

def main():

while True:

print("\nМеню:")

print("1. Обучить нейросеть")

print("2. Выбрать изображение для предсказания")

choice = input("Введите номер действия (или 'exit' для выхода): ")

if choice == '1':

prepare\_images()

shuffle\_data()

model = get\_model()

history = train(model)

show\_history\_plot(history)

elif choice == '2':

model = load\_model('model.keras')

image\_path = input("Введите путь к изображению: ")

display\_image\_with\_prediction(image\_path, model)

elif choice.lower() == 'exit':

break

def prepare\_image\_for\_prediction(image\_path):

image\_ela = handlers.ELA(image\_path, quality=90, size=(128, 128))

image\_ela = image\_ela.reshape(1, 128, 128, 3)

image\_hfn = handlers.HFN(image\_path, quality=90, size=(128, 128))

image\_hfn = image\_hfn.reshape(1, 128, 128, 1)

return image\_ela, image\_hfn

def predict\_single\_image(image\_path, model):

image\_ela, image\_hfn = prepare\_image\_for\_prediction(image\_path)

y\_pred = model.predict([image\_ela, image\_hfn])

y\_pred\_class = np.argmax(y\_pred, axis=1)[0]

confidence = np.amax(y\_pred) \* 100

return class\_names[y\_pred\_class], confidence

image\_size = (128, 128)

def display\_image\_with\_prediction(image\_path, model):

image = Image.open(image\_path)

plt.imshow(image)

plt.axis('off')

plt.show()

image\_class, confidence = predict\_single\_image(image\_path, model)

print(f'Prediction: Class: {image\_class}, Confidence: {confidence:.2f}%')

def prepare\_images():

global X\_ELA, X\_HFN, Y, AU\_FOLDER, TP\_FOLDER, GROUNDTRUTH\_FOLDER

quality = 90

size = (128, 128)

print('Loading...')

amount = 0

for \_, \_, fileNames in os.walk(AU\_FOLDER):

for fileName in fileNames:

amount += 1

for \_, \_, fileNames in os.walk(TP\_FOLDER):

for fileName in fileNames:

amount += 1

count = 0

for dirName, \_, fileNames in os.walk(AU\_FOLDER):

for fileName in fileNames:

count += 1

fullPath = os.path.join(dirName, fileName)

X\_ELA.append(handlers.ELA(fullPath, quality, size))

X\_HFN.append(handlers.HFN(fullPath, quality, size))

Y.append(0)

if(count % 50 == 0):

print("\033[A \033[A")

print(f'{count}/{amount}')

for dirName, \_, fileNames in os.walk(TP\_FOLDER):

for fileName in fileNames:

count += 1

fullPath = os.path.join(dirName, fileName)

X\_ELA.append(handlers.ELA(fullPath, quality, size))

X\_HFN.append(handlers.HFN(fullPath, quality, size))

Y.append(1)

if(count % 50 == 0):

print("\033[A \033[A")

print(f'{count}/{amount}')

X\_ELA = np.array(X\_ELA).reshape(-1, 128, 128, 3)

X\_HFN = np.array(X\_HFN).reshape(-1, 128, 128, 1)

Y = to\_categorical(Y, 2)

def shuffle\_data():

global X\_ELA, X\_HFN, Y

datas = [[X\_ELA[i], X\_HFN[i], Y[i]] for i in range(0, len(X\_ELA))]

datas = shuffle(datas, random\_state=47)

X\_ELA = [data[0] for data in datas]

X\_HFN = [data[1] for data in datas]

Y = [data[2] for data in datas]

def split\_data():

global X\_ELA, X\_HFN, Y

x\_ela\_train, x\_ela\_test, y\_train, y\_test = train\_test\_split(X\_ELA, Y, test\_size = 0.2, train\_size=0.8, random\_state=47)

x\_hfn\_train, x\_hfn\_test, \_, \_ = train\_test\_split(X\_HFN, Y, test\_size = 0.2, train\_size=0.8, random\_state=47)

return np.array(x\_ela\_train), np.array(x\_hfn\_train), np.array(y\_train), np.array(x\_ela\_test), np.array(x\_hfn\_test), np.array(y\_test)

def get\_cnn\_model(input\_shape):

input\_layer = Input(shape=input\_shape)

conv\_layer = Convolution2D(32, (5, 5), activation='relu')(input\_layer)

pool\_layer = MaxPool2D(pool\_size = (2, 2))(conv\_layer)

drop\_layer = Dropout(0.25)(pool\_layer)

flat\_layer = Flatten()(drop\_layer)

dense\_layer = Dense(128, activation='relu')(flat\_layer)

return [input\_layer, dense\_layer]

def get\_model():

cnn\_model1 = get\_cnn\_model((128, 128, 3))

cnn\_model2 = get\_cnn\_model((128, 128, 1))

conc\_layer = Concatenate(axis=1)([cnn\_model1[-1], cnn\_model2[-1]])

dense\_layer = Dense(128, activation='relu')(conc\_layer)

drop\_layer = Dropout(0.5)(dense\_layer)

output\_layer = Dense(2, activation='softmax')(drop\_layer)

model = Model(inputs=[cnn\_model1[0], cnn\_model2[0]], outputs=output\_layer)

model.summary()

return model

def train(model):

x\_ela\_train, x\_hfn\_train, y\_train, x\_ela\_test, x\_hfn\_test, y\_test = split\_data()

epochs = 20

batchSize = 64

model.compile(

optimizer = Adam(learning\_rate=1e-4),

loss = 'binary\_crossentropy',

metrics = ['accuracy'])

hist = model.fit(

[x\_ela\_train, x\_hfn\_train],

y\_train,

batch\_size = batchSize,

epochs = epochs,

callbacks=[early\_stopping],

validation\_data = ([x\_ela\_test, x\_hfn\_test], y\_test))

model.save('model.keras')

return hist

def show\_history\_plot(hist):

plt.xlabel('Epoch')

plt.ylabel('Metric')

plt.plot(hist.history['loss'], color='maroon', label="Training loss")

plt.plot(hist.history['val\_loss'], color='red', label="Validation loss")

plt.plot(hist.history['accuracy'], color='darkgreen', label="Training accuracy")

plt.plot(hist.history['val\_accuracy'], color='lime', label="Validation accuracy")

plt.legend(loc='best', shadow=True)

plt.show()

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

main()

Файл ela.py

# Error Level Analysis

from PIL import Image

import numpy as np

from . import internal

\_\_all\_\_ = ["Prepare"]

def Prepare(pathToImage: str, quality: int, size: tuple[int, int]) -> np.ndarray:

image = ELA(pathToImage, quality)

image = image.resize(size)

data = np.array(image).flatten() / 255

return data

def ELA(pathToImage: str, quality: int) -> Image:

image = Image.open(pathToImage).convert("RGB")

image = internal.Difference(image, quality)

image = Enhance(image)

return image

def Enhance(image: Image) -> Image:

extrema = max(image.getextrema()[0])

if extrema == 0:

extrema = 1

image = internal.Enhance(image, extrema)

return image

Файл hfn.py

# High Frequency Noise

from PIL import Image

import numpy as np

from . import internal

\_\_all\_\_ = ["Prepare"]

def Prepare(pathToImage: str, quality: int, size: tuple[int, int]) -> np.ndarray:

image = HFN(pathToImage, quality)

image = image.resize(size)

data = np.array(image).flatten() / 255

return data

def HFN(pathToImage: str, quality: int) -> Image:

image = Image.open(pathToImage).convert("L")

image = internal.Difference(image, quality)

image = Enhance(image)

return image

def Enhance(image: Image) -> Image:

extrema = max(image.getextrema())

if extrema == 0:

extrema = 1

image = internal.Enhance(image, extrema)

return image

Файл internal.py

from PIL import Image, ImageChops, ImageEnhance

import uuid

import os

\_tempFolder = 'temp'

\_\_all\_\_ = []

def Difference(image: Image, quality: int) -> Image:

folder = os.path.join(\_tempFolder, str(uuid.uuid4()))

os.mkdir(folder)

path = os.path.join(folder, "resaved.jpg")

image.save(path, "JPEG", quality=quality)

resavedImage = Image.open(path)

difference = ImageChops.difference(image, resavedImage)

os.remove(path)

os.rmdir(folder)

return difference

def Enhance(image: Image, extrema: int) -> Image:

scale = 255 / extrema

image = ImageEnhance.Brightness(image).enhance(scale)

return image