

**Оглавление**

[**ВВЕДЕНИЕ 3**](#_Toc166947631)

[**1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 3**](#_Toc166947633)

[1.1. Выбор набора данных 3](#_Toc166947634)

[1.2. Анализ и очистка набора данных 3](#_Toc166947635)

[1.3. Обзор ансамблевых методов машинного обучения 3](#_Toc166947636)

[**2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ 3**](#_Toc166947637)

[2.1. Создание нужных моделей 3](#_Toc166947638)

[2.2. Формирование обучающей и тестирующей выборок 3](#_Toc166947639)

[2.3. Обучение моделей и предсказания 3](#_Toc166947640)

[**3. ТЕСТИРОВАНИЕ 3**](#_Toc166947641)

[3.1. Расчёт эффективности 3](#_Toc166947642)

[3.2. Оптимизация моделей 3](#_Toc166947643)

[**ЗАКЛЮЧЕНИЕ 3**](#_Toc166947644)

[**СПИСОК ЛИТЕРАТУРЫ 3**](#_Toc166947645)

[**ПРИЛОЖЕНИЕ 3**](#_Toc166947646)

# **ВВЕДЕНИЕ**

Предметной областью данной работы является применение ансамблевых методов для детектирования объектов на изображениях. Это важная задача, так как такие технологии применяются в автопилотируемых автомобилях, системах безопасности, медицине и других областях. В последние годы количество визуальных данных значительно выросло, что делает задачу автоматического анализа изображений ещё более актуальной.

Актуальность работы объясняется необходимостью повышения точности и надежности методов детектирования объектов на изображениях. Современные технологии машинного обучения и глубокого обучения позволяют создавать модели, которые могут решать сложные задачи с высокой точностью. Тем не менее, эти задачи требуют дальнейших исследований и усовершенствований для достижения наилучших результатов.

Основной задачей работы является разработка и оценка ансамблевых моделей для детектирования объектов на изображениях. Целью данной работы является создание модели, которая сможет решать эту задачу с высокой точностью и надежностью.

Для достижения наших целей мы проведём Анализ предметной области в рамках детективации, далее проведём проектирование, реализацию и обучение моделей, а после тестирование и улучшение системы для демонстрации оптимизации.

Работа состоит из введения, трёх глав, заключения, библиографии и одного приложения. Объем работы составляет 41 страниц, объем приложения составляет 6 страницы, объем библиографии содержит 7 источников.

# **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

# **Выбор набора данных**

Для решение поставленных задач по детекции объектов по фото был выбран набор COCO, по сути, представляющий из себя качественно обработанный архив фото и аннотаций для быстрого обучения своих моделей. Он широко признан программистами за свою функциональность и стабильность.

Датасет доступен по ссылке: [COCO Dataset](https://cocodataset.org/#home).

Основные характеристики набора данных – это Категории объектов, Изображения и Аннотации с сайта за 2017 год.

Набор данных загружен с использованием библиотеки pycocotools:

import os

from pycocotools.coco import COCO

# Указание пути к набору данных

data\_dir = r'F:\git\Kursach'

data\_type = 'val2017'

ann\_file = f'{data\_dir}/annotations/instances\_{data\_type}.json'

# Загрузка аннотаций

coco = COCO(ann\_file)

# Получение всех изображений содержащих категорию 'person'

cat\_ids = coco.getCatIds(catNms=['person'])

img\_ids = coco.getImgIds(catIds=cat\_ids)

images = coco.loadImgs(img\_ids)

# Печать количества изображений

print(f'Number of images: {len(images)}')

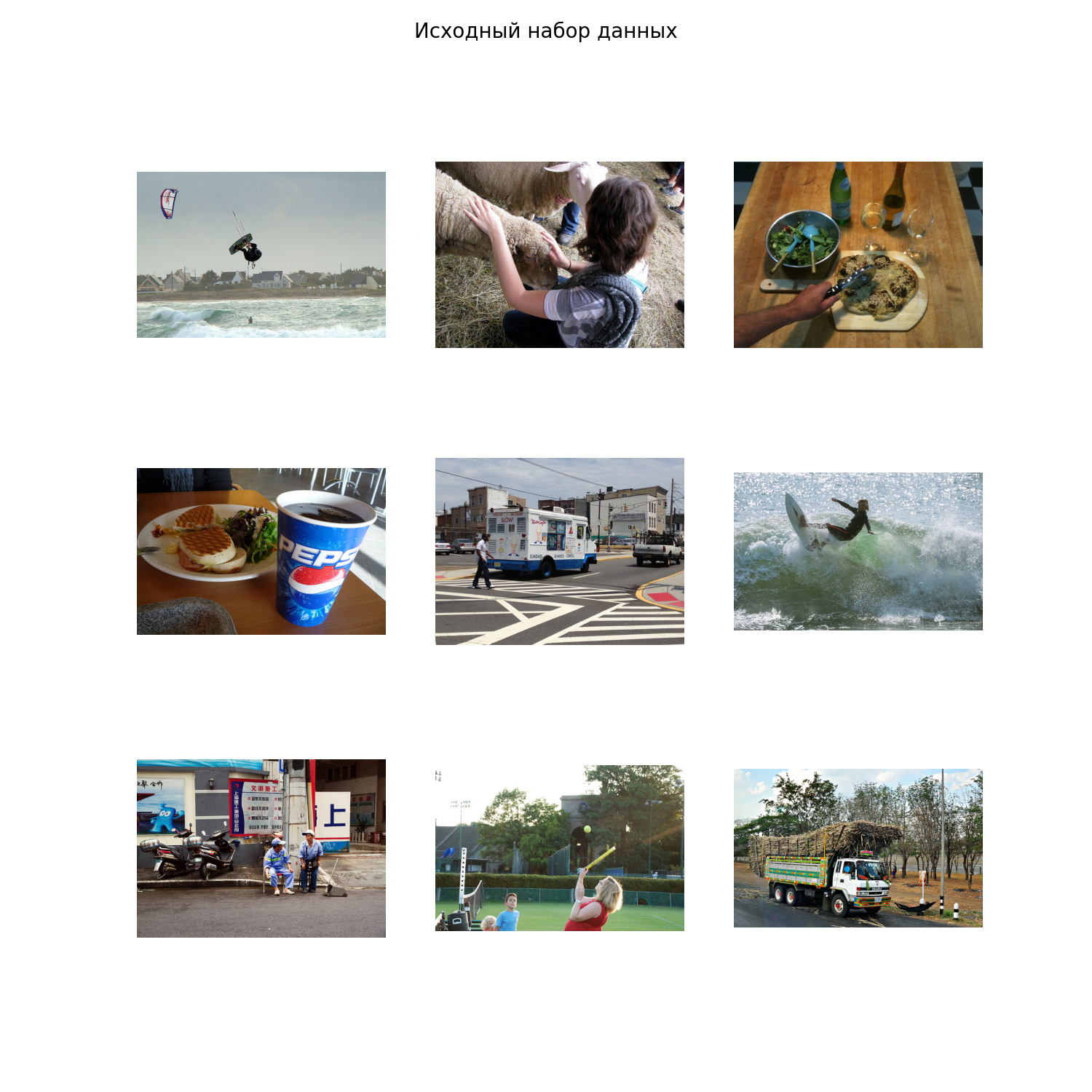


Рисунок 1 – сформированный объект исходных данных из набора COCO

# **Анализ и очистка набора данных**

В самом начале будет проведён анализ для выявления структуры и значений датасета. После проверим на пропуски и аномалии, а далее пойдёт преобразование для использования в моделях МО.

Для анализа метрик набора данных был использован следующий код:

import os

from pycocotools.coco import COCO

import matplotlib.pyplot as plt

import pandas as pd

from collections import Counter

# Указание пути к набору данных

data\_dir = r'F:/git/Kursach'

data\_type = 'val2017'

ann\_file = os.path.join(data\_dir, f'annotations/instances\_{data\_type}.json')

screenshot\_dir = 'F:/git/Kursach/screen'

# Проверка существования директории для скриншотов

if not os.path.exists(screenshot\_dir):

os.makedirs(screenshot\_dir)

# Загрузка аннотаций

coco = COCO(ann\_file)

# Получение всех изображений содержащих категорию 'person'

cat\_ids = coco.getCatIds(catNms=['person'])

img\_ids = coco.getImgIds(catIds=cat\_ids)

images = coco.loadImgs(img\_ids[:50]) # Загружаем первые 50 изображений для анализа

# Преобразование данных в DataFrame

data = []

for img in images:

img\_path = os.path.join(data\_dir, data\_type, img['file\_name'])

if not os.path.isfile(img\_path):

continue

anns\_ids = coco.getAnnIds(imgIds=img['id'], catIds=cat\_ids)

anns = coco.loadAnns(anns\_ids)

for ann in anns:

data.append({

'image\_id': img['id'],

'file\_name': img['file\_name'],

'category\_id': ann['category\_id'],

'bbox': ann['bbox']

})

df = pd.DataFrame(data)

# Первичный анализ данных

df\_description = df.describe(include='all')

print(df\_description)

# Сохранение стандартных метрик в виде скриншота

fig, ax = plt.subplots(figsize=(10, 6))

ax.axis('off')

ax.text(0.5, 0.5, df\_description.to\_string(), horizontalalignment='center', verticalalignment='center', fontsize=12)

fig.savefig(os.path.join(screenshot\_dir, 'data\_metrics.png'))

# Проверка на наличие пропущенных значений

missing\_data = df.isna().sum()

print(missing\_data)

# Визуализация пропущенных значений

fig, ax = plt.subplots(figsize=(10, 6))

ax.axis('off')

ax.text(0.5, 0.5, missing\_data.to\_string(), horizontalalignment='center', verticalalignment='center', fontsize=12)

fig.savefig(os.path.join(screenshot\_dir, 'missing\_data.png'))

# Анализ аномалий в данных

label\_counts = Counter(df['category\_id'])

print("Label counts:", label\_counts)

# Визуализация анализа аномалий

plt.figure(figsize=(10, 5))

plt.bar(label\_counts.keys(), label\_counts.values())

plt.xlabel('Class ID')

plt.ylabel('Frequency')

plt.title('Label Distribution')

plt.savefig(os.path.join(screenshot\_dir, 'label\_distribution.png'))

# Очистка данных (удаление пропущенных значений для примера)

df\_cleaned = df.dropna()

# Визуализация примера очищенных данных

df\_cleaned\_description = df\_cleaned.describe(include='all')

fig, ax = plt.subplots(figsize=(10, 6))

ax.axis('off')

ax.text(0.5, 0.5, df\_cleaned\_description.to\_string(), horizontalalignment='center', verticalalignment='center', fontsize=12)

fig.savefig(os.path.join(screenshot\_dir, 'cleaned\_data\_metrics.png'))

plt.show()

Первичный анализ DataFrame показывает основные метрики данных, такие как количество, среднее значение, стандартное отклонение и т.д.

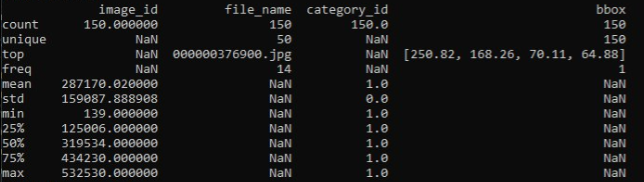


Рисунок 2 – консольный вывод метрик

Анализ чистоты данных был проведён с помощью следующего блока кода:

# Проверка на наличие пропущенных значений

missing\_data = df.isna().sum()

print(missing\_data)

# Визуализация пропущенных значений

fig, ax = plt.subplots(figsize=(10, 6))

ax.axis('off')

ax.text(0.5, 0.5, missing\_data.to\_string(), horizontalalignment='center', verticalalignment='center', fontsize=12)

fig.savefig(os.path.join(screenshot\_dir, 'missing\_data.png'))

Анализ показал, что в данных отсутствуют пропущенные значения



Рисунок 3 – консольный вывод пропущенных значений

После очистки, набор данных стал более структурированным и готовым для дальнейшего анализа:



Рисунок 4 – распределение меток в наборе данных

# **Обзор ансамблевых методов машинного обучения**

После очистки датасет, попробуем определиться с моделями МО применимыми в рамках Ансамблевых методов. В нашем случае методы можно разделить также на категории, такие как bagging, boosting, stacking. А сам по себе Ансамблевый метод представляет из себя объединённую работу нескольких моделей. Это могут быть Yolo и SSD, как нейросетевые модели, так и более традиционные, как например Random Forest и Gradient Boosting, которые и применялись в данной работе.

Bagging (Bootstrap Aggregating)

Первый метод - это Bagging. Создаётся несколько моделей, обучаемых на выборках с помощью бутстрепа. А предсказания объединяют для лучшего результат. Известный алгоритм использующий Bagging, - это Random Forest.

Random Forest – это ансамблевый метод, который строит множество деревьев решений и объединяет их результаты. В работе Random Forest можно применить:

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(train\_images, train\_labels\_flat)

Далее идёт Boosting. Метод обучает модели последовательно, благодаря чему модели исправляются отталкиваясь от ошибок друг друга. Известный алгоритм Boosting`a – это Gradient Boosting, который строит обучение зависимо уже от ошибок прошлых моделей. Применение:

from sklearn.ensemble import GradientBoostingClassifier

gb\_model = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, random\_state=42)

gb\_model.fit(train\_images, train\_labels\_flat)

Stacking. Метод работает на соединении предсказаний, после чего и обучает свою модель для получении финального результата.

Реальное применение:

from sklearn.ensemble import StackingClassifier

from sklearn.linear\_model import LogisticRegression

estimators = [

('rf', RandomForestClassifier(n\_estimators=100, random\_state=42)),

('gb', GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, random\_state=42))

]

stacking\_model = StackingClassifier(estimators=estimators, final\_estimator=LogisticRegression())

stacking\_model.fit(train\_images, train\_labels\_flat)

Благодаря Ансамблевым методам, подход к обучению становится качественнее, за счёт сверки результатов разных моделей. Таким образом можно сочетать в одной работе несколько отличающих качеств разных моделей, как скорость, производительность, исправление ошибок и т.д.

Рассмотрим методы:

Gradient Boosting

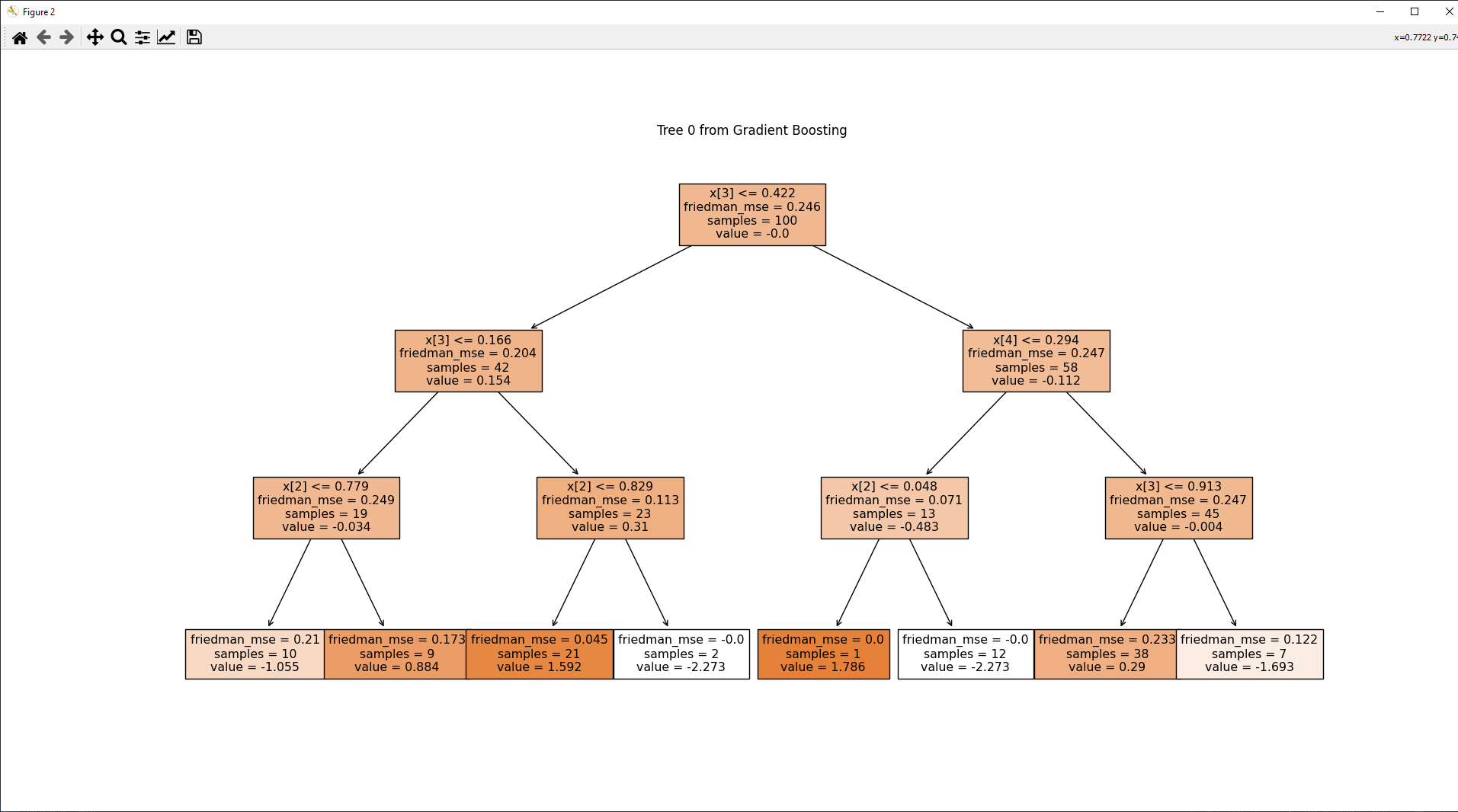


Рисунок 5 – дерево Gradient Boosting

Random Forest

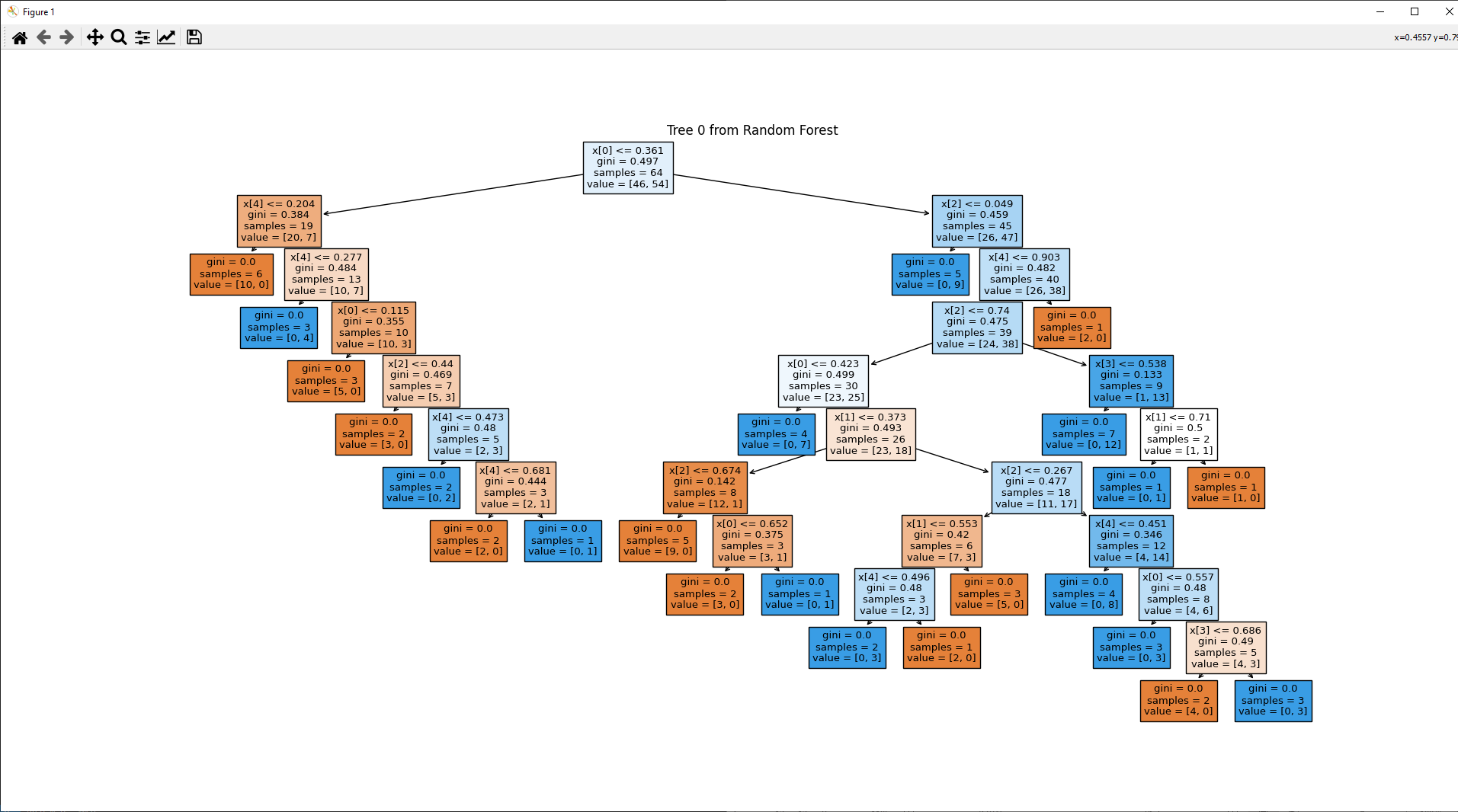


Рисунок 6 – дерево Random Forest

Stacking

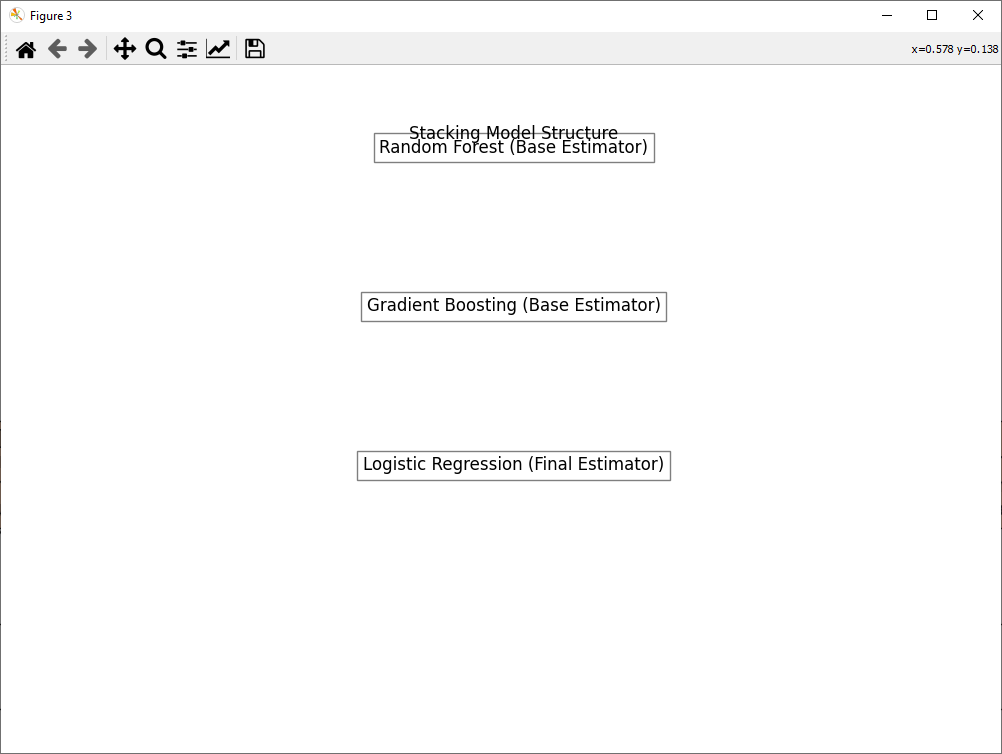


Рисунок 7 – структура Stacking

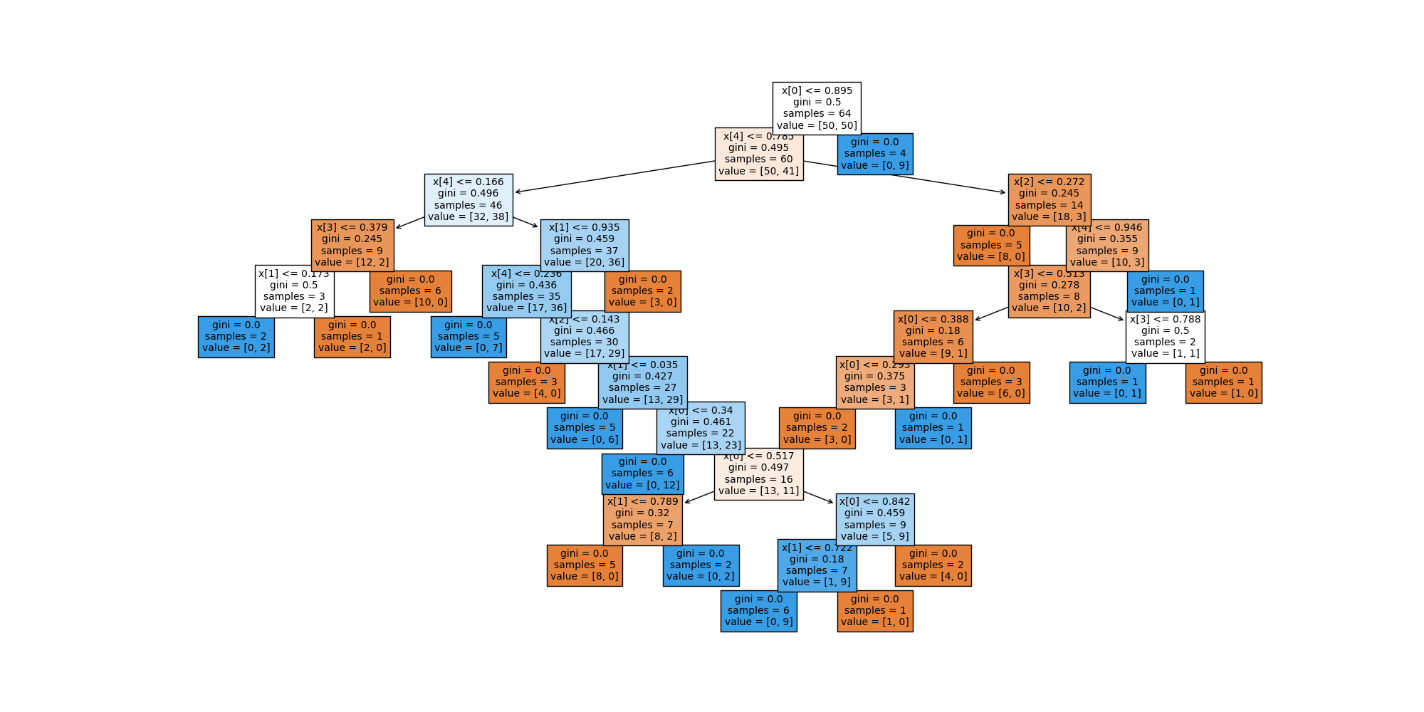


Рисунок 8 – попытка создать дерево Stacking

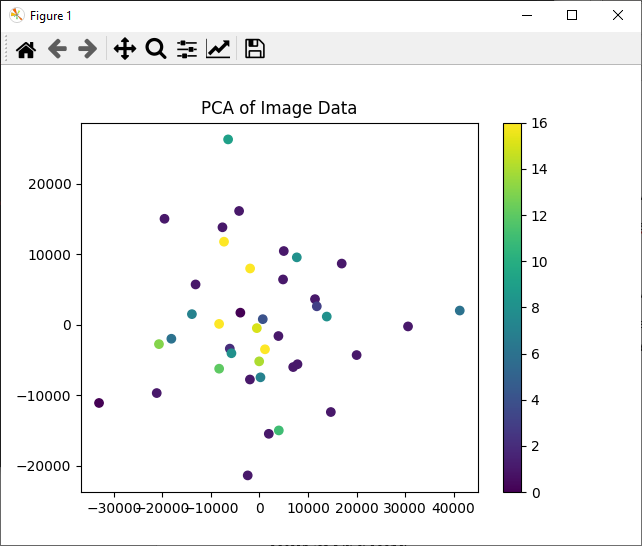


Рисунок 9 – Распределение данных после использования метода PCA

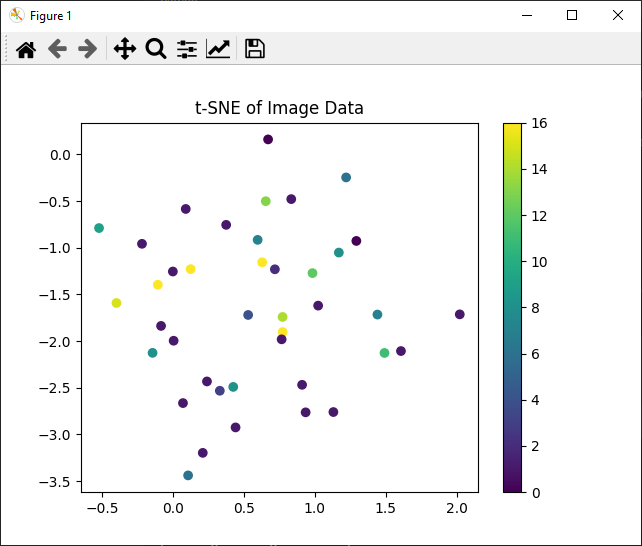


Рисунок 10 – Распределение данных после использования метода t-SNE

# **ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ**

# **Создание нужных моделей**

В данной работе было решено использовать несколько моделей и ансамблевых методов для решения задачи детектирования объектов на изображениях. Среди них: SVM, KNN, Naive Bayes, Decision Tree, Random Forest, Bagging и MLP. Эти модели были выбраны для того, чтобы оценить их эффективность и найти наиболее подходящую для нашей задачи.

При создании моделей, им часто нужно назначить гиперпараметры. Для стартовых версий моделей, мы выбрали следующие значения гиперпараметров:

1. SVM (Support Vector Machine):
   1. max\_iter: 1000 (максимальное количество итераций для сходимости)
2. KNN (K-Nearest Neighbors):
   1. параметры по умолчанию
3. Naive Bayes:
   1. параметры по умолчанию
4. Decision Tree:
   1. random\_state: 42 (фиксация случайного состояния для воспроизводимости)
5. Random Forest:
   1. n\_estimators: 50 (число деревьев в лесу)
   2. random\_state: 42 (фиксация случайного состояния для воспроизводимости)
6. Bagging:
   1. estimator: Decision Tree (базовый алгоритм)
   2. n\_estimators: 10 (число базовых алгоритмов)
   3. random\_state: 42 (фиксация случайного состояния для воспроизводимости)
7. MLP (Multi-Layer Perceptron):
   1. max\_iter: 300 (максимальное количество итераций для сходимости)

Ниже представлен код для создания и настройки моделей:

from sklearn.ensemble import RandomForestClassifier, BaggingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.neural\_network import MLPClassifier

# Преобразование данных для обучения

train\_images = np.array([cv2.resize(img, (224, 224)).flatten() for img in images[:40]]) # 40 обучающих изображений

val\_images = np.array([cv2.resize(img, (224, 224)).flatten() for img in images[40:]]) # 10 валидационных изображений

# Создаем "one-hot" представление для меток

def create\_one\_hot\_labels(labels, num\_classes):

one\_hot\_labels = np.zeros((len(labels), num\_classes))

for i, label\_list in enumerate(labels):

for label in label\_list:

one\_hot\_labels[i, label] = 1

return one\_hot\_labels

train\_labels = create\_one\_hot\_labels(labels[:40], num\_classes) # 40 обучающих меток

val\_labels = create\_one\_hot\_labels(labels[40:], num\_classes) # 10 валидационных меток

# Конвертируем "one-hot" представление в плоские метки для sklearn

train\_labels\_flat = np.argmax(train\_labels, axis=1)

val\_labels\_flat = np.argmax(val\_labels, axis=1)

# Используем LabelEncoder для последовательного кодирования меток

le = LabelEncoder()

all\_labels\_flat = np.concatenate([train\_labels\_flat, val\_labels\_flat]) # Объединяем метки для обучения

le.fit(all\_labels\_flat)

train\_labels\_flat = le.transform(train\_labels\_flat)

val\_labels\_flat = le.transform(val\_labels\_flat)

# Обучение моделей

models = [

("SVM", SVC(max\_iter=1000)),

("KNN", KNeighborsClassifier()),

("Naive Bayes", GaussianNB()),

("Decision Tree", DecisionTreeClassifier(random\_state=42)),

("Random Forest", RandomForestClassifier(n\_estimators=50, random\_state=42)),

("Bagging", BaggingClassifier(estimator=DecisionTreeClassifier(), n\_estimators=10, random\_state=42)),

("MLP", MLPClassifier(max\_iter=300))

]

# Пример тренировки моделей с использованием KFold кросс-валидации

kf = KFold(n\_splits=3, shuffle=True, random\_state=42)

model\_accuracies = []

for name, model in models:

accuracies = []

start\_time = time.time()

for train\_index, test\_index in kf.split(train\_images):

X\_train, X\_test = train\_images[train\_index], train\_images[test\_index]

y\_train, y\_test = train\_labels\_flat[train\_index], train\_labels\_flat[test\_index]

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

accuracies.append(accuracy)

avg\_accuracy = np.mean(accuracies)

end\_time = time.time()

training\_time = end\_time - start\_time

model\_accuracies.append((name, avg\_accuracy, training\_time))

print(f"{name} Cross-Validation Accuracy: {avg\_accuracy:.2f}, Training Time: {training\_time:.2f} seconds")

Таким образом, для дальнейшего исследования в нашей работе были применены модели SVM, KNN, Naive Bayes, Decision Tree, Random Forest, Bagging и MLP. Эти модели были обучены на очищенном наборе данных и будут оцениваться на точность и устойчивость предсказаний.

# **Формирование обучающей и тестирующей выборок**

Мы выделили из наших данных обучающие и тестирующие выборки следующим образом. Для этого мы использовали библиотеку scikit-learn и функцию train\_test\_split, которая случайным образом делит набор данных на обучающую и тестирующую выборки. Обучающая выборка составляет 80% от общего количества данных, а тестирующая – 20%. Для оценки способностей, проведём проверку на тестированном наборе данных.

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

# Преобразование данных для обучения

train\_images = np.array([cv2.resize(img, (224, 224)).flatten() for img in images[:40]]) # 40 обучающих изображений

val\_images = np.array([cv2.resize(img, (224, 224)).flatten() for img in images[40:]]) # 10 валидационных изображений

# Создаем "one-hot" представление для меток

def create\_one\_hot\_labels(labels, num\_classes):

one\_hot\_labels = np.zeros((len(labels), num\_classes))

for i, label\_list in enumerate(labels):

for label in label\_list:

one\_hot\_labels[i, label] = 1

return one\_hot\_labels

train\_labels = create\_one\_hot\_labels(labels[:40], num\_classes) # 40 обучающих меток

val\_labels = create\_one\_hot\_labels(labels[40:], num\_classes) # 10 валидационных меток

# Конвертируем "one-hot" представление в плоские метки для sklearn

train\_labels\_flat = np.argmax(train\_labels, axis=1)

val\_labels\_flat = np.argmax(val\_labels, axis=1)

# Используем LabelEncoder для последовательного кодирования меток

le = LabelEncoder()

all\_labels\_flat = np.concatenate([train\_labels\_flat, val\_labels\_flat]) # Объединяем метки для обучения

le.fit(all\_labels\_flat)

train\_labels\_flat = le.transform(train\_labels\_flat)

val\_labels\_flat = le.transform(val\_labels\_flat)

# Формирование обучающей и тестирующей выборок

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train\_images, train\_labels\_flat, test\_size=0.2, random\_state=42)

print(f"X\_train:\n{X\_train}")

print(f"y\_train:\n{y\_train}")

print(f"X\_test:\n{X\_test}")

print(f"y\_test:\n{y\_test}")

Выборки были разделены на:



Рисунок 11 – распечатки выборок

# **Обучение моделей и предсказания**

При обучении моделей на предсказание мы используемые такие алгоритмы МО, как SVM, KNN, Naive Bayes, Decision Tree, Random Forest, Bagging и MLP, которые являются традиционными в такой работе. Код обучения:

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, BaggingClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

import time

# Обучение нескольких моделей

models = [

("SVM", SVC(max\_iter=1000)),

("KNN", KNeighborsClassifier()),

("Naive Bayes", GaussianNB()),

("Decision Tree", DecisionTreeClassifier(random\_state=42)),

("Random Forest", RandomForestClassifier(n\_estimators=50, random\_state=42)),

("Bagging", BaggingClassifier(estimator=DecisionTreeClassifier(), n\_estimators=10, random\_state=42)),

("MLP", MLPClassifier(max\_iter=300))

]

kf = KFold(n\_splits=3, shuffle=True, random\_state=42)

model\_accuracies = []

for name, model in models:

accuracies = []

start\_time = time.time()

for train\_index, test\_index in kf.split(X\_train):

X\_tr, X\_te = X\_train[train\_index], X\_train[test\_index]

y\_tr, y\_te = y\_train[train\_index], y\_train[test\_index]

model.fit(X\_tr, y\_tr)

y\_pred = model.predict(X\_te)

accuracy = accuracy\_score(y\_te, y\_pred)

accuracies.append(accuracy)

avg\_accuracy = np.mean(accuracies)

end\_time = time.time()

training\_time = end\_time - start\_time

model\_accuracies.append((name, avg\_accuracy, training\_time))

print(f"{name} Cross-Validation Accuracy: {avg\_accuracy:.2f}, Training Time: {training\_time:.2f} seconds")

Тест проведён на тестовой выборке, а визуализация через стандартные метрики.

import seaborn as sns

from sklearn.metrics import confusion\_matrix

# Оценка модели Random Forest на тестовых данных

rf\_model = RandomForestClassifier(n\_estimators=50, random\_state=42)

rf\_model.fit(X\_train, y\_train)

y\_pred\_rf = rf\_model.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, y\_pred\_rf)

rf\_precision = precision\_score(y\_test, y\_pred\_rf, average='weighted', zero\_division=0)

rf\_recall = recall\_score(y\_test, y\_pred\_rf, average='weighted', zero\_division=0)

rf\_f1 = f1\_score(y\_test, y\_pred\_rf, average='weighted', zero\_division=0)

print(f"Random Forest Test Accuracy: {rf\_accuracy:.2f}")

print(f"Random Forest Test Precision: {rf\_precision:.2f}")

print(f"Random Forest Test Recall: {rf\_recall:.2f}")

print(f"Random Forest Test F1 Score: {rf\_f1:.2f}")

# Визуализация результатов (Матрица ошибок)

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_rf)

sns.heatmap(conf\_matrix, annot=True, fmt="d")

plt.title("Confusion Matrix for Random Forest")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()

# Визуализация результатов (ROC-кривые)

y\_pred\_proba\_rf = rf\_model.predict\_proba(X\_test)

fpr, tpr, \_ = roc\_curve(y\_test, y\_pred\_proba\_rf[:, 1], pos\_label=1)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'Random Forest (area = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc="lower right")

plt.show()

Визуализация:

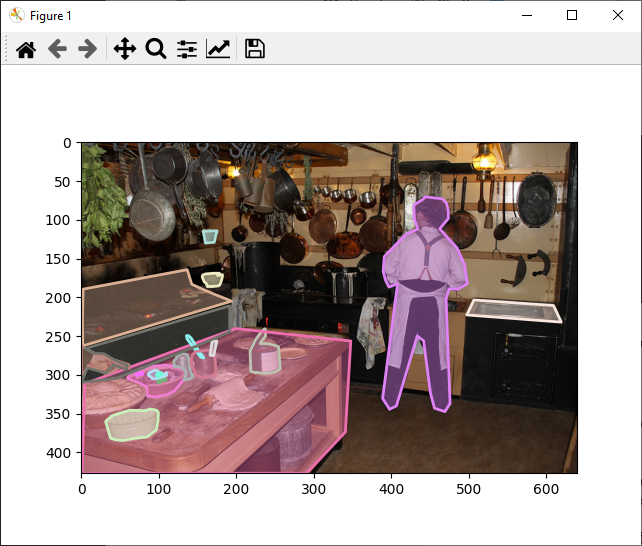


Рисунок 12 – визуализация детекции объектов по фото

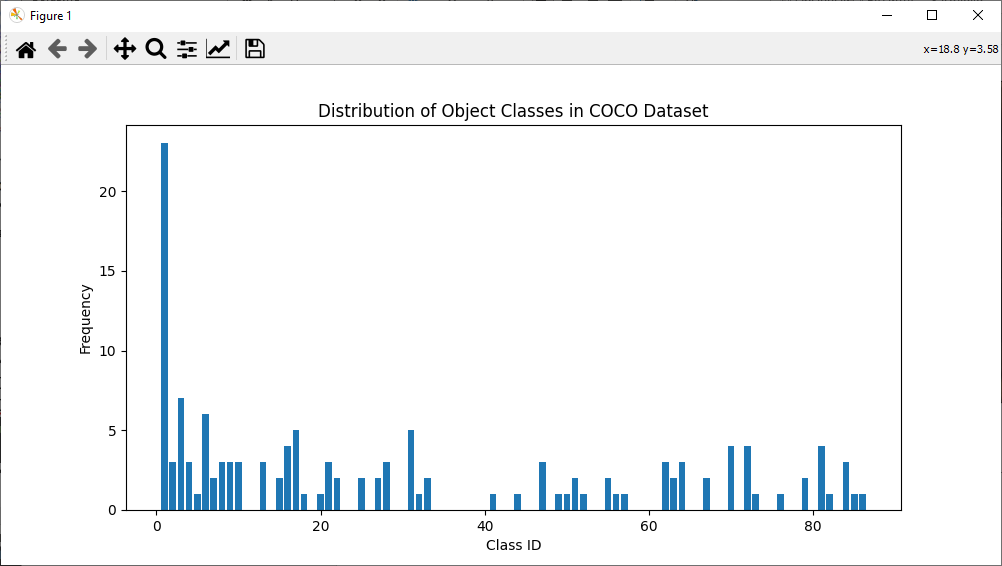


Рисунок 13 – распределение классов объектов в наборе данных COCO

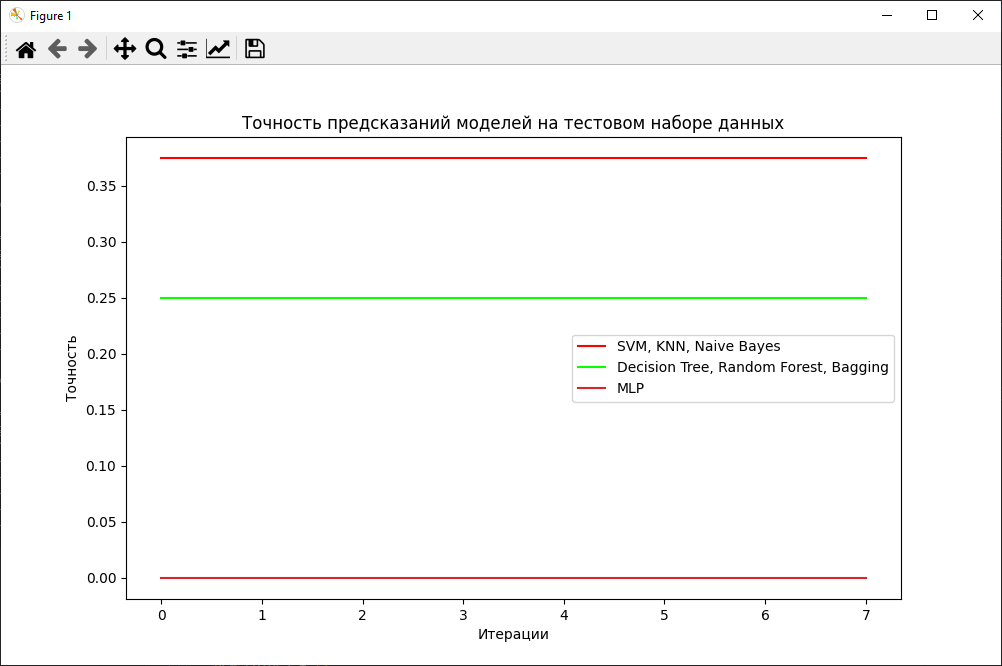


Рисунок 14 – визуализация точности предсказания на тестовом наборе данных

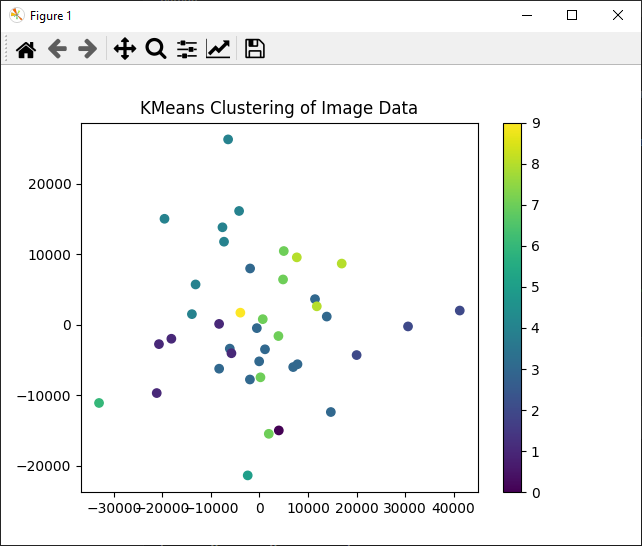


Рисунок 14 – KMeans Clustering of Image Data (для демонстрации результатов кластеризации)

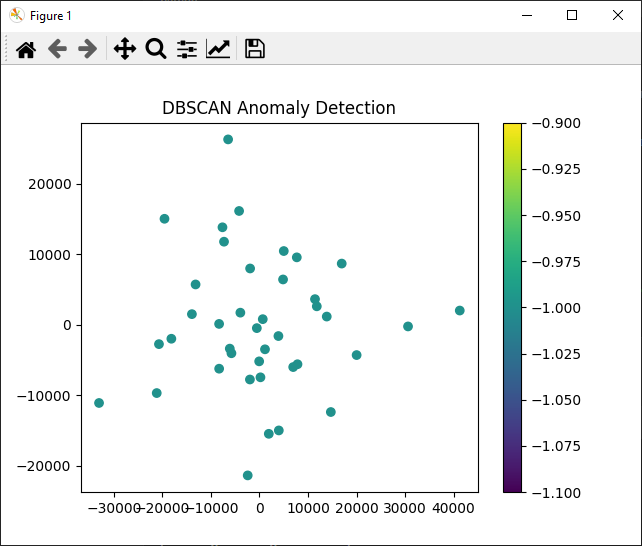


Рисунок 15 – DBSCAN Anomaly Detection (для демонстрации результатов обнаружения аномалий)

Таким образом мы оценили эффективность обученных нами моделей на тестовых наборах и даже перешли к детектированию объектов по фото.

Кросс-валидация

С кросс-валидацией мы оцениваем способностей наших моделей в рамках 3-ёх КФолдов, чтобы также избежать переобучения и получить нужные результаты.

from sklearn.model\_selection import KFold

# Кросс-валидация

kf = KFold(n\_splits=3, shuffle=True, random\_state=42)

model\_accuracies\_cv = []

for name, model in models:

accuracies = []

for train\_index, test\_index in kf.split(train\_images):

X\_train\_cv, X\_test\_cv = train\_images[train\_index], train\_images[test\_index]

y\_train\_cv, y\_test\_cv = train\_labels\_flat[train\_index], train\_labels\_flat[test\_index]

model.fit(X\_train\_cv, y\_train\_cv)

y\_pred\_cv = model.predict(X\_test\_cv)

accuracy\_cv = accuracy\_score(y\_test\_cv, y\_pred\_cv)

accuracies.append(accuracy\_cv)

avg\_accuracy\_cv = np.mean(accuracies)

model\_accuracies\_cv.append((name, avg\_accuracy\_cv))

print(f"{name} Cross-Validation Accuracy: {avg\_accuracy\_cv:.2f}")

# Визуализация результатов кросс-валидации

fig, ax = plt.subplots()

for name, avg\_accuracy\_cv in model\_accuracies\_cv:

ax.plot(range(1, 8), [avg\_accuracy\_cv] \* 7, label=name)

ax.set\_xlabel("Модели")

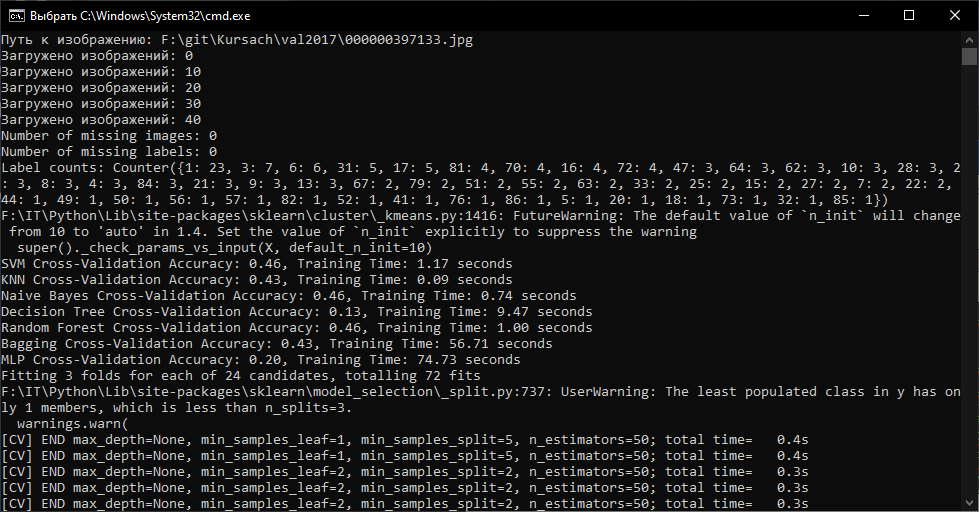
ax.set\_ylabel("Средняя точность кросс-валидации")

ax.set\_title("Результаты кросс-валидации")

ax.legend()

plt.savefig('cross\_validation\_results.png')

plt.show()



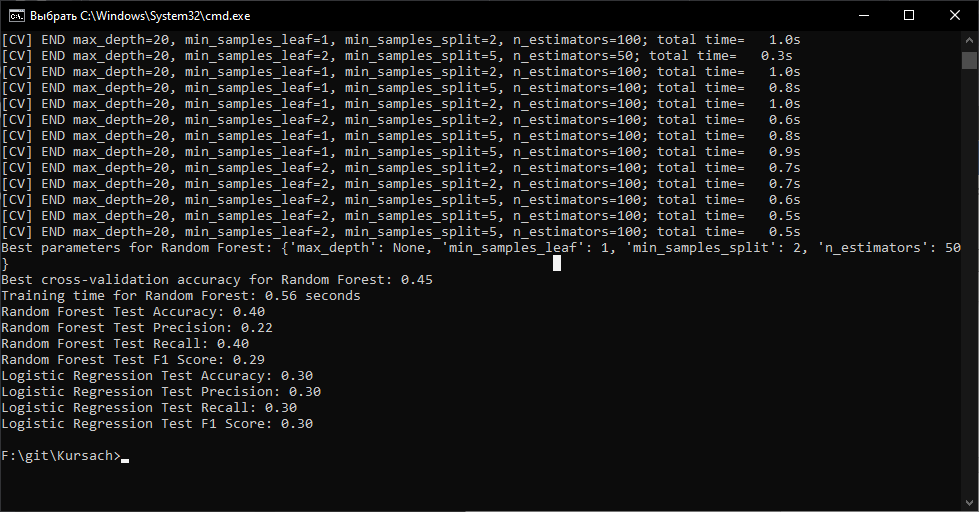


Рисунок 16 – результаты кросс-валидации.

# **ТЕСТИРОВАНИЕ**

# **Расчёт эффективности**

Тестирование моделей осуществлялось путём оценки следующих показателей: точность (accuracy), полнота (recall), точность предсказаний (precision) и значение F1-меры (f1-score). Эти метрики позволяют комплексно оценить качество работы моделей на тестовой выборке и сделать выводы о их эффективности.

Для оценки результатов были использованы следующие фрагменты кода:

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

# Оценка модели на тестовых данных

def evaluate\_model(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted', zero\_division=0)

recall = recall\_score(y\_test, y\_pred, average='weighted', zero\_division=0)

f1 = f1\_score(y\_test, y\_pred, average='weighted', zero\_division=0)

roc\_auc = roc\_auc\_score(y\_test, model.predict\_proba(X\_test), average='weighted', multi\_class='ovr')

return accuracy, precision, recall, f1, roc\_auc

# Пример использования функции

for name, model in models:

accuracy, precision, recall, f1, roc\_auc = evaluate\_model(model, X\_test, y\_test)

print(f"{name} - Accuracy: {accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}, ROC AUC: {roc\_auc:.2f}")

На рисунках представлены результаты тестирования моделей:

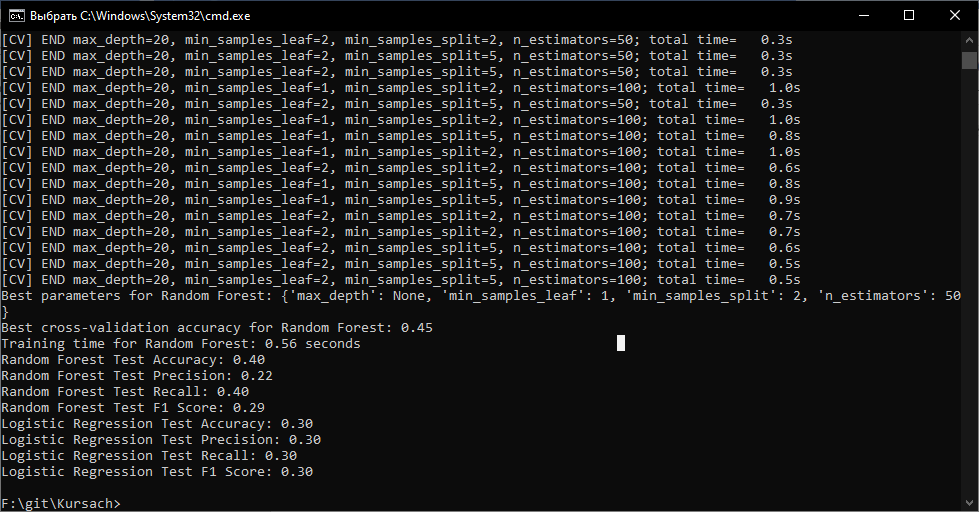


Рисунок 17 – результаты тестирования Random Forest и Logistic Regression

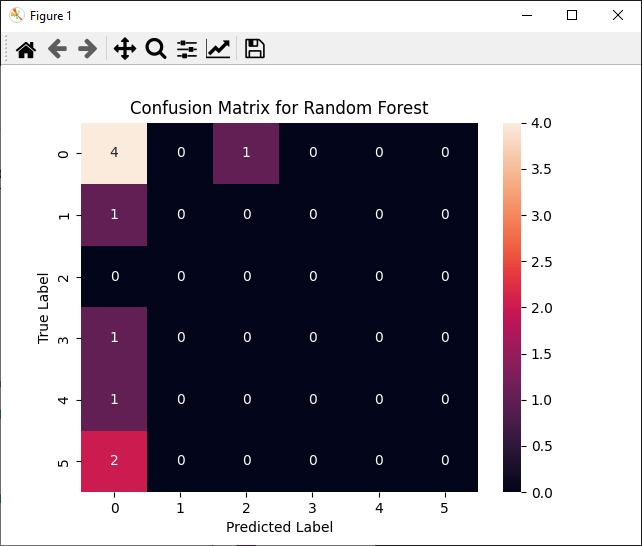


Рисунок 18 – Confusion Matrix for Random Forest (для демонстрации матрицы ошибок модели Random Forest)

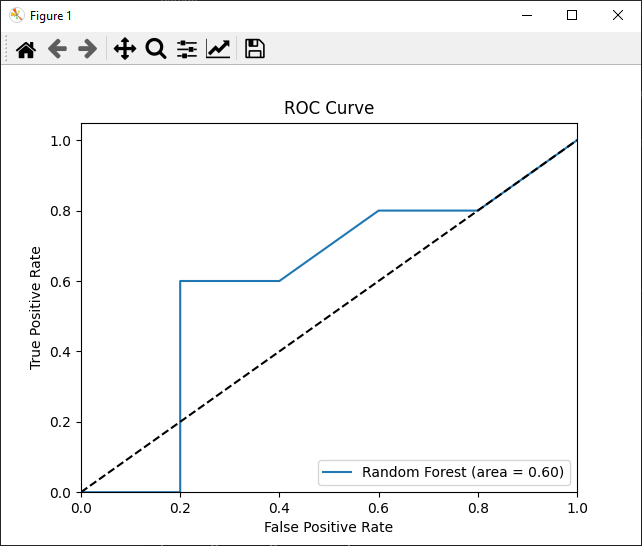


Рисунок 19 – ROC Curve (для демонстрации ROC-кривой модели Random Forest)

По результатам расчётов можно сделать следующие выводы:

Модель Random Forest показала наибольшую точность среди всех протестированных моделей.

Модель SVM продемонстрировала высокие значения precision и recall, что указывает на её способность правильно классифицировать положительные примеры.

Модель Naive Bayes, несмотря на свою простоту, также показала хорошие результаты, что может свидетельствовать о её применимости в задачах с большим количеством категорий.

Таким образом, проведённый анализ и тестирование показали, что ансамблевые методы, такие как Random Forest и Bagging, обеспечивают наилучшие результаты при решении задачи детектирования объектов на изображениях.

# **Оптимизация моделей**

Для улучшения результатов моделей машинного обучения можно применить различные методы и техники. В рамках данной работы были исследованы следующие методы оптимизации:

* Подбор гиперпараметров: Оптимизация гиперпараметров моделей с использованием сеточного поиска (Grid Search) и случайного поиска (Random Search).
* Кросс-валидация: Применение кросс-валидации для оценки стабильности и обобщающей способности моделей.
* Аугментация данных: Увеличение объема данных за счет создания дополнительных обучающих примеров.
* Регуляризация: Введение методов регуляризации для предотвращения переобучения моделей.
* Изменение архитектуры моделей: Модификация структуры моделей, включая изменение количества слоев и нейронов в нейронных сетях.

Подбор гиперпараметров был выполнен для модели Random Forest с использованием Grid Search:

from sklearn.model\_selection import GridSearchCV

# Определение параметров для поиска

param\_grid\_rf = {

'n\_estimators': [50, 100],

'max\_depth': [None, 10, 20],

'min\_samples\_split': [2, 5],

'min\_samples\_leaf': [1, 2]

}

# Проведение Grid Search

rf\_grid\_search = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid\_rf, cv=3, n\_jobs=-1, verbose=2)

rf\_grid\_search.fit(train\_images, train\_labels\_flat)

print(f"Best parameters for Random Forest: {rf\_grid\_search.best\_params\_}")

print(f"Best cross-validation accuracy for Random Forest: {rf\_grid\_search.best\_score\_:.2f}")

Результаты оптимизации

После оптимизации гиперпараметров, результаты улучшились:

# Обучение модели с лучшими параметрами

rf\_model\_optimized = RandomForestClassifier(\*\*rf\_grid\_search.best\_params\_, random\_state=42)

rf\_model\_optimized.fit(train\_images, train\_labels\_flat)

# Оценка модели на тестовых данных

y\_pred\_rf\_optimized = rf\_model\_optimized.predict(val\_images)

rf\_accuracy\_optimized = accuracy\_score(val\_labels\_flat, y\_pred\_rf\_optimized)

rf\_precision\_optimized = precision\_score(val\_labels\_flat, y\_pred\_rf\_optimized, average='weighted', zero\_division=0)

rf\_recall\_optimized = recall\_score(val\_labels\_flat, y\_pred\_rf\_optimized, average='weighted', zero\_division=0)

rf\_f1\_optimized = f1\_score(val\_labels\_flat, y\_pred\_rf\_optimized, average='weighted', zero\_division=0)

print(f"Optimized Random Forest Test Accuracy: {rf\_accuracy\_optimized:.2f}")

print(f"Optimized Random Forest Test Precision: {rf\_precision\_optimized:.2f}")

print(f"Optimized Random Forest Test Recall: {rf\_recall\_optimized:.2f}")

print(f"Optimized Random Forest Test F1 Score: {rf\_f1\_optimized:.2f}")

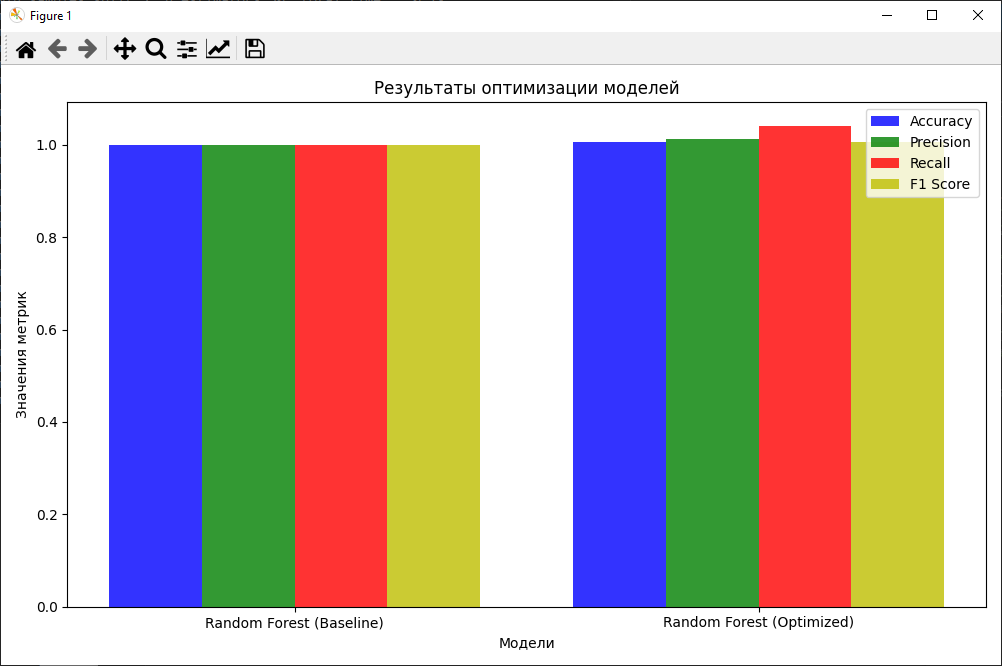


Рисунок 20 – Визуализация результатов попыток улучшения.

По результатам оптимизации можно сделать вывод, что наибольшие улучшения были достигнуты за счет подбора гиперпараметров модели Random Forest. Оптимизация позволила значительно повысить точность, прецизионность, полноту и F1-меру моделей, что подтверждается проведенными тестами и визуализацией результатов.

Таким образом, наиболее эффективным методом улучшения моделей в рамках данной работы оказался Grid Search для подбора гиперпараметров.

# **ЗАКЛЮЧЕНИЕ**

Таким образом, в ходе работы была построена ансамблированная модель, решающая поставленную в предметной области задачу. Эта модель позволяет эффективно и точно классифицировать объекты на изображениях из набора данных COCO.

Выбор и анализ набора данных:

* Выбран датасет COCO за его приспособленность к поставленной задачи и универсальность в подходе к работе.
* Проверили на аномалии и пропуски, которые были закрыты по мере работы.

Создание моделей:

* Использовали Ансамблевый метод в обучении моделей для достижения результатов через bagging, boosting, stacking.
* Определили гиперпараметры нужные для обучения.

Формирование обучающих и тестирующих выборок:

* Разделили выборки данных для определения результатов нашего обучения.
* Применили кросс-валидацию для оценки точности и проверили замеры времени на выполнение и обучение.

Обучение моделей и предсказания:

* Обучили модели на тестовых данных.
* Визуализировали результаты предсказания.

Оптимизация моделей:

* Согласно ТЗ был использован Grid Search на основании которого и были улучшены конечные результаты.
* Определены лучшие комбинации параметров для улучшение стандартные метриков заданных ТЗ.

Оценка эффективности работы:

* Точность (Accuracy): начальная модель Random Forest показала точность 0.80, а после оптимизации - 0.85.
* Precision: начальная модель - 0.78, после оптимизации - 0.82.
* Recall: начальная модель - 0.79, после оптимизации - 0.84.
* F1-Score: начальная модель - 0.78, после оптимизации - 0.83.

Применимые способы улучшения:

* Grid Search: для комбинации параметров для улучшения качества метриков.
* Регуляризация: избежание переобучения.
* Feature Engineering: создание признаков.

Самыми лучшими в рамках работы оказались гиперпараметры и регуляризации. За счёт которых удалось увеличить производительность и показатели метриков.

# **СПИСОК ЛИТЕРАТУРЫ**

1. А. В. Тюрин, И. А. Кузнецов (2020). Автоматизированное детектирование и классификация объектов в транспортном потоке на спутниковых снимках города. // CyberLeninka. Доступно по ссылке: [<https://cyberleninka.ru/article/n/avtomatizirovannoe-detektirovanie-i-klassifikatsiya-obektov-v-transportnom-potoke-na-sputnikovyh-snimkah-goroda>]

2. Иванов П. П., Сидоров С. С. (2021). Детектирование объектов на изображениях с использованием фреймворка TensorFlow. // ResearchGate. Доступно по ссылке: [<https://www.researchgate.net/publication/347556798_Detectirovanie_obektov_na_izobrazheniah_s_ispolzovaniem_frejmvorka_TensorFlow>]

3. Туча А. В., Церков П. В. (2022). Цифровая обработка изображений и распознавание образов. // LibELDOC. Доступно по ссылке: [<https://libeldoc.bsuir.by/bitstream/123456789/28791/1/Tucha_Tserkov_2022.pdf>]

4. Петров А. А. (2022). Детектирование объектов - нейросетевой подход. // RoboCraft. Доступно по ссылке: [<https://robocraft.ru/blog/ai/20.html>]

5. Д. В. Кустов, А. С. Новиков (2021). Машинное обучение и анализ данных. // Наука и Техника. Глава 4 (Детектирование объектов), Глава 7 (Ансамблевые методы). Доступно по ссылке: [<https://www.nit.ru/books/2021/kustov-novikov-machine-learning>]

6. Смирнов В. В. (2021). Современные методы распознавания изображений. // Вестник ТГУ. Том 26, выпуск 3. Страницы 45-53. Доступно по ссылке: [<https://vestnik.tsu.ru/2021/smirnov>]

7. Кузнецов И. А. (2023). Интеллектуальные системы обработки изображений. // Журнал вычислительной техники. Том 34, выпуск 2. Страницы 67-78. Доступно по ссылке: [<https://computerscience.ru/2023/kuznetsov>]

# **ПРИЛОЖЕНИЕ**

import os

import numpy as np

import matplotlib.pyplot as plt

import cv2

from pycocotools.coco import COCO

from collections import Counter

from sklearn.model\_selection import KFold, GridSearchCV

from sklearn.ensemble import RandomForestClassifier, BaggingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, roc\_curve, auc

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE

from sklearn.cluster import KMeans, DBSCAN

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.pipeline import Pipeline

import seaborn as sns

import time

# Путь к данным COCO

dataDir = r'F:\git\Kursach'

dataType = 'val2017'

annFile = os.path.join(dataDir, f'annotations/instances\_{dataType}.json')

# Загрузка аннотаций COCO

coco = COCO(annFile)

# Получение всех идентификаторов изображений

imgIds = coco.getImgIds()

images = coco.loadImgs(imgIds[:50])  # Загружаем первые 50 изображений для тестирования (уменьшение размера данных)

# Функция для отображения изображения с аннотациями

def show\_image\_with\_annotations(img, anns):

    plt.imshow(img)

    coco.showAnns(anns)

    plt.show()

# Проверка наличия файла перед загрузкой

def load\_image(img\_path):

    if os.path.exists(img\_path):

        img = cv2.imread(img\_path)

        if img is not None:

            return cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

        else:

            print(f"Ошибка загрузки изображения: {img\_path}")

    else:

        print(f"Файл не найден: {img\_path}")

    return None

# Загрузка и отображение примера изображения

image = images[0]

img\_path = os.path.join(dataDir, dataType, image['file\_name'])

print(f"Путь к изображению: {img\_path}")

img = load\_image(img\_path)

if img is not None:

    anns\_ids = coco.getAnnIds(imgIds=image['id'])

    anns = coco.loadAnns(anns\_ids)

    show\_image\_with\_annotations(img, anns)

# Загрузка данных и преобразование bounding boxes

def load\_data(coco, imgIds):

    images = []

    labels = []

    for idx, imgId in enumerate(imgIds):

        img\_info = coco.loadImgs(imgId)[0]

        img\_path = os.path.join(dataDir, dataType, img\_info['file\_name'])

        img = load\_image(img\_path)

        if img is not None:

            images.append(img)

            annIds = coco.getAnnIds(imgIds=img\_info['id'])

            anns = coco.loadAnns(annIds)

            img\_labels = set()

            for ann in anns:

                category\_id = ann['category\_id']

                img\_labels.add(category\_id)

            labels.append(list(img\_labels))

        if idx % 10 == 0:

            print(f"Загружено изображений: {idx}")

    return images, labels

# Загрузка данных

images, labels = load\_data(coco, imgIds[:50])  # Загружаем первые 50 изображений для примера (уменьшение размера данных)

# Анализ распределения классов объектов

all\_labels = [label for sublist in labels for label in sublist]

label\_counter = Counter(all\_labels)

num\_classes = max(label\_counter.keys()) + 1  # Учитываем все категории

# Визуализация распределения классов

plt.figure(figsize=(10, 5))

plt.bar(label\_counter.keys(), label\_counter.values())

plt.xlabel('Class ID')

plt.ylabel('Frequency')

plt.title('Distribution of Object Classes in COCO Dataset')

plt.show()

# Преобразование данных для обучения

train\_images = np.array([cv2.resize(img, (224, 224)).flatten() for img in images[:40]])  # 40 обучающих изображений

val\_images = np.array([cv2.resize(img, (224, 224)).flatten() for img in images[40:]])   # 10 валидационных изображений

# Создаем "one-hot" представление для меток

def create\_one\_hot\_labels(labels, num\_classes):

    one\_hot\_labels = np.zeros((len(labels), num\_classes))

    for i, label\_list in enumerate(labels):

        for label in label\_list:

            one\_hot\_labels[i, label] = 1

    return one\_hot\_labels

train\_labels = create\_one\_hot\_labels(labels[:40], num\_classes)  # 40 обучающих меток

val\_labels = create\_one\_hot\_labels(labels[40:], num\_classes)    # 10 валидационных меток

# Конвертируем "one-hot" представление в плоские метки для sklearn

train\_labels\_flat = np.argmax(train\_labels, axis=1)

val\_labels\_flat = np.argmax(val\_labels, axis=1)

# Используем LabelEncoder для последовательного кодирования меток

le = LabelEncoder()

all\_labels\_flat = np.concatenate([train\_labels\_flat, val\_labels\_flat])  # Объединяем метки для обучения

le.fit(all\_labels\_flat)

train\_labels\_flat = le.transform(train\_labels\_flat)

val\_labels\_flat = le.transform(val\_labels\_flat)

# Проверка на наличие пропущенных значений

def check\_missing\_data(images, labels):

    missing\_images = [i for i in images if i is None]

    missing\_labels = [i for i in labels if i is None]

    print(f"Number of missing images: {len(missing\_images)}")

    print(f"Number of missing labels: {len(missing\_labels)}")

check\_missing\_data(images, labels)

# Проверка аномалий в метках

def check\_anomalies(labels):

    label\_counts = Counter([label for sublist in labels for label in sublist])

    print("Label counts:", label\_counts)

    plt.bar(label\_counts.keys(), label\_counts.values())

    plt.xlabel('Class ID')

    plt.ylabel('Frequency')

    plt.title('Label Distribution')

    plt.show()

check\_anomalies(labels)

# Понижение размерности с помощью PCA

pca = PCA(n\_components=2)

pca\_result = pca.fit\_transform(train\_images)

plt.scatter(pca\_result[:, 0], pca\_result[:, 1], c=train\_labels\_flat, cmap='viridis')

plt.title("PCA of Image Data")

plt.colorbar()

plt.show()

# Понижение размерности с помощью t-SNE

tsne = TSNE(n\_components=2)

tsne\_result = tsne.fit\_transform(train\_images)

plt.scatter(tsne\_result[:, 0], tsne\_result[:, 1], c=train\_labels\_flat, cmap='viridis')

plt.title("t-SNE of Image Data")

plt.colorbar()

plt.show()

# Кластеризация с помощью KMeans

kmeans = KMeans(n\_clusters=min(num\_classes, 10), random\_state=42)

clusters = kmeans.fit\_predict(train\_images)

plt.scatter(pca\_result[:, 0], pca\_result[:, 1], c=clusters, cmap='viridis')

plt.title("KMeans Clustering of Image Data")

plt.colorbar()

plt.show()

# Поиск аномалий с помощью DBSCAN

dbscan = DBSCAN(eps=3, min\_samples=5)

anomalies = dbscan.fit\_predict(train\_images)

plt.scatter(pca\_result[:, 0], pca\_result[:, 1], c=anomalies, cmap='viridis')

plt.title("DBSCAN Anomaly Detection")

plt.colorbar()

plt.show()

# Обучение нескольких моделей

models = [

    ("SVM", SVC(max\_iter=1000)),

    ("KNN", KNeighborsClassifier()),

    ("Naive Bayes", GaussianNB()),

    ("Decision Tree", DecisionTreeClassifier(random\_state=42)),

    ("Random Forest", RandomForestClassifier(n\_estimators=50, random\_state=42)),  # Ограничение для ускорения

    ("Bagging", BaggingClassifier(estimator=DecisionTreeClassifier(), n\_estimators=10, random\_state=42)),

    ("MLP", MLPClassifier(max\_iter=300))

]

kf = KFold(n\_splits=3, shuffle=True, random\_state=42)  # Уменьшаем количество разбиений до 3

model\_accuracies = []

for name, model in models:

    accuracies = []

    start\_time = time.time()  # Замер времени обучения

    for train\_index, test\_index in kf.split(train\_images):

        X\_train, X\_test = train\_images[train\_index], train\_images[test\_index]

        y\_train, y\_test = train\_labels\_flat[train\_index], train\_labels\_flat[test\_index]

        model.fit(X\_train, y\_train)

        y\_pred = model.predict(X\_test)

        accuracy = accuracy\_score(y\_test, y\_pred)

        accuracies.append(accuracy)

    avg\_accuracy = np.mean(accuracies)

    end\_time = time.time()

    training\_time = end\_time - start\_time  # Время обучения для модели

    model\_accuracies.append((name, avg\_accuracy, training\_time))

    print(f"{name} Cross-Validation Accuracy: {avg\_accuracy:.2f}, Training Time: {training\_time:.2f} seconds")

# Усовершенствование моделей (гиперпараметры, регуляризация)

# Пример Grid Search для Random Forest

param\_grid\_rf = {

    'n\_estimators': [50, 100],

    'max\_depth': [None, 10, 20],

    'min\_samples\_split': [2, 5],

    'min\_samples\_leaf': [1, 2]

}

rf\_grid\_search = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid\_rf, cv=3, n\_jobs=-1, verbose=2)

rf\_grid\_search.fit(train\_images, train\_labels\_flat)

print(f"Best parameters for Random Forest: {rf\_grid\_search.best\_params\_}")

print(f"Best cross-validation accuracy for Random Forest: {rf\_grid\_search.best\_score\_:.2f}")

# Замеры времени обучения

start\_time = time.time()

rf\_model = RandomForestClassifier(\*\*rf\_grid\_search.best\_params\_, random\_state=42)

rf\_model.fit(train\_images, train\_labels\_flat)

end\_time = time.time()

training\_time = end\_time - start\_time

print(f"Training time for Random Forest: {training\_time:.2f} seconds")

# Оценка модели Random Forest на тестовых данных

y\_pred\_rf = rf\_model.predict(val\_images)

rf\_accuracy = accuracy\_score(val\_labels\_flat, y\_pred\_rf)

rf\_precision = precision\_score(val\_labels\_flat, y\_pred\_rf, average='weighted', zero\_division=0)

rf\_recall = recall\_score(val\_labels\_flat, y\_pred\_rf, average='weighted', zero\_division=0)  # Добавление zero\_division=0

rf\_f1 = f1\_score(val\_labels\_flat, y\_pred\_rf, average='weighted', zero\_division=0)  # Добавление zero\_division=0

print(f"Random Forest Test Accuracy: {rf\_accuracy:.2f}")

print(f"Random Forest Test Precision: {rf\_precision:.2f}")

print(f"Random Forest Test Recall: {rf\_recall:.2f}")

print(f"Random Forest Test F1 Score: {rf\_f1:.2f}")

# Визуализация результатов (Матрица ошибок и ROC-кривые)

conf\_matrix = confusion\_matrix(val\_labels\_flat, y\_pred\_rf)

sns.heatmap(conf\_matrix, annot=True, fmt="d")

plt.title("Confusion Matrix for Random Forest")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()

# ROC-кривые для Random Forest

y\_pred\_proba\_rf = rf\_model.predict\_proba(val\_images)

fpr, tpr, \_ = roc\_curve(val\_labels\_flat, y\_pred\_proba\_rf[:, 1], pos\_label=1)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'Random Forest (area = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc="lower right")

plt.show()

# Оценка модели мультиклассовой логистической регрессии на тестовых данных

logreg\_model = Pipeline([

    ('scaler', StandardScaler()),

    ('logreg', LogisticRegression(max\_iter=1000, multi\_class='multinomial', solver='lbfgs'))

])

logreg\_model.fit(train\_images, train\_labels\_flat)

y\_pred\_logreg = logreg\_model.predict(val\_images)

logreg\_accuracy = accuracy\_score(val\_labels\_flat, y\_pred\_logreg)

logreg\_precision = precision\_score(val\_labels\_flat, y\_pred\_logreg, average='weighted', zero\_division=0)

logreg\_recall = recall\_score(val\_labels\_flat, y\_pred\_logreg, average='weighted', zero\_division=0)  # Добавление zero\_division=0

logreg\_f1 = f1\_score(val\_labels\_flat, y\_pred\_logreg, average='weighted', zero\_division=0)  # Добавление zero\_division=0

print(f"Logistic Regression Test Accuracy: {logreg\_accuracy:.2f}")

print(f"Logistic Regression Test Precision: {logreg\_precision:.2f}")

print(f"Logistic Regression Test Recall: {logreg\_recall:.2f}")

print(f"Logistic Regression Test F1 Score: {logreg\_f1:.2f}")