Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**
(МИ ВлГУ)

Факультет информационных технологий и радиоэлектроники
Кафедра информационных систем

КУРСОВАЯ РАБОТА

по курсу Теория нейронных сетей

на тему: Разработка ИИ ассистента на базе нейронной сети для обучения карточной игре

Руководитель

к. т. н., доц. каф. ИС
(уч. степень, звание)

Борданов И. А.
(фамилия, инициалы)

(оценка)

(подпись) (дата)

Члены комиссии

Студент ИС - 122
(группа)

Клинцов С. М.
(фамилия, инициалы)

(подпись) (Ф.И.О.)

(подпись) (Ф.И.О.)

(подпись) (дата)

Муром 2025

В ходе курсовой работе было разработано приложение-ассистент для карточной игры:

- обучена и внедрена модель нейронной сети;
- разработан интерфейс взаимодействия с пользователем;
- проведено тестирование и отладка проекта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире карточные игры остаются не только популярным развлечением, но и эффективным инструментом для развития стратегического мышления, памяти и навыков принятия решений. Одной из таких игр является покер — интеллектуальная игра, сочетающая в себе элементы вероятностного анализа, психологии и тактического планирования. Благодаря своей глубине и разнообразию игровых ситуаций покер завоевал огромную популярность среди игроков по всему миру.

Объектом исследования данной работы является разработка виртуального ассистента для карточной игры, способного считывать карты в руках игрока и на основе полученной информации выдавать текущую комбинацию.

Цель работы — закрепление знаний и применение навыков программирования, полученных в ходе изучения курсов по алгоритмам и искусственному интеллекту, для создания интеллектуального помощника, способного улучшить игровой опыт пользователя.

Для достижения поставленной цели необходимо решить следующие задачи:

- изучение правил и механик выбранной карточной игры;
- анализ существующих алгоритмов и стратегий, применяемых в карточных играх;
- проектирование архитектуры приложения с использованием объектно-ориентированного подхода;
- разработка системы анализа игрового состояния;
- реализация графического пользовательского интерфейса и его интеграция с камерой для работы в реальном времени;

– тестирование и оптимизация работы ассистента.

Результатом выполнения работы станет готовое приложение, способное анализировать карты в руках игрока и оценивать вероятности выигрыша в реальном времени.

1 Анализ технического задания

Исходным заданием работы является разработка виртуального ассистента для карточной игры с использованием компьютерного зрения и нейронных сетей. Основными сложностями в реализации стали:

- обработка изображений игрового поля в реальном времени;
- распознавание карт и игровых ситуаций.

Для решения этих задач потребовался тщательный выбор технологий, обеспечивающих высокую производительность, удобство разработки и корректную работу алгоритмов машинного обучения.

1.1 Язык программирования: Python

Языком разработки стал Python, что обусловлено его доминирующим положением в сфере машинного обучения и компьютерного зрения.

По сравнению с Java, Python предлагает более лаконичный синтаксис, обширную экосистему специализированных библиотек и более быстрое прототипирование алгоритмов. Хотя Java демонстрирует лучшую производительность в некоторых сценариях, для данного проекта важнее была доступность готовых решений и простота интеграции компонентов.

1.2 Графический интерфейс и обработка изображений: OpenCV

В качестве основного инструмента для обработки изображений была выбрана библиотека OpenCV (cv2). Её преимущество перед Java-решениями заключается в специализации на задачах компьютерного зрения - она предоставляет простые и эффективные методы для работы с видео и изображениями, поддерживает GPU-ускорение и хорошо интегрируется с нейросетевыми библиотеками. В то время как Java-библиотеки (JavaFX, Swing) больше ориентированы на классические графические интерфейсы и требуют дополнительных надстроек для аналогичного функционала.

1.3 Нейронные сети: Ultralytics (YOLO) и PyTorch

Для распознавания карт использовался фреймворк Ultralytics с моделью YOLOv8, который сочетает высокую скорость работы с хорошей точностью детекции.

В качестве основы для возможного дообучения модели и реализации дополнительных нейросетевых компонентов был выбран PyTorch - один из наиболее гибких и популярных фреймворков для глубокого обучения. Альтернативные решения (TensorFlow) либо требовали больше усилий для настройки, либо уступали в точности.

1.4 Среда разработки: VScode

Средой разработки стал Visual Studio Code - легковесный, но мощный редактор с отличной поддержкой Python, удобной работой с Jupyter Notebooks и широкими возможностями кастомизации. Его универсальность и производительность делают его оптимальным выбором для подобных проектов по сравнению с более специализированными IDE.

2 Проектирование приложения

После анализа требований и возможных архитектурных решений была выбрана следующая структура проекта (см. Рисунок 1). Основной упор сделан на модульность, разделение логики и интерфейса, а также гибкость для возможного расширения функционала.

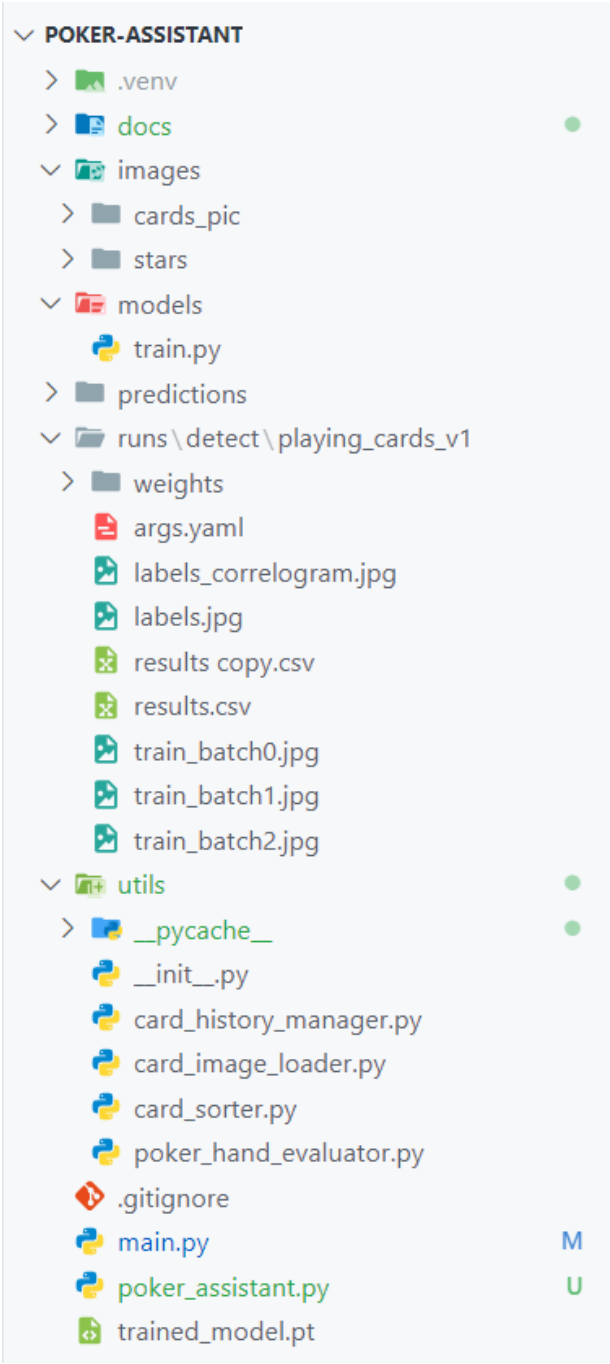

Рисунок 1 – Структура проекта

2.1 Датасет

Для обучения и валидации модели был использован публичный датасет "Playing Cards" с платформы Roboflow, содержащий размеченные изображения игральных карт.

Его структура:

- train/ (21203 изображений);
- valid/ (2020 изображений);
- test/ (1010 изображений).

Каждый сплит содержит:

- изображения (.jpg) 640x640 пикселей;
- текстовые файлы с аннотациями (.txt) для YOLO-модели.

Рисунок 2 - пример изображения из датасета

Именно этот дата сет был выбран из-за его готовности в обучении YOLO-модели (предобработанные изображения и аннотации в native-формате), а также покрытия типичных сценариев карточных игр (разброс, наложение).

2.2 Модель нейронной сети

Во время проектирования модели было встречено множество преград и ошибок, о чём в хронологическом порядке и будет повествоваться далее.

2.2.1 Первая версия модели

Для распознавания игральных карт, после некоторых экспериментов, была разработана свёрточная нейронная сеть (CNN) на базе TensorFlow/Keras:

```
model = models.Sequential([
        # Блок 1
        layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
        layers.BatchNormalization(),
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.2),

        # Блок 2
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.3),

        # Блок 3
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
```

```
            layers.BatchNormalization(),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.4),

            # Классификатор
            layers.Flatten(),
            layers.Dense(256, activation='relu'),
            layers.BatchNormalization(),
            layers.Dropout(0.5),
            layers.Dense(num_classes, activation='softmax')
        ])
        return model
```

Архитектура модели включала три блока свёрток с постепенным увеличением глубины фильтров (32 → 64 → 128) и классификатор:

Каждый блок содержал:

- два слоя Conv2D с ядром 3×3 и активацией ReLU;
- BatchNormalization для ускорения обучения;
- MaxPooling2D (2×2) для уменьшения размерности;
- Dropout (0.2–0.4) для борьбы с переобучением;

Классификатор:

- Flatten для преобразования 3D-тензора в вектор;
- полносвязные слой с 10 нейронами (по числу классов) с Dropout (0.5) и финальный softmax для классификации 52 классов карт.

2.2.2 Анализ прогресса обучения CNN и обоснование перехода на YOLOv8

Рассмотрим процесс обучение сети:

```
Epoch 1: val_accuracy improved from -inf to 0.02273, saving model to
model_checkpoints/cards_model_best.weights.h5

67/67 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 406s 6s/step - accuracy: 0.0655 - loss:
4.2657 - val_accuracy: 0.0227 - val_loss: 8.0566

Epoch 2/30

67/67 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.2381 - loss:
2.7814

Epoch 2: val_accuracy improved from 0.02273 to 0.03846, saving model to
model_checkpoints/cards_model_best.weights.h5

67/67 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 373s 6s/step - accuracy: 0.2387 - loss:
2.7790 - val_accuracy: 0.0385 - val_loss: 8.5553

Epoch 3/30

67/67 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.3989 - loss:
2.0679

Epoch 3: val_accuracy did not improve from 0.03846

67/67 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 372s 6s/step - accuracy: 0.3994 - loss:
2.0657 - val_accuracy: 0.0332 - val_loss: 9.2633

Epoch 4/30

67/67 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.5478 - loss:
1.4467

Epoch 4: val_accuracy did not improve from 0.03846

67/67 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 367s 5s/step - accuracy: 0.5482 - loss:
1.4454 - val_accuracy: 0.0367 - val_loss: 10.0377
```

Результаты обучения свёрточной нейронной сети демонстрируют критические проблемы, сделавшие этот подход нежизнеспособным для поставленной задачи:

– потери на валидации увеличивались (8.05 - 8.55 - 9.26 - 10.03), что свидетельствует о фундаментальном несоответствии архитектуры задачи: сеть не могла выучить полезные признаки для классификации карт;

– разрыв между тренировочной и валидационной метриками указывает на сильное переобучение: модель запоминала тренировочные данные вместо выявления общих признаков.

Были предприняты и другие попытки создания модели, но из-за общей сложности задачи не сколько классификации (с чем и так есть проблемы), но и тем детекции, привели к решению использования оптимизированной YOLOv8.

2.2.3 Модель YOLO

Модель YOLO (You Only Look Once) кардинально изменила подход к детекции объектов, объединив процессы локализации и классификации в единую сквозную архитектуру. Её ключевая особенность - способность анализировать всё изображение за один проход нейросети, что обеспечивает высокую скорость работы при сохранении точности.

Основу работы YOLO составляет разделение входного изображения на сетку ячеек. Каждая ячейка отвечает за предсказание объектов, чьи центры масс попадают в её границы. Для каждого потенциального объекта модель предсказывает координаты ограничивающей рамки, показатель уверенности в наличии объекта и вероятности принадлежности к определённым классам.

Важнейшим этапом обработки является применение алгоритма Non-Maximum Suppression (NMS), который фильтрует дублирующиеся предсказания, оставляя только наиболее точные варианты. Это позволяет устранить проблему множественных срабатываний на один объект.

В итоге всё сводится к двум основополагающим пунктам того, что должна делать сеть:

– точное определения ограничивающей рамки;
– правильная классификация объекта в рамке.

Для проекта выбор YOLOv8n оказался оптимальным решением. Эта модификация сочетает высокую скорость обработки с достойной точностью распознавания. Особенно важно, что архитектура эффективно работает с мелкими объектами, такими как игральные карты, даже в условиях их частичного перекрытия или сложного освещения.

3 Разработка приложения

Далее приступаем к реализации задуманной архитектуры и обучению модели (с исходным кодом можно ознакомиться по ссылке в списке использованных источников).

3.1 Обучение модели

Вот каким образом происходило обучение:

```
model = YOLO('yolov8n.pt')



results = model.train(
    data='datasets/data.yaml',  # путь к вашему конфигурационному файлу
    epochs=100,         # количество эпох
    imgsz=640,          # размер изображения
    batch=16,            # размер батча (уменьшите, если не хватает памяти)
    patience=10,        # ранняя остановка если нет улучшений
    device='0',         # '0' для GPU, 'cpu' для CPU
    name='playing_cards_v1',   # имя эксперимента
    pretrained=True,    # использовать предобученные веса
    optimizer='auto',  # автоматический выбор оптимизатора
    lr0=0.01,           # начальная скорость обучения
    augment=True        # аугментация данных
)
```

Итоговое обучение модели проводилось в течение 65 эпох, далее полное плато. Анализ метрик демонстрирует устойчивую сходимость модели, достигшую высоких показателей детекции.

3.1.1 Динамика ключевых метрик

Потери при обучении:

- box_loss (ошибка локализации) снизилась с 1.45 до 1.02 (-30%);
- cls_loss (ошибка классификации) упала с 3.48 до 0.46 (-87%);
- dfl_loss (распределение размеров bbox) стабилизировалась около 0.98.

Все типы потерь показали аналогичную динамику и на валидации, что подтверждает отсутствие переобучения.

Точность детекции:

- Precision: выросла с 43.1% до 99.9% после 20-й эпохи;
- Recall: достигла 100% к 18-й эпохе;
- mAP@0.5: стабилизировался на уровне 99.5% с 10-й эпохи;
- mAP@0.5-0.95: улучшился с 33.9% до 83.8%, отражая прогресс в детекции при разных порогах IoU.

3.1.2 Критические точки обучения

Эпохи 1-10: Быстрое улучшение метрик за счёт адаптации предобученных признаков.

Эпохи 20-30: Плато по основным метрикам, потребовавшее увеличения аугментации данных.

Эпохи 40-65: Дальнейшая оптимизация за счёт снижения learning rate (с 0.01 до 0.005).

Таким образом получилось обучить высокопроизводительную модель с 83.8% удачных детекций для работы в реальном времени.

3.2 Основной класс PokerAssistant.

Класс PokerAssistant является ядром покерного ассистента и выполняет следующие ключевые функции:

Инициализация компонентов:

- загружает модель YOLO для детекции карт (self.model);
- создает экземпляры вспомогательных классов: CardHistoryManager для стабилизации результатов и CardImageLoader для работы с изображениями карт.

Основной цикл обработки видео:

- захватывает кадры с веб-камеры;
- обрабатывает каждый кадр методом _process_frame();
- отображает интерфейс с детекцией карт и информацией о комбинациях.

3.3 Класс CardHistoryManager

Детекция карт в реальном времени подвержена случайным ошибкам. Чтобы избежать этой проблемы и внести стабильность в отображении данных в GUI был разработан данный класс.

Использует кольцевую очередь (deque) для хранения истории последних детекций. Это позволяет не забивать память данными и оперативно переключаться между разными “запросами” пользователя.

Применяет статистику (Counter) для определения наиболее вероятных карт на экране. Своего рода буфер, который стабилизирует результаты детекции.

Автоматически очищает историю при отсутствии карт более одной секунды.

3.4 Класс PokerHandEvaluator

Иерархически проверяет комбинации от самой сильной (флеш-рояль) до слабой (старшая карта) и, также, на основе математической статистики, выдаёт вероятностную оценка каждой из них.

- поддержка всех стандартных покерных комбинаций;
- проверка валидности входных данных (корректность мастей и рангов);
- оптимизированные методы для проверки специфических комбинаций (стрит, флеш).

3.5 Руководство пользователя

На представленных изображениях показан процесс использования ИИ ассистента для обучения карточной игре. ИИ ассистент помогает пользователю распознавать карты, анализировать комбинации и получать информацию об их редкости и тем самым говоря о вероятности победы.

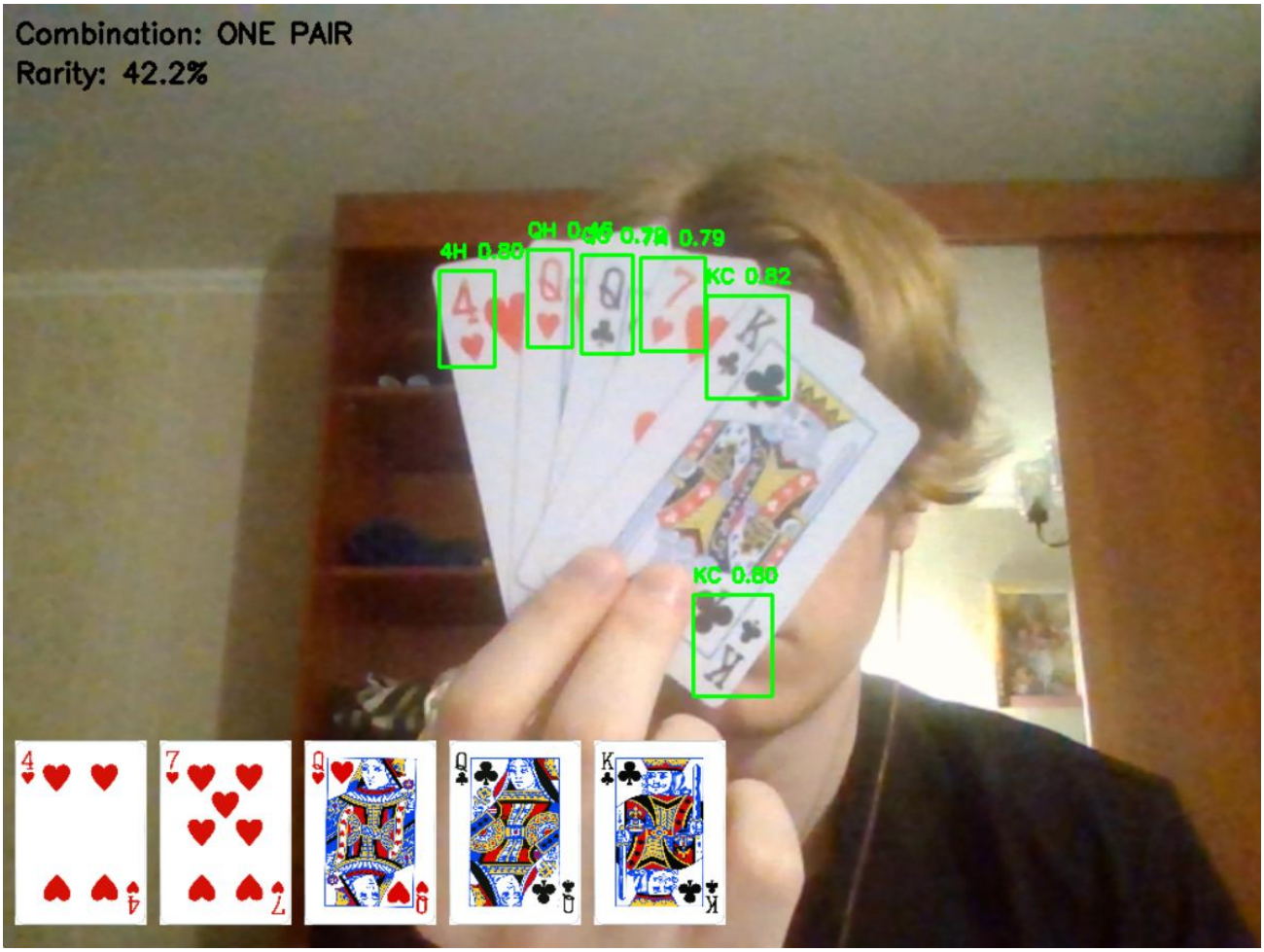

Рисунок 3 - главный экран приложения

Левый нижний угол экрана: отображается текущая "рука" игрока в виде миниатюр карт. Это позволяет пользователю видеть, какие карты у него на руках в данный момент.

Центр экрана: показаны карты, которые ИИ ассистент распознает на изображении. Каждая карта обведена зеленой рамкой, и рядом с ней указано ее значение и вероятность правильного распознавания.

Левый верхний угол экрана: отображается комбинация карт, которую ИИ ассистент анализирует. В данном случае показана комбинация "ONE PAIR" (одна пара) с указанием ее редкости (42.2%).

4 Тестирование

Тестирование программного обеспечения (ПО) является неотъемлемой частью процесса разработки и обеспечивает высокое качество конечного продукта. Тестирование выполняется для проверки соответствия ПО требованиям, выявления ошибок, дефектов и недоработок, а также для проверки функциональности, производительности и безопасности.

4.1 Тестирование считывания комбинаций

1) В руке “Старшая карта”:
   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.
   Результат: соответствует ожидаемому.

2) В руке “Пара”:
   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.
   Результат: соответствует ожидаемому.

3) В руке “Две пары”:
   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.
   Результат: соответствует ожидаемому.

4) В руке “Тройка”:
   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.
   Результат: соответствует ожидаемому.

5) В руке “Стрит”:
   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.

Результат: соответствует ожидаемому.

6) В руке “Флеш”:

   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.

   Результат: соответствует ожидаемому.

7) В руке “Фулл хаус”:

   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.

   Результат: соответствует ожидаемому.

8) В руке “Каре”:

   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.

   Результат: соответствует ожидаемому.

9) В руке “Стрит флеш”:

   Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.

   Результат: соответствует ожидаемому.

10) В руке “Флеш рояль”:

    Ожидаемый результат: правильное отображение комбинации на экране и вероятность её выпадения.

    Результат: соответствует ожидаемому.

11) В руке не 5 карт:

    Ожидаемый результат: просто отображение изображений всех карт в руке.

    Результат: соответствует ожидаемому.

4.2 Тестирование стабильности отображения карт и комбинаций

1) Быстрая смена комбинаций карт (каждые 2 секунды):

Ожидаемый результат: корректное и стабильное отображение как комбинации, так и карт.

Результат: соответствует ожидаемому.

4.3 Тестирование отображения изображений считанных карт

1) В руке какие-то карты:

   Ожидаемый результат: отображение их изображений в нижней левой части приложения.

   Результат: соответствует ожидаемому.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был разработан виртуальный ассистент для карточных игр, использующий современные технологии компьютерного зрения и машинного обучения. В ходе работы достигнуты следующие цели:

Реализована система детекции карт на основе модели YOLOv8, обеспечивающая высокую точность распознавания и работу в реальном времени.

Создана стабилизация результатов через CardHistoryManager и сортировку, которые фильтруют ложные срабатывания и определяет наиболее вероятные карты на столе.

Разработан анализатор покерных комбинаций (PokerHandEvaluator), корректно определяющий силу руки и вероятность выпадения каждой комбинации.

Построен интерактивный интерфейс с использованием OpenCV, отображающий:

- детекцию карт в реальном времени;
- текущую комбинацию и её редкость;
- миниатюры распознанных карт.

Обеспечена модульность системы, позволяющая легко расширять функционал (например, добавлять новые игры или улучшать детекцию).

Разработанный ассистент может быть использован:

- как тренажёр для покера, помогающий игрокам анализировать свои руки;
- в качестве базы для более сложных проектов (например, с интеграцией ИИ для предсказания стратегий).

Таким образом, все поставленные задачи выполнены в полном объёме. Полученный опыт в области компьютерного зрения и машинного обучения открывает возможности для реализации более сложных проектов, таких как распознавание эмоций игроков или анализ стратегий в реальном времени.

Разработанная система демонстрирует, что даже сложные задачи (вроде детекции мелких объектов в динамической среде) могут быть эффективно решены с помощью современных инструментов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Goodfellow I., Bengio Y., Courville A. Deep learning : MIT press, 2016.
2. Bishop C. M. Pattern recognition and machine learning : Springer, 2006.
3. Haykin S. Neural network and learning machines : Pearson Education, 2009.
4. Chollet F. Deep learning with Python : Manning Publication, 2017.
5. Документация по ultralytics [электронный ресурс] : Режим доступа: https://docs.ultralytics.com.
6. Документация по TensorFlow [электронный ресурс] : Режим доступа: https://www.tensorflow.org/tutorials.
7. Датасет для обучения модели [электронный ресурс] : Режим доступа: https://universe.roboflow.com/augmented-startups/playing-cards-ow27d.
8. GitHub с исходным кодом программы [электронный ресурс] : Режим доступа: https://github.com/unfamiliarS/Poker-Assistant.