**Речь для презентации модели на хакатоне «Мониторинг экосистемы через IT решения»**

**(Сопровождается слайдами)**

**Продукт:** Автоматизированная система обнаружения пожаров и дыма на основе анализа изображений.

**Команда:** «ТЕАМ-А»

Добрый день, уважаемые члены жюри и участники хакатона! Мы, команда «ТЕАМ-А», представляем вашему вниманию инновационное IT-решение, направленное на решение критически важной проблемы – своевременное обнаружение пожаров и дыма. В наше время, наряду с изменениями в климате и участившимися пожарами лесов на территории РФ, оперативное реагирование на первоначальные очаги возгорания становится жизненно необходимым. Традиционные методы мониторинга часто оказываются недостаточно эффективными, так как либо требуют больших человеческих ресурсов и времени, либо не обеспечивают своевременность, в результате чего огромные лесные массивы выгорают, что впоследствии ведет к еще более сильным экономическим потерям.

Наше решение основано на машинном зрении, способном различать очаги возгорания и дым. Данное решение может быть применено и интегрировано в действующие системы мониторинга пожарной безопасности, а также с использованием беспилотных воздушных судов (дронов, самолетов) в самых разных областях бизнеса и промышленности, а также в армии и МЧС, где важен оперативный мониторинг с целью принятия своевременных мер. Вот только некоторые из них:

* **Лесное хозяйство и охрана лесов:** Обнаружение лесных пожаров на ранних стадиях, что позволяет оперативно реагировать и минимизировать ущерб. Оперативное выявление стоянок браконьеров и туристов, нарушающих природоохранное законодательство.
* **Нефтегазовая промышленность:** Мониторинг нефтеперерабатывающих заводов, буровых вышек и трубопроводов для предотвращения возгораний и аварий.
* **Энергетика:** Контроль электростанций, подстанций и линий электропередач для своевременного выявления пожаров и предотвращения сбоев в электроснабжении.
* **Производственные предприятия:** Мониторинг цехов, складов и производственных линий для предотвращения пожаров, вызванных технологическими процессами или неисправностью оборудования.
* **Сельское хозяйство:** Обнаружение возгораний на полях и в сельскохозяйственных угодьях, что позволяет предотвратить потерю урожая и распространение огня на соседние территории.
* **Системы раннего оповещения о ЧС:** Интеграция с системами оповещения для оперативного информирования населения о пожарах.
* **Научные исследования и экомониторинг:** Мониторинг городской экосистемы на предмет выбросов в атмосферу и образований смога, влияющего на экологию и здоровье людей.
* **Дата-центры, музеи, складские комплексы.**
* **Армия и МЧС:** Выявление потенциальных угроз, авиационная и космическая разведка.

**Основные преимущества модели машинного зрения по обнаружению пожаров и дыма на основе анализа изображений:**

* Автоматизированная система обнаружения пожаров и дыма.
* Использование алгоритмов компьютерного зрения, способных выявлять очаги возгорания по факту, без использования датчиков дыма и температуры.
* Возможность интеграции с существующими системами мониторинга.

**Взаимодействие членов команды в процессе хакатона:**

В процессе работы над продуктом участники команды взаимодействовали с использованием мессенджеров, а также приложения для управления задачами Week с целью координации и принятия решений. Была использована Kanban-доска.

**Техническая часть реализации модели:**

**Данные для анализа:**

NEMO: Nevada Smoke Detection Dataset (основанный на фотографиях пожаров и дыма в лесах штата Невада, США) – набор данных для задачи обнаружения объектов. Набор данных состоит из 2934 изображений и имеет два раздела: train (тренировочные, 2684 изображения) и val (валидационные, 250 изображений).

Представленная модель машинного обучения представляет собой компиляцию уже имеющихся решений, реализованных на GitHub и Kaggle. Модель была проанализирована и скомпилирована.

**Возможная архитектура работы указанной системы:**

* Сбор данных (изображения из различных источников: камеры наблюдения, дроны, спутники).
* Предобработка изображений (изменение размера, нормализация, аугментация).
* Анализ изображений с помощью модели глубокого обучения.
* Классификация изображений (fire, non\_fire, smoke).
* Оповещение о выявленных возгораниях.

**Ключевые технологии:**

* Python, библиотеки OpenCV, PIL, PyTorch.
* Алгоритмы компьютерного зрения (гистограммы, хеширование).

**Описание технической реализации (3 минуты):**

В основе модели лежит обработка изображений, состоящая из нескольких этапов. Сначала мы собираем изображения из различных источников. Затем изображения проходят этап предобработки, включающий изменение размера, нормализацию и аугментацию. Аугментация используется для увеличения разнообразия обучающей выборки и повышения устойчивости модели к различным условиям съемки. Далее обработанные изображения поступают на вход модели глубокого обучения, которая классифицирует их на три класса: "пожар", "нет пожара" и "дым". Для выявления дубликатов в датасете, что важно для обучения более надежной модели, мы использовали методы хеширования и сравнения гистограмм. Это позволило нам очистить данные и избежать переобучения. Мы провели детальный анализ данных, включая анализ баланса классов, гистограмм цветности, контрастности, яркости и резкости. Этот анализ показал, что классы хорошо различимы по этим характеристикам, что является хорошей предпосылкой для успешной классификации.

**Работа кода условно делится на следующие составляющие:**

* **Импорт библиотек и определение путей к базам данных:** Импортируются необходимые библиотеки: os для работы с файловой системой, pandas для работы с данными в формате DataFrame, base64 для кодирования изображений. Определяются пути к папкам, где хранятся изображения с разными метками (fire, non\_fire, smoke) и неразмеченные изображения (test). Пути указывают на Google Drive.
* **Сбор данных и создание DataFrame:**
  + Сбор данных из размеченных папок.
  + Создание DataFrame для размеченных данных.
  + Сохранение размеченного DataFrame в CSV.
  + Аналогичный процесс для неразмеченных данных (папка test). Код итерируется по папкам с размеченными изображениями (fire, non\_fire, smoke). Для каждого изображения вызывается функция load\_image\_as\_base64, результат добавляется в список data в виде словаря {"class": label, "image": image\_base64}. Из списка data создается DataFrame df. DataFrame сохраняется в CSV-файл dataset.csv. Аналогичные действия выполняются для неразмеченных изображений из папки test, результат сохраняется в test\_dataset.csv. В test\_df\_2 добавляется столбец 'class' со значением np.nan, чтобы структура DataFrame была совместима с df для дальнейшей обработки.
* **Проверка на дубликаты:** Эти блоки кода, включая установку библиотеки imagehash, markdown пояснения и функцию check\_duplicates, предназначены для поиска и удаления дубликатов изображений в датасете. Используются два метода сравнения: хеширование изображений (imagehash – быстрый способ выявления почти идентичных изображений) и сравнение гистограмм (cv2.compareHist – более точный метод, позволяющий выявлять изображения, имеющие схожее цветовое распределение, даже если они немного отличаются). Функция check\_duplicates сравнивает изображения по хешам и гистограммам, формирует списки дубликатов (duplicates) и индексов для удаления (item\_to\_delete). После проверки дубликаты удаляются из DataFrame df.
* **Визуализация примеров изображений:** Отображение примеров изображений из DataFrame df с соответствующими метками классов.
* **EDA (Exploratory Data Analysis) – Анализ баланса классов:** Подсчитывается количество изображений в каждом классе (value\_counts()). Строится круговая диаграмма для визуализации баланса классов. Делается вывод о том, что данные хорошо сбалансированы и можно использовать метрику accuracy.
* **Анализ характеристик изображений (цветность, контрастность, яркость, резкость):** Вычисление и визуализация различных характеристик изображений для каждого класса (fire, non\_fire, smoke): цветность (вычисляются средние гистограммы Продолжаю редактирование текста речи для презентации, начиная с места, где остановился предыдущий ответ:
* …гистограммы для каждого RGB канала. Контрастность: вычисляются гистограммы контраста на основе цветового пространства Lab. Яркость: вычисляется средняя яркость (Luminance) для каждого изображения. Резкость: вычисляется на основе оператора Собеля. Результаты визуализируются в виде графиков и гистограмм.
* **Разделение на обучающую и тестовую выборки:** DataFrame df перемешивается: shuffled\_df = df.sample(frac=1, random\_state=42).reset\_index(drop=True). Данные разделяются на обучающую (train\_df) и тестовую (test\_df) выборки с помощью train\_test\_split (80% на обучение, 20% на тест). random\_state используется для воспроизводимости результатов.
* **Создание Dataset и DataLoader:**
* Функция для декодирования Base64 в изображение PIL: def decode\_base64\_to\_image(base64\_str):
* Определение датасета: class CustomDataset(Dataset): def \_\_init\_\_(self, dataframe, transform=None):
* Пример создания DataLoader:
* Python
* train\_dataset = CustomDataset(train\_df, transform=train\_transforms)
* test\_dataset = CustomDataset(test\_df, transform=test\_transforms)
* test\_dataset\_2 = CustomDataset(test\_df\_2, transform=test\_transforms\_2) # для неразмеченных данных
* train\_loader = DataLoader(train\_dataset, batch\_size=24, shuffle=True)
* test\_loader = DataLoader(test\_dataset, batch\_size=24, shuffle=True)
* test\_loader\_2 = DataLoader(test\_dataset\_2, batch\_size=24, shuffle=True) # для неразмеченных данных
* Определяется кастомный класс CustomDataset. Этот класс отвечает за загрузку и предобработку данных для PyTorch.
* **Тренировка модели:**
* Использовались следующие библиотеки: PyTorch, timm (PyTorch Image Models) (библиотека, предоставляющая готовые к использованию предобученные модели); torchsummary (инструмент для отображения структуры модели); tqdm (библиотека для создания индикаторов прогресса), matplotlib, scikit-learn; torchvision (библиотека для работы с изображениями в PyTorch (преобразования, загрузка)).
* Определение доступного устройства (GPU или CPU) и установка соответствующей переменной device.
* Использование timm.create\_model для загрузки предобученной модели rexnetr\_300. Параметр num\_classes=3 указывает на то, что модель будет классифицировать на 3 класса. Функция train\_setup переносит модель на выбранное устройство (device), устанавливает количество эпох обучения, функцию потерь (CrossEntropyLoss) и оптимизатор (Adam).
* Использование torchsummary.summary для вывода информации о структуре модели, количествах параметров и размере входных данных.
* Функции обучения и тестирования:
* Функция train выполняет один шаг обучения модели на батче данных. Она включает в себя прямой проход (forward pass), вычисление функции потерь, обратное распространение ошибки (backward pass) и шаг оптимизации. Также вычисляются точность и средняя потеря на эпоху. Функция принимает lr\_scheduler для изменения Learning Rate в процессе обучения.
* Функция test выполняет оценку модели на тестовом наборе данных. Она отключает вычисление градиентов (torch.no\_grad()) для экономии памяти и ускорения процесса. Также вычисляются точность и средняя потеря на эпоху.
* Цикл обучения:
* Запуск цикла обучения по эпохам.
* Внутри цикла вызываются функции train и test для обучения и оценки модели на каждой эпохе.
* Вывод информации о потерях и точности на обучающем и тестовом наборах данных, а также текущем значении Learning Rate.
* Сохранение весов модели после каждой эпохи в файл /content/drive/MyDrive/hackathon/last.pth.
* Реализация ранней остановки (early stopping) на основе потери на валидационном наборе данных. Если потеря не уменьшается в течение patience эпох, обучение останавливается. Лучшие веса модели сохраняются в файл /content/drive/MyDrive/hackathon/best.pth.
* Визуализация результатов обучения: Построение графиков потерь и точности на обучающем и валидационном наборах данных с помощью matplotlib.
* Загрузка лучшей модели и оценка на тестовом наборе: Загрузка весов лучшей модели из файла /content/drive/MyDrive/hackathon/best.pth. Оценка модели на тестовом наборе данных и получение предсказанных и истинных меток.
* Оценка качества классификации: Использование sklearn.metrics.classification\_report для вывода отчета о классификации, который включает в себя точность, полноту, F1-меру и поддержку для каждого класса. Вычисление общей точности (accuracy) с помощью accuracy\_score.
* Применение модели к произвольным изображениям (из тестового датасета): Цикл по изображениям из тестового датасета. Добавление размерности батча с помощью unsqueeze(0). Получение логитов (не нормализованных вероятностей) с помощью model(image.to(device)). Преобразование логитов в вероятности с помощью torch.nn.Softmax(dim=1). Визуализация исходного изображения и вероятностей классов с помощью matplotlib.
* **Ключевые моменты:**
* Использование предобученной модели (rexnetr\_300) для ускорения обучения и улучшения результатов.
* Реализация цикла обучения с вычислением потерь и точности.
* Использование ранней остановки для предотвращения переобучения.
* Визуализация результатов обучения и оценка качества классификации.
* Демонстрация применения обученной модели к новым изображениям.
* **Демонстрация на ноутбуке:**
* ДЕМОНСТРАЦИЯ ЧЕРЕЗ LIVE DIGITAL
* **Заключение:**
* Мы уверены, что наша разработка может внести значительный вклад в защиту окружающей среды и предотвращение катастрофических последствий лесных пожаров, а также экономических потерь, связанных с пожарами в сельском хозяйстве, энергетике, нефтегазовой промышленности, на производственных предприятиях. В дальнейшем мы планируем развивать систему, расширяя источники данных и обучая модель на узкоспециализированных данных (спутниковые снимки, снимки промышленных предприятий).