

UL8

Ultralytics YOLO автоматически логирует данные обучения и валидации, такие как метрики `loss`, `mAP`, `precision`, и их можно визуализировать в реальном времени через **TensorBoard**

Запустите TensorBoard на сервере

```
tensorboard --logdir=runs --host=0.0.0.0 --port=6006
```

- `-logdir=runs`: Указывает папку, где хранятся логи (по умолчанию Ultralytics сохраняет их в `runs/`).
- `-host=0.0.0.0`: Позволяет принимать соединения на любом IP-адресе сервера.
- `-port=6006`: Указывает порт для TensorBoard (вы можете указать другой порт, если 6006 занят).

Настройте SSH-порт-форвардинг

На своем локальном компьютере выполните команду для перенаправления порта:

```
bash
КопироватьРедактировать
ssh -L 6006:localhost:6006 user@server_ip
```

- `6006:localhost:6006`:
 - Первый `6006`: порт на вашем локальном компьютере.
 - `localhost:6006`: адрес и порт на сервере, где запущен TensorBoard.
- `user@server_ip`: замените `user` на ваше имя пользователя, а `server_ip` — на IP-адрес вашего сервера.

Теперь ваш локальный порт `6006` будет связан с портом `6006` на сервере.

(2) Автоматический запуск TensorBoard внутри кода

Вы можете добавить запуск TensorBoard прямо в код перед началом тренировки. Для этого выполните следующие шаги:

1. Добавьте запуск TensorBoard в код:

```
python
КопироватьРедактировать
import os
import subprocess

# Запуск TensorBoard из кода
logdir = "runs" # Папка, где YOLO сохраняет логи
tensorboard_cmd = f"tensorboard --logdir={logdir} --host=0.0.0.0 --port=6006"

# Запускаем TensorBoard как subprocess
print("Запуск TensorBoard...")
tensorboard_process = subprocess.Popen(tensorboard_cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

# Убедитесь, что папка для логов существует
os.makedirs(logdir, exist_ok=True)

# Тренировка модели
from ultralytics import YOLO
```

Базовый код тренировки YOLO

```
python
КопироватьРедактировать
from ultralytics import YOLO

# Загрузка модели YOLO (предобученная или новая)
model = YOLO('yolov8n.pt') # Используйте yolov8n.pt для легк
```

ой модели

```
# Обучение модели
```

```
model.train(data='data.yaml', epochs=10, imgsz=640, batch=16)
```

```
# Валидация модели
```

```
metrics = model.val()
```

```
print(f"Validation metrics: {metrics}")
```

```
# Инференс на изображении
```

```
results = model.predict(source='datasets/images/val/image001.jpg', save=True, conf=0.25)
```

Реализация Active Learning

```
import os
```

```
import shutil
```

```
import numpy as np
```

```
from ultralytics import YOLO
```

```
# === Параметры ===
```

```
base_dir = "datasets"
```

```
train_dir = os.path.join(base_dir, "images/train")
```

```
val_dir = os.path.join(base_dir, "images/val")
```

```
unlabeled_dir = os.path.join(base_dir, "images/unlabeled")
```

```
label_dir = os.path.join(base_dir, "labels")
```

```
# Убедитесь, что папки существуют
```

```
os.makedirs(train_dir, exist_ok=True)
```

```
os.makedirs(val_dir, exist_ok=True)
```

```
os.makedirs(unlabeled_dir, exist_ok=True)
```

```
# === Начальный размеченный набор ===
```

```

initial_samples = 10 # Количество начальных примеров
all_images = sorted(os.listdir(unlabeled_dir)) # Все изображения в неразмеченном наборе

# Переносим начальные примеры в train
for i in range(initial_samples):
    shutil.move(os.path.join(unlabeled_dir, all_images[i]), train_dir)

# === Цикл Active Learning ===
model = YOLO('yolov8n.pt') # Загружаем YOLO модель

iterations = 5 # Количество итераций Active Learning
batch_size = 5 # Количество добавляемых данных на каждой итерации

for iteration in range(iterations):
    print(f"\n=== Active Learning Iteration {iteration + 1} ===")

    # Создаем файл data.yaml
    with open('data.yaml', 'w') as f:
        f.write(f"""
train: {train_dir}
val: {val_dir}
nc: 1
names: ['object']
""")

    # Обучаем YOLO
    model.train(data='data.yaml', epochs=5, imgsz=640)

    # Прогоняем неразмеченные данные через модель
    unlabeled_images = sorted(os.listdir(unlabeled_dir))
    uncertainties = []

```

```

for img_name in unlabeled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    results = model.predict(source=img_path, save=False)

    # Оцениваем неопределенность (Uncertainty Sampling)
    predictions = results[0].boxes.conf
    if predictions.numel() > 0:
        uncertainties.append((img_name, 1 - predictions.m
ax().item()))
    else:
        uncertainties.append((img_name, 1)) # Высокая не
определенность, если нет предсказаний

    # Выбираем данные с наибольшей неопределенностью
    uncertainties = sorted(uncertainties, key=lambda x: x[1],
reverse=True)
    selected_samples = uncertainties[:batch_size]

    # Перемещаем выбранные данные в train
    for img_name, _ in selected_samples:
        shutil.move(os.path.join(unlabeled_dir, img_name), tr
ain_dir)

print("\n=== Active Learning Completed ===")

```

Объяснение

1. Инициализация:

- `initial_samples` задает количество изображений, которые будут использоваться как начальный размеченный набор.
- Все остальные данные помещаются в "нераспределенный" набор.

2. Active Learning:

- На каждой итерации модель обучается на текущем размеченном наборе (`train`).
- Остальные данные (`unlabeled`) прогоняются через модель.
- Данные с наибольшей неопределенностью выбираются и добавляются в тренировочный набор.

3. Оценка неопределенности:

- Используется `Uncertainty Sampling` , где наиболее неопределенные данные — это те, для которых модель не уверена (низкая confidence score).

4. Итерации:

- Итерации продолжаются, пока не будут использованы все неразмеченные данные или не достигнуто максимальное число итераций.

Нужно реализовать Active Learning с выборочным прогоном случайного поднабора неразмеченных данных (чтобы ускорить процесс) и последующим добавлением наиболее "неуверенных" примеров для обучения. Давайте всё объединим в один полный код.

Код для Active Learning

```
python
КопироватьРедактировать
import os
import random
import shutil
import numpy as np
from ultralytics import YOLO

# === Шаг 1: Подготовка данных ===
# Пути к папкам
base_dir = "datasets"
```

```

labeled_dir = os.path.join(base_dir, "images/labeled")    # Размеченные данные
unlabeled_dir = os.path.join(base_dir, "images/unlabeled") # Неразмеченные данные
labels_dir = os.path.join(base_dir, "labels") # Аннотации
os.makedirs(labeled_dir, exist_ok=True)
os.makedirs(unlabeled_dir, exist_ok=True)

# Создаем "исходные" данные
all_images = [f"img_{i}.jpg" for i in range(10000)] # Имена изображений
random.shuffle(all_images)

# Первичный раздел данных
initial_samples = 100 # Количество изображений для начального обучения
labeled_images = all_images[:initial_samples]
unlabeled_images = all_images[initial_samples:]

# Размещаем данные в папках
for img_name in labeled_images:
    shutil.copy(f"{base_dir}/source/{img_name}", f"{labeled_dir}/{img_name}")
    shutil.copy(f"{base_dir}/source/{img_name.replace('.jpg', '.txt')}", f"{labels_dir}/{img_name.replace('.jpg', '.txt')}")

for img_name in unlabeled_images:
    shutil.copy(f"{base_dir}/source/{img_name}", f"{unlabeled_dir}/{img_name}")

# === Шаг 2: Параметры Active Learning ===
epochs_per_cycle = 5 # Количество эпох на каждую итерацию
max_epochs = 30 # Общее количество эпох обучения
batch_size = 100 # Количество "неуверенных" изображений, добавляемых за цикл

```

```

sample_size = 500      # Случайное подмножество из неразмечен
ных данных для анализа

# === Модель YOLO ===
model = YOLO('yolov8n.pt')

# Цикл Active Learning
current_epoch = 0

while current_epoch < max_epochs and len(unlabeled_images) >
0:
    print(f"\n=== Active Learning Epoch {current_epoch + 1} =
===")

    # Создаем файл data.yaml
    with open('data.yaml', 'w') as f:
        f.write(f"""
train: {labeled_dir}
val: {labeled_dir}  # Для простоты используем размеченные дан
ные как валидацию
nc: 1
names: ['object']  # Название вашего класса
""")

    # === Шаг 3: Обучение модели ===
    print("Training model on labeled data...")
    model.train(data='data.yaml', epochs=epochs_per_cycle, im
gsz=640)
    current_epoch += epochs_per_cycle

    # === Шаг 4: Выбор подмножества неразмеченных данных ===
    print("Selecting random subset of unlabeled data for anal
ysis...")
    sampled_images = random.sample(unlabeled_images, min(samp
le_size, len(unlabeled_images)))

```



```

# === Шаг 5: Прогон через модель и расчет неопределенност
и ===
uncertainties = []
for img_name in sampled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    results = model.predict(source=img_path, save=False)
    predictions = results[0].boxes.conf # Уверенности пр
едсказаний

    if predictions.numel() > 0:
        uncertainties.append((img_name, 1 - predictions.m
ax().item())) # 1 - max(confidence)
    else:
        uncertainties.append((img_name, 1)) # Высокая не
определенность, если нет предсказаний

# Сортируем по неопределенности
uncertainties = sorted(uncertainties, key=lambda x: x[1],
reverse=True)

# === Шаг 6: Добавление данных в размеченный набор ===
selected_samples = uncertainties[:batch_size]
print(f"Adding {len(selected_samples)} samples to the lab
eled dataset...")

for img_name, _ in selected_samples:
    shutil.move(os.path.join(unlabeled_dir, img_name), o
s.path.join(labeled_dir, img_name))
    shutil.move(os.path.join(unlabeled_dir, img_name.repl
ace('.jpg', '.txt')),
                os.path.join(labels_dir, img_name.replace
('.jpg', '.txt')))
    unlabeled_images.remove(img_name)

# Вывод оставшегося количества неразмеченных данных
print(f"Remaining unlabeled images: {len(unlabeled_image

```

```
s)}}")

print("\n=== Active Learning Completed ===")
```

Объяснение кода

1. Подготовка данных:

- Данные делятся на **размеченные** и **неразмеченные**.
- Первая выборка из 100 изображений используется для начального обучения.

2. Итерации Active Learning:

- Модель обучается на размеченном наборе в течение заданного количества эпох.
- Из неразмеченных данных случайно выбирается подмножество (`sample_size`).
- Эти данные прогоняются через модель, и для них вычисляется **неопределенность**.

3. Выбор новых данных:

- Из выборки выбираются 100 самых "неуверенных" изображений.
- Эти изображения добавляются в размеченный набор для следующего цикла обучения.

4. Остановка обучения:

- Алгоритм останавливается, когда либо достигается максимальное количество эпох, либо заканчиваются неразмеченные данные.

Параметры, которые можно настроить

1. `batch_size`:

- Количество изображений, добавляемых на каждом цикле (например, 100).

2. `sample_size` :

- Объем случайной выборки для анализа неопределенности (например, 500).

3. `epochs_per_cycle` :

- Количество эпох обучения на каждом цикле (например, 5).

4. `max_epochs` :

- Общий лимит на количество эпох (например, 30).

Как следить за сходимостью

- Во время обучения YOLO автоматически сохраняет метрики (`loss` , `mAP50` , `precision` , `recall`) в папке `runs/` . Эти метрики можно отслеживать через **TensorBoard**:

```
bash
КопироватьРедактировать
tensorboard --logdir=runs --host=0.0.0.0 --port=6006
```

Итог

Этот код сочетает активное обучение с выборочным прогоном случайного подмножества данных. Вы можете легко адаптировать его под свои данные, изменяя пути, параметры выборки и количество эпох.

несколько популярных стратегий Active Learning, которые вы можете интегрировать в ваш текущий код. Все они заменяют блок выбора данных по **неуверенности** (в вашем случае `Uncertainty Sampling`).

1. Random Sampling (Случайная выборка)

Описание

- Эта стратегия выбирает данные случайным образом из неразмеченного набора.

- Используется как базовый метод для сравнения с другими стратегиями.

Реализация

Вместо выбора по неопределенности, данные выбираются случайно:

```
python
КопироватьРедактировать
# === Шаг 5: Случайная выборка ===
print("Selecting random samples from unlabeled data...")
selected_samples = random.sample(unlabeled_images, min(batch_
size, len(unlabeled_images)))
```

Когда использовать?

- Когда нет необходимости в сложной логике выбора данных.
- Чтобы сравнить эффективность Active Learning с базовой случайной стратегией.

2. Entropy Sampling (Выборка по энтропии)

Описание

- Вычисляется **энтропия распределения вероятностей предсказаний**:
 $H(p) = -\sum p_i \log(p_i)$
 $H(p) = -\sum p_i \log(p_i)$
- Чем выше энтропия, тем больше модель "не уверена" в предсказании.

Реализация

Вместо `1 - max(confidence)` используем энтропию:

```
python
КопироватьРедактировать
# === Шаг 5: Вычисление энтропии ===
print("Calculating entropy for unlabeled data...")
```

```

uncertainties = []

for img_name in sampled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    results = model.predict(source=img_path, save=False)
    predictions = results[0].boxes.conf # Уверенности предсказаний

    if predictions.numel() > 0:
        probabilities = predictions.numpy()
        entropy = -np.sum(probabilities * np.log(probabilities + 1e-10)) # Сумма -p*log(p)
        uncertainties.append((img_name, entropy))
    else:
        uncertainties.append((img_name, float('inf'))) # Максимальная неопределенность, если нет предсказаний

# Сортируем по энтропии
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
selected_samples = uncertainties[:batch_size]

```

Когда использовать?

- Когда модель возвращает распределения вероятностей (подходит для детекции объектов, где есть confidence scores).
- Когда важен учет всех возможных классов.

3. Margin Sampling (Выборка по разнице вероятностей)

Описание

- Вычисляется разница между двумя самыми высокими вероятностями:
Margin= $p_1 - p_2$

$\text{Margin} = p_1 - p_2$

- Чем меньше разница, тем меньше уверенность модели.

Реализация

```
python
КопироватьРедактировать
# === Шаг 5: Margin Sampling ===
print("Calculating margin for unlabeled data...")
uncertainties = []

for img_name in sampled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    results = model.predict(source=img_path, save=False)
    predictions = results[0].boxes.conf # Уверенности предсказаний

    if predictions.numel() > 1:
        sorted_probs = np.sort(predictions.numpy())[::-1] # Сортируем по убыванию
        margin = sorted_probs[0] - sorted_probs[1] # Разница между двумя наивысшими вероятностями
        uncertainties.append((img_name, 1 - margin)) # Чем меньше разница, тем больше неопределенность
    else:
        uncertainties.append((img_name, 1)) # Максимальная неопределенность, если один класс или нет предсказаний

# Сортируем по неопределенности
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
selected_samples = uncertainties[:batch_size]
```

Когда использовать?

- Когда распределение вероятностей предсказаний модели содержит несколько "близких" классов.

4. Core-Set Sampling (Покрытие набора)

Описание

- Стратегия выбирает примеры, которые увеличивают разнообразие размеченного набора.
- Использует расстояния между embedding векторами (векторами признаков), чтобы найти "разнообразные" данные.

Реализация

Для этого нам нужно извлечь **feature embeddings** из модели (YOLO поддерживает это через API):

```
python
КопироватьРедактировать
from sklearn.metrics.pairwise import euclidean_distances

# === Шаг 5: Core-Set Sampling ===
print("Calculating Core-Set sampling...")
labeled_embeddings = [] # Векторы признаков размеченных данн
ых
unlabeled_embeddings = [] # Векторы признаков неразмеченных
данных

# Извлекаем признаки из размеченных данных
for img_name in labeled_images:
    img_path = os.path.join(labeled_dir, img_name)
    results = model(img_path, save=False, return_features=True)
    # Возвращаем векторы признаков
    labeled_embeddings.append(results.features)

# Извлекаем признаки из случайного подмножества неразмеченных
```

```

данных
for img_name in sampled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    results = model(img_path, save=False, return_features=True)
    # Возвращаем векторы признаков
    unlabeled_embeddings.append(results.features)

# Рассчитываем расстояния между векторами признаков
distances = euclidean_distances(unlabeled_embeddings, labeled_embeddings)
min_distances = np.min(distances, axis=1) # Минимальное расстояние для каждого неразмеченного образца

# Сортируем по расстояниям
uncertainties = [(sampled_images[i], min_distances[i]) for i in range(len(sampled_images))]
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)

# Выбираем самые "далекие" примеры
selected_samples = uncertainties[:batch_size]

```

Когда использовать?

- Когда важно поддерживать разнообразие в размеченном наборе.
- Полезно для сложных данных, где важны редкие или уникальные примеры.

5. Query-by-Committee (Выбор с комитетом моделей)

Описание

- Используются несколько разных моделей или моделей с разной инициализацией.

- Неразмеченные данные прогоняются через все модели, и для каждого примера оценивается степень разногласия в предсказаниях.

Реализация

```
python
КопироватьРедактировать
# === Шаг 5: Query-by-Committee ===
print("Query-by-Committee Sampling...")
uncertainties = []

# Создаем несколько моделей (например, с разными весами)
models = [YOLO('yolov8n.pt'), YOLO('yolov8s.pt'), YOLO('yolov8m.pt')]

for img_name in sampled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    predictions = []

    # Прогоняем через каждую модель
    for committee_model in models:
        results = committee_model.predict(source=img_path, save=False)
        predictions.append(results[0].boxes.conf.numpy())

    # Считаем дисперсию предсказаний
    predictions = np.array(predictions)
    disagreement = np.var(predictions, axis=0).mean() # Средняя дисперсия
    uncertainties.append((img_name, disagreement))

# Сортируем по разногласиям
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
```

```
selected_samples = uncertainties[:batch_size]
```

Когда использовать?

- Когда вы хотите использовать ансамблевые методы для повышения качества выбора данных.

Как интегрировать?

1. Выберите стратегию (например, замените блок `# === Шаг 5 ===` на одну из реализаций выше).
2. Запустите итерации Active Learning.
3. Сравните производительность каждой стратегии (например, с помощью `mAP`, `precision`, или `recall`).

Применение Гауссова распределения в Active Learning

1. Постановка задачи

- Модель генерирует вероятности принадлежности объекта к классам (например, confidence scores в YOLO).
- Предположим, что эти вероятности следуют **Гауссовому распределению**.
- Чем дальше предсказание от центра Гауссова распределения, тем выше неопределенность модели.

2. Как использовать?

- Рассчитываем расстояние предсказания от среднего значения (например, `mu = 0.5` для равномерной уверенности) с использованием функции плотности вероятности Гауссова распределения: $P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

где xxx — confidence score, μ — среднее значение, σ — стандартное отклонение.

- Чем ниже значение $P(x)$, тем больше неопределенность, и такие данные стоит добавить в следующий цикл обучения.

Реализация Гауссового подхода

```
python
КопироватьРедактировать
import numpy as np
from scipy.stats import norm

# === Шаг 5: Гауссово распределение для оценки неопределеннос
ти ===
print("Calculating uncertainties using Gaussian distributio
n...")
uncertainties = []

# Параметры Гауссова распределения
mu = 0.5 # Среднее значение (уверенность модели в "нормально
м" случае)
sigma = 0.1 # Стандартное отклонение (допустимый разброс уве
ренности)

for img_name in sampled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    results = model.predict(source=img_path, save=False)
    predictions = results[0].boxes.conf # Уверенности предск
азаний

    if predictions.numel() > 0:
        confidences = predictions.numpy()
        gauss_uncertainty = -np.sum(norm.pdf(confidences, loc
=mu, scale=sigma)) # Гауссовая плотность
        uncertainties.append((img_name, gauss_uncertainty))
```

```

else:
    uncertainties.append((img_name, float('inf'))) # Максимальная неопределенность, если нет предсказаний

# Сортируем по неопределенности
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
selected_samples = uncertainties[:batch_size]

```

Объяснение кода

1. `mu` и `sigma`:

- `mu = 0.5`: Среднее значение уверенности модели, если все классы равновероятны.
- `sigma = 0.1`: Разброс уверенности, определяет "нормальные" значения.

2. Функция плотности вероятности (PDF):

- Используется `scipy.stats.norm.pdf` для вычисления вероятности $P(x)$ каждого confidence score.

$P(x)P(x)$

- Чем ближе уверенность к среднему μ , тем выше значение $P(x)$.

$\mu \backslash \mu$

$P(x)P(x)$

3. Сортировка по неопределенности:

- Чем ниже значение $P(x)$, тем выше неопределенность.
- $P(x)P(x)$
- Выбираем `batch_size` примеров с максимальной неопределенностью.

Когда использовать Гауссово распределение?

- Когда confidence scores модели имеют примерно **нормальное распределение** (близкое к Гауссовому).
- Если вы хотите учитывать "отклонения" от средних предсказаний модели.
- Полезно для задач, где важно учитывать как слишком низкие, так и слишком высокие уверенности.

Confidence scores (уверенность предсказания) — это вероятность, которую модель присваивает своему предсказанию. В задачах **детекции объектов** (например, YOLO) это значение отражает уверенность модели в том, что найденный объект действительно принадлежит к предсказанному классу.

1. Что такое confidence score в детекции объектов?

- Для каждого обнаруженного объекта YOLO возвращает **bounding box** (координаты рамки), класс объекта и **уверенность** (`confidence score`) — вероятность того, что объект действительно существует в рамке.
- Уверенность обычно находится в диапазоне `[0, 1]`, где:
 - **1.0** означает максимальную уверенность (модель абсолютно уверена в объекте).
 - **0.0** означает отсутствие уверенности (модель считает, что объекта нет).

Пример:

- Для изображения модель может выдать:

```
vbnet
КопироватьРедактировать
Object 1: Class='car', Confidence=0.92, Box=[50, 50, 200, 200]
Object 2: Class='person', Confidence=0.85, Box=[30, 30, 100, 150]
```

2. Есть ли confidence scores в вашей задаче?

Да, в вашей задаче (детекция объектов с использованием YOLO) **confidence scores** присутствуют. В модели YOLO каждая рамка (bounding box), которую она предсказывает, имеет `confidence score`.

Когда вы используете метод `model.predict` в Ultralytics YOLO, результаты включают `confidence scores` для всех обнаруженных объектов. Пример:

```
python
КопироватьРедактировать
results = model.predict(source='path_to_image', save=False)

# Доступ к bounding boxes и confidence scores
for result in results:
    print(result.bboxes.xyxy)  # Координаты рамки (xmin, ymin,
                                xmax, ymax)
    print(result.bboxes.conf)  # Confidence scores для рамок
    print(result.bboxes.cls)   # Классы (идентификаторы классов)
```

3. Как использовать confidence scores в Active Learning?

В Active Learning вы можете использовать **confidence scores** для оценки неопределенности предсказаний. Вот несколько подходов:

(1) Uncertainty Sampling

- Выбирайте примеры с наименьшими `confidence scores`.
- Логика: Если модель имеет низкую уверенность в предсказаниях, эти примеры наиболее "информативны".

Реализация:

```
python
КопироватьРедактировать
uncertainties = []
for img_name in sampled_images:
```

```

img_path = os.path.join(unlabeled_dir, img_name)
results = model.predict(source=img_path, save=False)
confidences = results[0].boxes.conf # Confidence scores
if confidences.numel() > 0:
    uncertainties.append((img_name, 1 - confidences.max(
    ).item())) # Чем ниже уверенность, тем выше неопределенность
else:
    uncertainties.append((img_name, 1)) # Максимальная неопределенность, если нет предсказаний

# Сортируем по неопределенности
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
selected_samples = uncertainties[:batch_size]

```

(2) Entropy Sampling (Энтропия)

- Если у вас есть распределение вероятностей (confidence scores для всех классов), можно вычислить **энтропию**:

$$H(p) = -\sum p_i \log(p_i)$$

$$H(p) = -\sum p_i \log(p_i) \quad H(p) = -\sum p_i \log(p_i)$$

- Чем выше энтропия, тем больше неопределенность.

Реализация:

```

python
КопироватьРедактировать
from scipy.stats import entropy

uncertainties = []
for img_name in sampled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    results = model.predict(source=img_path, save=False)

```

```

confidences = results[0].boxes.conf # Confidence scores
if confidences.numel() > 0:
    probabilities = confidences.numpy()
    ent = entropy(probabilities) # Вычисляем энтропию
    uncertainties.append((img_name, ent))
else:
    uncertainties.append((img_name, float('inf'))) # Максимальная неопределенность, если нет предсказаний

# Сортируем по энтропии
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
selected_samples = uncertainties[:batch_size]

```

(3) Margin Sampling

- Вычисляется разница между двумя наивысшими confidence scores.
- Чем меньше разница, тем меньше уверенность.

Реализация:

```

python
КопироватьРедактировать
uncertainties = []
for img_name in sampled_images:
    img_path = os.path.join(unlabeled_dir, img_name)
    results = model.predict(source=img_path, save=False)
    confidences = results[0].boxes.conf # Confidence scores
    if confidences.numel() > 1:
        sorted_confidences = np.sort(confidences.numpy())[:-1]

        margin = sorted_confidences[0] - sorted_confidences
[1] # Разница между двумя наивысшими значениями
        uncertainties.append((img_name, 1 - margin)) # Чем м

```



```
еньше разница, тем больше неопределенность
    else:
        uncertainties.append((img_name, 1)) # Максимальная неопределенность, если один класс или нет предсказаний

# Сортируем по неопределенности
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
selected_samples = uncertainties[:batch_size]
```

4. Зачем confidence scores в Active Learning?

- **Информативность данных:** `confidence scores` показывают, где модель менее уверена, что позволяет сосредоточиться на сложных примерах.
- **Эффективность разметки:** Вы можете избежать разметки простых данных, сосредоточившись на примерах с низкой уверенностью.

Выбор между хранением изображений в папках или в массиве зависит от ваших задач, доступных ресурсов и сценариев использования. Давайте рассмотрим преимущества и недостатки каждого подхода в контексте вашей задачи (**активного обучения с YOLO**).

1. Хранение изображений в папках

Этот подход наиболее распространён и предпочтителен в задачах компьютерного зрения.

Преимущества:

1. **Совместимость с фреймворками:**

- YOLO и большинство других библиотек для компьютерного зрения ожидают, что данные будут организованы в виде файловой структуры (например, `train/`, `val/`, `unlabeled/`).
- Легко работать с файлами и путями.

2. Масштабируемость:

- Вы можете легко добавлять или удалять изображения, не изменяя структуру данных.
- Большие наборы данных удобнее хранить на диске, а не в памяти.

3. Разделение данных:

- Хранение данных в папках (`labeled/`, `unlabeled/`) упрощает управление размеченными и неразмеченными наборами.

4. Удобство разметки:

- Многие инструменты для разметки (например, LabelImg, Roboflow) работают с файловыми структурами.

Недостатки:

1. Более медленный доступ:

- Если вы хотите обработать все изображения, чтение с диска будет медленнее, чем доступ из памяти.

2. Обработка требует дополнительных шагов:

- Вам нужно загружать изображения из папок в виде массивов, если требуется дальнейшая обработка в памяти.

2. Хранение изображений в массиве (например, NumPy, PyTorch Tensors)

Преимущества:

1. Быстрый доступ:

- Все изображения уже загружены в оперативную память (RAM), что ускоряет их обработку.

- Особенно полезно для моделей, где требуется частая итерация по всем данным.

2. Гибкость в обработке:

- Удобно применять различные преобразования или фильтры ко всем изображениям сразу.

3. Параллельная обработка:

- Легко обрабатывать данные с использованием GPU, если они уже загружены в тензоры.

Недостатки:

1. Большой объём памяти:

- Для больших наборов данных (например, 10k изображений) оперативной памяти может не хватить.
- Например, RGB-изображение размером 640x640 занимает ~1.2 МБ, а 10 000 таких изображений — ~12 ГБ.

2. Трудность работы с размеченными/неразмеченными данными:

- Разделение данных на размеченные и неразмеченные становится сложнее управлять (потребуется отдельный механизм).

3. Сложная интеграция с разметкой и библиотеками:

- Инструменты разметки и большинство фреймворков (например, YOLO) требуют файловую структуру.

3. Рекомендация: комбинированный подход

Основной подход: хранение в папках

- Храните изображения в файловой системе, разделяя их на:
 - Размеченные (`labeled/`)
 - Неразмеченные (`unlabeled/`)
 - Валидационные (`val/`)

- Пример структуры:

```
kotlin
КопироватьРедактировать
datasets/
├── images/
│   ├── labeled/
│   ├── unlabeled/
│   └── val/
├── labels/
│   ├── labeled/
│   └── val/
```

Когда использовать массивы?

- Если у вас есть этапы предварительной обработки, требующие работы со всеми изображениями сразу (например, преобразования или расчёт признаков).
- Храните изображения в массиве только временно, для ускорения таких операций.

Пример использования:

```
python
КопироватьРедактировать
import os
from PIL import Image
import numpy as np

# Загрузка изображений из папки в массив
def load_images_to_array(folder):
    images = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = Image.open(img_path).convert('RGB') # Открываем
```

```

м изображение
        images.append(np.array(img)) # Конвертируем в массив
    return np.array(images)

# Пример использования
images_array = load_images_to_array("datasets/images/labelled/")
print(f"Загружено {len(images_array)} изображений.")

```

4. Пример интеграции с Active Learning

Хранение изображений в папках, но с возможностью временной загрузки подмножества в массив:

```

python
КопироватьРедактировать
import random
import shutil
from ultralytics import YOLO

# Параметры
labeled_dir = "datasets/images/labelled"
unlabeled_dir = "datasets/images/unlabeled"
batch_size = 100 # Количество изображений для добавления

# 1. Выбор случайного подмножества неразмеченных данных
sampled_images = random.sample(os.listdir(unlabeled_dir), batch_size)

# 2. Загрузка подмножества в память
def load_sampled_images(image_list, base_path):
    images = []
    for img_name in image_list:
        img_path = os.path.join(base_path, img_name)

```

```

        img = Image.open(img_path).convert('RGB') # Конверти
руем в RGB
        images.append(np.array(img))
    return np.array(images)

sampled_array = load_sampled_images(sampled_images, unlabeled
_dir)

# 3. Прогон через модель
model = YOLO('yolov8n.pt')
for img_name, img_array in zip(sampled_images, sampled_arra
y):
    results = model.predict(img_array, save=False)

# 4. Перемещение выбранных изображений в размеченную папку
for img_name in sampled_images:
    shutil.move(os.path.join(unlabeled_dir, img_name), os.pat
h.join(labeled_dir, img_name))

```

Итог

- **Для вашей задачи лучше хранить изображения в папках**, так как:
 - YOLO ожидает файловую структуру.
 - Удобнее управлять размеченными и неразмеченными данными.
 - Масштабируемость при большом объёме данных.
- **Массивы (или тензоры)** могут использоваться временно для ускорения операций с подмножествами данных (например, выборка, предсказание, преобразование).

Вот пример кода обучения модели YOLO **без активного обучения**. Здесь мы будем обучать модель на одном фиксированном размеченном наборе данных (например, все данные уже размечены и разделены на тренировочные и валидационные наборы).

Код для обучения модели YOLO без Active Learning

```
python
КопироватьРедактировать
import os
from ultralytics import YOLO

# === Шаг 1: Подготовка данных ===
# Файл data.yaml описывает структуру данных
# Пример содержимого data.yaml:
# train: datasets/images/train
# val: datasets/images/val
# nc: 1 # Количество классов
# names: ['object'] # Название класса

data_yaml = "data.yaml" # Убедитесь, что файл data.yaml существует

# === Шаг 2: Подготовка модели ===
model = YOLO("yolov8n.pt") # Загружаем предобученную YOLOv8n модель (можно заменить на другую)

# === Шаг 3: Обучение модели ===
print("Starting training...")
model.train(
    data=data_yaml,      # Путь к файлу data.yaml
    epochs=30,           # Количество эпох
    imgsz=640,           # Размер изображений (640x640)
    batch=16,            # Размер батча
```

```

        name="fixed_training" # Имя папки для сохранения результатов
    )

# === Шаг 4: Валидация модели ===
print("Validating model...")
metrics = model.val(data=data_yaml) # Валидация на валидационном наборе
print(f"Validation metrics: {metrics}")

# === Шаг 5: Инференс (опционально) ===
print("Running inference on a sample image...")
results = model.predict(source="datasets/images/val/sample_image.jpg", save=True, conf=0.25)

# Результаты инференса сохраняются в папке runs/predict
print("Inference completed. Results saved in 'runs/predict'.")

```

Объяснение

1. Подготовка данных

Создайте файл `data.yaml`, который описывает структуру данных. Например:

```

yaml
КопироватьРедактировать
train: datasets/images/train
val: datasets/images/val

nc: 1 # Количество классов
names: ['object'] # Название вашего класса

```


Разместите данные в следующей структуре:

```
bash
КопироватьРедактировать
datasets/
├── images/
│   ├── train/  # Изображения для обучения
│   └── val/    # Изображения для валидации
├── labels/
│   ├── train/  # Метки для обучения
│   └── val/    # Метки для валидации
```

2. Подготовка модели

- Модель загружается через Ultralytics YOLO. Вы можете использовать предобученные веса (например, `yolov8n.pt`, `yolov8s.pt`) или обучить модель с нуля.

3. Обучение модели

- Используется команда `model.train()`, которая автоматически загружает данные, указанные в `data.yaml`.
- Основные параметры:
 - `epochs`: Количество эпох (например, 30).
 - `imgsz`: Размер изображений, до которых они будут масштабироваться перед подачей в модель.
 - `batch`: Размер батча.
 - `name`: Имя эксперимента для сохранения результатов.

4. Валидация

- Выполняется команда `model.val()`, которая оценивает модель на валидационном наборе и возвращает метрики (`mAP`, `precision`, `recall` и т.д.).

5. Инференс

- Вы можете выполнить предсказание на изображении, используя команду `model.predict()`.
- Параметры:
 - `source` : Путь к изображению (или папке).
 - `save=True` : Сохраняет результаты предсказаний (bounding boxes) в папке `runs/predict`.
 - `conf` : Порог уверенности (например, 0.25).

Вывод

Этот код обучает модель на фиксированном наборе данных без использования Active Learning. Вы можете использовать его для сравнения эффективности модели, обученной с использованием Active Learning. Если вам нужно что-то уточнить или изменить, напишите! 😊

Вы сказали:

как мне теперь сравнить обучение двух нейросетей разными способами

ChatGPT сказал:

ChatGPT

Чтобы сравнить обучение двух нейросетей, обученных разными способами (например, с **Active Learning** и без него), вы можете использовать метрики производительности модели, такие как **mAP** (mean Average Precision), **Precision**, **Recall**, и графики потерь (Loss) на этапах обучения и валидации.

1. Этапы сравнения

1. Сравнение метрик на одном и том же тестовом наборе:

- Вычислите метрики на фиксированном тестовом наборе данных для обеих моделей.
- Основные метрики:
 - **mAP@50** (средняя точность при 50% перекрытии рамок).

- **mAP@50-95** (средняя точность для разных порогов перекрытия рамок).
- **Precision** (точность).
- **Recall** (полнота).

2. Сравнение графиков обучения:

- Постройте графики потерь (Loss) для обеих моделей:
 - **box loss** (потеря рамок).
 - **obj loss** (потеря объекта).
 - **cls loss** (потеря классификации).
- Сравните графики метрик, таких как **mAP**, **Precision**, и **Recall**, на этапах обучения.

3. Сравнение на реальных данных:

- Проверьте предсказания моделей на одних и тех же изображениях и сравните качество предсказаний (например, правильность рамок и классов).

2. Сравнение метрик на тестовом наборе

Обучите обе модели, а затем выполните валидацию на одном и том же тестовом наборе.

Пример кода для получения метрик:

```
python
КопироватьРедактировать
from ultralytics import YOLO

# Загрузка обученных моделей
model_active = YOLO('runs/detect/active_learning/weights/best.pt') # Модель с Active Learning
model_fixed = YOLO('runs/detect/fixed_training/weights/best.pt') # Модель без Active Learning
```

```

# Валидация на тестовом наборе
print("Validating Active Learning model...")
metrics_active = model_active.val(data='data.yaml', split='test')

print("Validating Fixed Training model...")
metrics_fixed = model_fixed.val(data='data.yaml', split='test')

# Вывод метрик
print("\n=== Metrics Comparison ===")
print(f"Active Learning Model - mAP@50: {metrics_active['metrics/mAP50']:.3f}")
print(f"Fixed Training Model - mAP@50: {metrics_fixed['metrics/mAP50']:.3f}")
print(f"Active Learning Model - mAP@50-95: {metrics_active['metrics/mAP50-95']:.3f}")
print(f"Fixed Training Model - mAP@50-95: {metrics_fixed['metrics/mAP50-95']:.3f}")

```

3. Построение графиков

(1) Графики потерь

Во время обучения YOLO автоматически сохраняет данные о потерях (`losses`) и метриках в папке `runs/`. Вы можете извлечь их и построить графики.

Пример кода для построения графиков:

```

python
КопироватьРедактировать
import yaml
import matplotlib.pyplot as plt

# Загрузка данных обучения

```

```

with open("runs/detect/active_learning/results.yaml", "r") as
f:
    active_results = yaml.safe_load(f)

with open("runs/detect/fixed_training/results.yaml", "r") as
f:
    fixed_results = yaml.safe_load(f)

# Данные для графиков
epochs = range(len(active_results['metrics/mAP50']))
mAP_active = active_results['metrics/mAP50']
mAP_fixed = fixed_results['metrics/mAP50']

box_loss_active = active_results['loss/box']
box_loss_fixed = fixed_results['loss/box']

# Построение графиков
plt.figure(figsize=(12, 6))

# График mAP
plt.subplot(1, 2, 1)
plt.plot(epochs, mAP_active, label='Active Learning', color
='blue')
plt.plot(epochs, mAP_fixed, label='Fixed Training', color='or
ange')
plt.xlabel('Epochs')
plt.ylabel('mAP@50')
plt.title('mAP@50 Comparison')
plt.legend()

# График Box Loss
plt.subplot(1, 2, 2)
plt.plot(epochs, box_loss_active, label='Active Learning', co
lor='blue')
plt.plot(epochs, box_loss_fixed, label='Fixed Training', colo
r='orange')

```

```
plt.xlabel('Epochs')
plt.ylabel('Box Loss')
plt.title('Box Loss Comparison')
plt.legend()

plt.tight_layout()
plt.show()
```

(2) Графики Precision и Recall

Аналогично, можно построить графики для **Precision** и **Recall**.

4. Сравнение визуально

Прогон изображений через обе модели

Прогоните одни и те же тестовые изображения через обе модели, чтобы визуально оценить результаты.

Пример:

```
python
КопироватьРедактировать
from ultralytics import YOLO

# Загрузка моделей
model_active = YOLO('runs/detect/active_learning/weights/best.pt')
model_fixed = YOLO('runs/detect/fixed_training/weights/best.pt')

# Прогон тестового изображения
image_path = "datasets/images/test/sample_image.jpg"

print("Running inference on Active Learning model...")
results_active = model_active.predict(source=image_path, save
```

```
=True, name="active_results")

print("Running inference on Fixed Training model...")
results_fixed = model_fixed.predict(source=image_path, save=True, name="fixed_results")

print("Results saved in 'runs/predict'.")
```

- Сравните результаты в папках:

- `runs/predict/active_results`
- `runs/predict/fixed_results`

5. Анализ результатов

После выполнения сравнения:

1. Метрики на тестовом наборе:

- Сравните значения `mAP@50`, `mAP@50-95`, `Precision`, и `Recall`.
- Определите, какая модель лучше справляется с задачей на тестовых данных.

2. Графики:

- Сравните динамику потерь и метрик на этапе обучения.
- Посмотрите, как быстро модели достигают сходимости.

3. Визуализация предсказаний:

- Сравните качество предсказаний на сложных изображениях.
- Посмотрите, как модели справляются с неоднозначными или сложными случаями.

Итог

Сравнение двух моделей включает:

- Сравнение **метрик производительности** на тестовом наборе.
- Построение **графиков обучения и валидации**.
- Оценку **визуального качества предсказаний**.

Понял! Давайте построим графики, где на одной плоскости будут изображены результаты двух моделей. По оси X будет количество эпох, а по оси Y — точность модели (например, **mAP@50**). Вот пример кода:

Код для построения сравнительных графиков

```
python
КопироватьРедактировать
import yaml
import matplotlib.pyplot as plt

# === Шаг 1: Загрузка данных обучения ===
# Загрузка результатов обучения модели с Active Learning
with open("runs/detect/active_learning/results.yaml", "r") as f:
    active_results = yaml.safe_load(f)

# Загрузка результатов обучения модели без Active Learning
with open("runs/detect/fixed_training/results.yaml", "r") as f:
    fixed_results = yaml.safe_load(f)

# === Шаг 2: Подготовка данных для графиков ===
# Извлечение количества эпох
epochs = range(len(active_results['metrics/mAP50']))

# Извлечение mAP@50 для обеих моделей
mAP_active = active_results['metrics/mAP50']
mAP_fixed = fixed_results['metrics/mAP50']
```



```
# === Шаг 3: Построение графика ===
plt.figure(figsize=(10, 6))

# График для модели с Active Learning
plt.plot(epochs, mAP_active, label='Active Learning', color='blue', linestyle='--', marker='o')

# График для модели без Active Learning
plt.plot(epochs, mAP_fixed, label='Fixed Training', color='orange', linestyle='-', marker='s')

# Настройки графика
plt.title("Comparison of Model Accuracy Over Epochs", fontsize=16)
plt.xlabel("Epochs", fontsize=14)
plt.ylabel("mAP@50", fontsize=14)
plt.legend(fontsize=12)
plt.grid(True)
plt.tight_layout()

# Показ графика
plt.show()
```

Объяснение кода

1. Загрузка результатов обучения:

- Данные о метриках (например, `mAP@50`) берутся из файлов `results.yaml`, которые автоматически создаются YOLO в процессе обучения.

2. Извлечение данных для графиков:

- `active_results['metrics/mAP50']` — список значений mAP@50 для каждой эпохи модели с Active Learning.
- `fixed_results['metrics/mAP50']` — аналогично для модели без Active Learning.

3. Построение графиков:

- Используются два графика:
 - Один для модели с Active Learning (синий пунктир).
 - Второй для модели без Active Learning (оранжевая линия).

4. Настройка графика:

- Заголовок, подписи осей, легенда и сетка добавлены для улучшения визуализации.

Пример результата

График покажет, как точность моделей (mAP@50) изменяется в зависимости от количества эпох. Например:

- Синяя линия (Active Learning) может показывать более быстрое увеличение точности.
- Оранжевая линия (Fixed Training) может демонстрировать более медленное улучшение.

Дополнительно: Сравнение других метрик

Если вы хотите сравнить **Precision**, **Recall** или **mAP@50-95**, нужно заменить ключи `metrics/mAP50` в коде на соответствующие ключи из `results.yaml`:

- **Precision:** `metrics/precision`
- **Recall:** `metrics/recall`
- **mAP@50-95:** `metrics/mAP50-95`

Пример:

```
python
КопироватьРедактировать
precision_active = active_results['metrics/precision']
precision_fixed = fixed_results['metrics/precision']
```

Затем можно добавить новые линии на график.

Если у вас будут дополнительные вопросы или нужен более сложный график, дайте знать! 😊

Вы сказали:

а как в процессе обучения формируется mAP@50. то есть что это за равнение такое и как оно появляется в процессе обучения

ChatGPT сказал:

ChatGPT

mAP@50 (mean Average Precision at IoU=50%) — это ключевая метрика в задачах детекции объектов. Она измеряет, насколько хорошо модель обнаруживает объекты на изображении. Давайте разберёмся, что это такое, как вычисляется и как появляется в процессе обучения.

1. Что такое mAP@50?

mAP — это среднее значение **AP (Average Precision)** по всем классам. AP, в свою очередь, оценивает качество предсказаний для одного класса на основе Precision и Recall.

- **Precision (точность):**

$$\text{Precision} = \frac{TP}{TP + FP}$$

- TP (True Positive): Правильные детекции (модель правильно обнаружила объект).
- FP (False Positive): Неправильные детекции (модель нашла объект там, где его нет).

- **Recall (полнота):**

$$\text{Recall} = \frac{TP}{TP + FN}$$

- FN (False Negative): Пропущенные объекты (модель не нашла объект, который есть на изображении).

- **IoU (Intersection over Union):**

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

- IoU измеряет перекрытие между предсказанной рамкой и истинной рамкой (ground truth).
- **mAP@50** означает, что $\text{IoU} \geq 0.50$ используется для определения True Positive.

2. Как формируется mAP в процессе обучения?

mAP рассчитывается в несколько этапов:

Шаг 1: Предсказания модели

Для каждого изображения модель:

1. Генерирует список обнаруженных объектов с координатами рамок, confidence scores и классами.
2. Confidence score используется для ранжирования предсказаний (чем выше, тем выше приоритет предсказания).

Шаг 2: Сопоставление предсказаний с ground truth

Для каждого класса:

1. **Сортируем предсказания по confidence scores.**
2. Сопоставляем предсказания с истинными рамками (`ground truth`) на основе IoU:
 - Если $\text{IoU} \geq 0.50$, то предсказание считается TP.
 - Если $\text{IoU} < 0.50$ или объект уже найден, предсказание считается FP.

Шаг 3: Вычисление Precision и Recall

После сопоставления вычисляем Precision и Recall для каждого порога confidence (обычно шагом 0.01):

- Например, при $\text{confidence} \geq 0.9$ модель может находить меньше объектов (Recall ниже), но с высокой точностью (Precision выше).
- При $\text{confidence} \geq 0.1$ модель находит почти все объекты (Recall выше), но появляются ложные срабатывания (Precision ниже).

Шаг 4: Построение Precision-Recall (P-R) кривой

На основе значений Precision и Recall строится P-R кривая для каждого класса.

Шаг 5: Вычисление AP (Average Precision)

AP вычисляется как площадь под P-R кривой:

$$AP = \int_0^1 \text{Precision}(R) dR$$

Шаг 6: Вычисление mAP

mAP рассчитывается как среднее AP по всем классам:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i$$

где N — количество классов.

3. Как mAP обновляется в процессе обучения?

- **На каждой эпохе:**
 1. YOLO выполняет валидацию на проверочных данных.
 2. Вычисляются Precision, Recall, IoU и AP для каждого класса.
 3. Рассчитывается mAP как среднее AP по всем классам.
 4. Значение mAP сохраняется для последующего анализа.
 - **Тренд mAP:**
 - **В начале обучения:** mAP низкий, так как модель ещё плохо определяет объекты.
 - **В процессе обучения:** mAP увеличивается, так как модель становится лучше на тренировочных данных.
 - **При переобучении:** mAP на валидации может начать снижаться, если модель начинает запоминать тренировочные данные.
-

4. Пример вычисления mAP@50

Допустим, у нас есть два класса: `car` и `person`.

Для класса `car`:

- P-R кривая выглядит так:

Precision : [1.0, 0.9, 0.8], Recall : [0.1, 0.4, 0.8]

- AP (`car`) = площадь под P-R кривой = **0.67**

Для класса `person`:

- P-R кривая:

Precision : [0.9, 0.8, 0.7], Recall : [0.2, 0.5, 0.9]

- AP (`person`) = площадь под P-R кривой = **0.72**

mAP@50:

$$mAP@50 = \frac{AP(car) + AP(person)}{2} = \frac{0.67 + 0.72}{2} = 0.695$$

5. Как посмотреть mAP@50 в YOLO?

При обучении YOLO автоматически вычисляет mAP@50 и выводит его в логах. Например:

```
kotlin
КопироватьРедактировать
Epoch 10/30
train: ... val: ...
mAP@50: 0.695
```

Вы также можете посмотреть mAP в файле `results.yaml`:

```
yaml
КопироватьРедактировать
```

```
metrics:
  mAP50: 0.695
  mAP50-95: 0.432
  precision: 0.812
  recall: 0.730
```

6. Как понять mAP?

- **Высокий mAP@50** (например, >0.7): Модель хорошо справляется с задачей, правильно определяя объекты.
- **Низкий mAP@50** (например, <0.5): Модель плохо находит объекты, возможно, требуется больше данных или улучшение архитектуры.

Если хотите углубиться в какой-то конкретный аспект (например, детализация P-R кривых), дайте знать! 😊

Вы сказали:

то есть мы ее получаем во время predict?

ChatGPT сказал:

ChatGPT

Нет, mAP (mean Average Precision) рассчитывается во время валидации (`val`), а не во время предсказания (`predict`). Давайте разберемся.

1. Когда рассчитывается mAP?

- **mAP вычисляется на этапе валидации (`val`),** когда модель оценивается на валидационном или тестовом наборе данных.
- Это происходит после завершения каждой эпохи обучения или если вы явно вызвали метод `model.val()`.

Пример валидации:


```
python
КопироватьРедактировать
from ultralytics import YOLO

# Загрузка обученной модели
model = YOLO('runs/detect/train/weights/best.pt')

# Валидация модели
metrics = model.val(data='data.yaml', split='val') # split
='val' указывает на валидационный набор

# Вывод метрик
print(f"mAP@50: {metrics['metrics/mAP50']}")
print(f"mAP@50-95: {metrics['metrics/mAP50-95']}")
```

На этапе валидации YOLO:

1. Прогоняет изображения валидационного набора через модель (то есть выполняет предсказание).
2. Сопоставляет предсказания с истинными метками (ground truth).
3. Рассчитывает Precision, Recall, IoU, AP для каждого класса и усредняет их для вычисления mAP.

2. Что происходит на этапе predict?

На этапе `predict` YOLO:

1. Выполняет предсказания на предоставленных изображениях.
2. Возвращает:
 - Координаты рамок (bounding boxes),
 - Confidence scores,
 - Идентификаторы классов для обнаруженных объектов.

Однако на этапе `predict` **mAP не вычисляется**, так как у модели нет информации о ground truth (истинных метках). `predict` используется для применения модели на новых данных.

Пример:

```
python
КопироватьРедактировать
from ultralytics import YOLO

# Загрузка модели
model = YOLO('runs/detect/train/weights/best.pt')

# Прогон изображения через модель
results = model.predict(source='image.jpg', conf=0.25)

# Вывод предсказаний
for r in results:
    print(r.boxes.xyxy)  # Координаты рамок
    print(r.boxes.conf)  # Уверенности предсказаний
    print(r.boxes.cls)   # Классы
```

3. Разница между `predict` и `val`

| Характеристика | <code>predict</code> | <code>val</code> |
|---------------------|-----------------------------------|--------------------------------------|
| Цель | Применение модели на новых данных | Оценка модели |
| Данные | Нет меток (только изображения) | Метки доступны (валидационный набор) |
| Результат | Предсказания (bounding boxes) | Метрики (mAP, Precision, Recall) |
| Когда использовать? | Для тестирования или предсказания | Для оценки производительности модели |

4. Как mAP формируется в процессе обучения?

Во время обучения YOLO автоматически вызывает `val()` после каждой эпохи, чтобы:

1. Оценить качество модели на валидационном наборе.
2. Сохранить метрики (`mAP`, `Precision`, `Recall`) в логах.

5. Если вам нужно сравнить mAP для двух моделей

1. Убедитесь, что у вас есть общий тестовый набор данных.
2. Выполните валидацию обеих моделей:

```
python
КопироватьРедактировать
model1 = YOLO('path_to_model1.pt')
model2 = YOLO('path_to_model2.pt')

metrics1 = model1.val(data='data.yaml', split='test')
metrics2 = model2.val(data='data.yaml', split='test')

print(f"Model 1 mAP@50: {metrics1['metrics/mAP50']}")
print(f"Model 2 mAP@50: {metrics2['metrics/mAP50']}")
```

Итог

- `mAP` вычисляется на этапе `val`, а не `predict`.
- `predict` используется для применения модели, но метрики на этом этапе не рассчитываются.
- Для оценки модели и получения mAP необходимо выполнить валидацию.

Если у вас остались вопросы, например, как настроить тестовый набор или что-то ещё, напишите! 😊

Вы сказали:

так а у меня модель с активным обучением и без него не перезапишут друг друга? Я же их. по очереди запущу а потом буду сравнивать а что

ChatGPT сказал:

ChatGPT

Не переживайте, модели с активным обучением и без него не перезапишут друг друга, если вы правильно зададите параметры для сохранения результатов. Вот, как это работает:

1. Где YOLO сохраняет результаты?

По умолчанию YOLO сохраняет результаты каждого запуска в папке `runs/`:

- Для задачи детекции — это `runs/detect/`.
- Внутри этой папки для каждого запуска создаётся уникальная папка на основе параметра `name`.

Пример:

- Если вы запускаете команду обучения с `name=train`, результаты сохранятся в `runs/detect/train`.
- Если вы используете `name=train2`, результаты сохранятся в `runs/detect/train2`.

2. Как избежать перезаписи результатов?

Чтобы избежать перезаписи:

1. **Убедитесь, что у каждой модели своё уникальное имя.**
2. Укажите имя с помощью параметра `name` при вызове `model.train()`.

Пример для двух моделей:

```
python
КопироватьРедактировать
from ultralytics import YOLO

# Модель с активным обучением
model_active = YOLO("yolov8n.pt")
```

```

model_active.train(
    data="data.yaml",
    epochs=30,
    imgsz=640,
    batch=16,
    name="active_learning"
)

# Модель без активного обучения
model_fixed = YOLO("yolov8n.pt")
model_fixed.train(
    data="data.yaml",
    epochs=30,
    imgsz=640,
    batch=16,
    name="fixed_training"
)

```

3. Где искать результаты?

- Модель с активным обучением: `runs/detect/active_learning/`
- Модель без активного обучения: `runs/detect/fixed_training/`

4. Что делать, если не указать `name` ?

Если вы не укажете параметр `name`, YOLO автоматически назовёт папку `train`, и последующий запуск может перезаписать предыдущие результаты.

Чтобы избежать этого, вы можете:

1. **Явно указать уникальные имена.**
2. Включить параметр `exist_ok=True`, чтобы YOLO создало новую папку автоматически, не перезаписывая старую:

```
python
КопироватьРедактировать
model.train(data="data.yaml", epochs=30, name="train", exist_ok=True)
```

5. Как сравнивать модели после обучения?

После обучения вы можете найти результаты обеих моделей в их папках. В каждой папке находятся:

- `results.yaml` — Метрики обучения и валидации.
- `weights/best.pt` — Лучшие веса модели.
- `val_batch*.jpg` — Визуализация предсказаний на валидационном наборе.

Пример сравнения:

```
python
КопироватьРедактировать
# Загрузка моделей
model_active = YOLO("runs/detect/active_learning/weights/best.pt")
model_fixed = YOLO("runs/detect/fixed_training/weights/best.pt")

# Валидация обеих моделей
metrics_active = model_active.val(data="data.yaml", split="test")
metrics_fixed = model_fixed.val(data="data.yaml", split="test")

# Вывод mAP для сравнения
print(f"Active Learning mAP@50: {metrics_active['metrics/mAP50']:.3f}")
print(f"Fixed Training mAP@50: {metrics_fixed['metrics/mAP50']:.3f}")
```

```
0']:.3f}"))
```

6. Как организовать папки для удобства?

Чтобы упростить управление результатами, вы можете использовать следующую структуру:

```
markdown
КопироватьРедактировать
runs/
  └─ detect/
    │   └─ active_learning/
    │       └─ results.yaml
    │       └─ weights/
    └─ fixed_training/
        └─ results.yaml
        └─ weights/
```

Вывод

- Используйте параметр `name` при вызове `model.train()` для уникального имени эксперимента.
- Не забывайте включать `exist_ok=True`, чтобы избежать перезаписи.
- Сравнивайте модели по их метрикам (`mAP`, `Precision`, `Recall`) из соответствующих папок.

Давайте упростим ваш код и интегрируем разделение данных на обучающие и валидационные для **Active Learning** и **обычного обучения**. В результате вы

получите возможность построить графики обучения для обеих моделей на одной плоскости.

1. Шаги для реализации

1. Добавить разделение данных на тренировочные и валидационные:

- Выделяем **20% данных** для валидации, а остальные используются для обучения.
- Сохраняем их в папках `train/` и `val/`.

2. Интегрировать в Active Learning:

- Обновляем `train` в `data.yaml` при добавлении новых данных.
- `val` остаётся неизменным, чтобы отслеживать метрики на одном и том же валидационном наборе.

3. Обычное обучение:

- Обучаем модель на всём размеченном наборе (80% данных), используя тот же `val`.

4. Сравнение:

- Построим графики mAP@50 по эпохам для обеих моделей.
-

2. Полный код

```
python
КопироватьРедактировать
import os
import random
import shutil
import yaml
import matplotlib.pyplot as plt
from ultralytics import YOLO

# === Шаг 1: Разделение данных ===
def split_data(all_images_dir, all_labels_dir, train_dir, val
```



```

_dir, train_label_dir, val_label_dir, val_split=0.2):
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)
    os.makedirs(train_label_dir, exist_ok=True)
    os.makedirs(val_label_dir, exist_ok=True)

    images = [f for f in os.listdir(all_images_dir) if f.endswith(
'.jpg')]
    random.shuffle(images)

    val_count = int(len(images) * val_split)
    val_images = images[:val_count]
    train_images = images[val_count:]

    for img_name in train_images:
        shutil.copy(os.path.join(all_images_dir, img_name), o
s.path.join(train_dir, img_name))
        shutil.copy(os.path.join(all_labels_dir, img_name.repl
ace('.jpg', '.txt')),
                    os.path.join(train_label_dir, img_name.repl
ace('.jpg', '.txt')))

    for img_name in val_images:
        shutil.copy(os.path.join(all_images_dir, img_name), o
s.path.join(val_dir, img_name))
        shutil.copy(os.path.join(all_labels_dir, img_name.repl
ace('.jpg', '.txt')),
                    os.path.join(val_label_dir, img_name.repl
ace('.jpg', '.txt')))

    print(f"Train images: {len(train_images)}, Val images: {l
en(val_images)}")
    return len(train_images), len(val_images)

# Разделяем данные
all_images_dir = "datasets/images/all"

```

```

all_labels_dir = "datasets/labels/all"
train_dir = "datasets/images/train"
val_dir = "datasets/images/val"
train_label_dir = "datasets/labels/train"
val_label_dir = "datasets/labels/val"

split_data(all_images_dir, all_labels_dir, train_dir, val_dir,
train_label_dir, val_label_dir)

# === Шаг 2: Подготовка Active Learning ===
def active_learning(model_path, train_dir, val_dir, train_label_dir,
val_label_dir, iterations=5, batch_size=100):
    model = YOLO(model_path)
    mAP_history = []

    for iteration in range(iterations):
        print(f"\n=== Active Learning Iteration {iteration + 1} ===")

        # Обучение модели
        model.train(
            data="data.yaml",
            epochs=1,
            imgsz=640,
            batch=16,
            name="active_learning",
            resume=True
        )

        # Валидация и сохранение mAP
        metrics = model.val(data="data.yaml", split="val")
        mAP_history.append(metrics["metrics/mAP50"])

        # Выборка самых неуверенных данных
        all_unlabeled_images = [f for f in os.listdir(train_dir) if f not in os.listdir(val_dir)]

```

```

        sampled_images = random.sample(all_unlabeled_images,
min(batch_size, len(all_unlabeled_images)))

        # Добавляем их в размеченные данные
        for img_name in sampled_images:
            shutil.move(os.path.join(all_images_dir, img_name),
os.path.join(train_dir, img_name))
            shutil.move(os.path.join(all_labels_dir, img_name.replace('.jpg', '.txt')),
os.path.join(train_label_dir, img_name.replace('.jpg', '.txt')))

        return mAP_history

# === Шаг 3: Обучение модели без Active Learning ===
def train_fixed_model(model_path, train_dir, val_dir, epochs=
10):
    model = YOLO(model_path)
    mAP_history = []

    for epoch in range(epochs):
        print(f"\n=== Training Fixed Model Epoch {epoch + 1}
===")

        # Обучение модели
        model.train(
            data="data.yaml",
            epochs=1,
            imgsz=640,
            batch=16,
            name="fixed_training",
            resume=True
        )

        # Валидация и сохранение mAP
        metrics = model.val(data="data.yaml", split="val")

```

```

        mAP_history.append(metrics["metrics/mAP50"])

    return mAP_history

# === Шаг 4: Построение графиков ===
def plot_results(active_history, fixed_history):
    epochs = range(1, len(active_history) + 1)

    plt.figure(figsize=(10, 6))
    plt.plot(epochs, active_history, label="Active Learning",
             color="blue", marker="o", linestyle="--")
    plt.plot(epochs, fixed_history, label="Fixed Training", color="orange",
             marker="s", linestyle="-")

    plt.title("Comparison of Active Learning and Fixed Training",
              fontsize=16)
    plt.xlabel("Epochs", fontsize=14)
    plt.ylabel("mAP@50", fontsize=14)
    plt.legend(fontsize=12)
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# === Шаг 5: Запуск экспериментов ===
# Обучение модели с Active Learning
active_mAP_history = active_learning("yolov8n.pt", train_dir,
                                     val_dir, train_label_dir, val_label_dir, iterations=10)

# Обучение модели без Active Learning
fixed_mAP_history = train_fixed_model("yolov8n.pt", train_dir, val_dir, epochs=10)

# Построение графиков
plot_results(active_mAP_history, fixed_mAP_history)

```

Объяснение кода

1. Разделение данных:

- Функция `split_data()` делит набор данных на `train` и `val`, сохраняя их в соответствующих папках.

2. Active Learning:

- На каждой итерации добавляются `batch_size` самых "неуверенных" изображений в размеченный набор.
- После каждой итерации происходит валидация и сохраняется значение `mAP@50`.

3. Обычное обучение:

- Модель обучается на фиксированном тренировочном наборе, и после каждой эпохи также сохраняется `mAP@50`.

4. Построение графиков:

- Результаты `mAP@50` для обеих моделей строятся на одной плоскости, чтобы вы могли легко сравнить сходимость.

Итог

- Этот код позволяет разделить данные, обучить модели с Active Learning и без него, а затем построить графики их сходимости.
- На графике вы увидите, какая модель быстрее обучается или достигает более высокого качества.

Если что-то осталось неясным или вы хотите улучшить/адаптировать код, напишите! 😊

```

from ultralytics import YOLO
import numpy as np
import os
from shutil import copyfile

# === Шаг 1: Подготовка COCO128 ===
# Загружаем COCO128 через Ultralytics (встроенный датасет)
coco128_yaml = "coco128.yaml" # Путь к встроенному датасету Ultralytics
model = YOLO('yolov8n.pt') # Загружаем предобученную модель

# === Шаг 2: Настройка для Active Learning ===
# Параметры
iterations = 5 # Количество итераций AL
initial_samples = 10 # Сколько примеров взять в начальном размеченном наборе
batch_size = 5 # Сколько примеров добавлять за итерацию

# Папки для Active Learning
base_dir = "active_learning_coco128"
os.makedirs(base_dir, exist_ok=True)
train_dir = os.path.join(base_dir, "train")
val_dir = os.path.join(base_dir, "val")
unlabeled_dir = os.path.join(base_dir, "unlabeled")
os.makedirs(train_dir, exist_ok=True)
os.makedirs(val_dir, exist_ok=True)
os.makedirs(unlabeled_dir, exist_ok=True)

# Разделяем COCO128 на размеченные и неразмеченные данные
images_dir = "datasets/coco128/images/train2017"
labels_dir = "datasets/coco128/labels/train2017"
all_images = sorted(os.listdir(images_dir))
all_labels = sorted(os.listdir(labels_dir))

# Берем небольшую часть данных для начала (начальный размеченный набор)
for i in range(initial_samples):
    copyfile(os.path.join(images_dir, all_images[i]), os.path.join(train_dir, all_images[i]))
    copyfile(os.path.join(labels_dir, all_labels[i]), os.path.join(train_dir, all_labels[i]))

# Остальные данные считаем "нераспределенными"
for i in range(initial_samples, len(all_images)):
    copyfile(os.path.join(images_dir, all_images[i]), os.path.join(unlabeled_dir, all_images[i]))

```

```

# Остальные данные считаем "нераспределенными"
for i in range(initial_samples, len(all_images)):
    copyfile(os.path.join(images_dir, all_images[i]), os.path.join(unlabeled_dir, all_images[i]))
    copyfile(os.path.join(labels_dir, all_labels[i]), os.path.join(unlabeled_dir, all_labels[i]))

# === Шаг 3: Цикл Active Learning ===
for iteration in range(iterations):
    print(f"\n=== Active Learning Iteration {iteration + 1} ===")

    # Создаем файл data.yaml для текущей итерации
    data_yaml = os.path.join(base_dir, f"data_iter_{iteration + 1}.yaml")
    with open(data_yaml, "w") as f:
        f.write(f"""
train: {train_dir}
val: {val_dir}
nc: 80
names: {list(range(80))}
        """)

    # Обучаем YOLO на текущем размеченном наборе
    model.train(data=data_yaml, epochs=5, imgsz=640)

    # === Шаг 4: Выбор данных для разметки ===
    unlabeled_images = sorted(os.listdir(unlabeled_dir))
    uncertainties = []

    # Прогоняем неразмеченные данные через модель и считаем неопределенность
    for img_name in unlabeled_images:
        img_path = os.path.join(unlabeled_dir, img_name)
        results = model.predict(source=img_path, save=False)
        predictions = results[0].boxes.conf # Уверенности предсказаний
        if predictions.numel() > 0:
            uncertainties.append((img_name, 1 - predictions.max().item())) # 1 - max(conf)
        else:
            uncertainties.append((img_name, 1)) # Нет предсказаний, высокая неопределенность

    # Сортируем по неопределенности (от наибольшей к наименьшей)
    uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
    selected_samples = uncertainties[:batch_size]

```

```

# Сортируем по неопределенности (от наибольшей к наименьшей)
uncertainties = sorted(uncertainties, key=lambda x: x[1], reverse=True)
selected_samples = uncertainties[:batch_size]

# Перемещаем выбранные данные в размеченный набор
for img_name, _ in selected_samples:
    label_name = img_name.replace(".jpg", ".txt")
    copyfile(os.path.join(unlabeled_dir, img_name), os.path.join(train_dir, img_name))
    copyfile(os.path.join(unlabeled_dir, label_name), os.path.join(train_dir, label_name))
    os.remove(os.path.join(unlabeled_dir, img_name))
    os.remove(os.path.join(unlabeled_dir, label_name))

# Оцениваем текущую модель на валидационном наборе
metrics = model.val(data=data_yaml)
print(f"Iteration {iteration + 1} - mAP50: {metrics['metrics/mAP50']:.4f}")

print("\n=== Active Learning Completed ===")

```