Yolo và các phiên bản

So sánh và lựa chọn model Yolo cho dự án phân loại ảnh bạo lực, người lớn và bình thường và triển khai với Nestjs, FastAPI

Nhóm 15

Nguyễn Tuấn Thành – 22022624

Các phần thuyết trình:

1. Ưu nhược điểm của từng mô hình yolo.
2. Kiến trúc Yolov8n.
3. Training Yolov8n.
4. Triển khai trong ứng dụng mạng xã hội.

1. Ưu nhược điểm của từng model Yolo

YOLOv1 (2016)

- **Ưu điểm:**
  - Rất nhanh (real-time).
  - Đưa ra khái niệm one-stage detector: phát hiện toàn bộ ảnh trong một lần truyền qua mạng.
- **Nhược điểm:**
  - Độ chính xác thấp, đặc biệt với vật nhỏ.
  - Không tốt khi có nhiều vật thể gần nhau.
  - Localization kém.

YOLOv2 (YOLO9000 - 2017)

- **Ưu điểm:**
  - Sử dụng anchor boxes → cải thiện phát hiện vật nhỏ.
  - Dùng batch normalization → nhanh và chính xác hơn.
  - Kết hợp phân loại + detection (YOLO9000).
- **Nhược điểm:**
  - Mặc dù cải tiến, nhưng vẫn chưa chính xác bằng các two-stage detectors như Faster R-CNN.

1. Ưu nhược điểm của từng model Yolo

YOLOv3 (2018)

- **Ưu điểm:**
  - Multi-scale prediction → cải thiện phát hiện vật nhỏ.
  - Backbone mạnh hơn (Darknet-53).
  - Rất phổ biến, dùng nhiều trong thực tế.
- **Nhược điểm:**
  - Vẫn không có tính năng như segmentation hoặc instance mask.
  - Model khá nặng.

YOLOv4 (2020)

- **Ưu điểm:**
  - Sử dụng nhiều kỹ thuật mới: CSPDarknet, Mish, Mosaic data augmentation, DropBlock, CIoU, etc.
  - Tối ưu rất tốt giữa tốc độ và độ chính xác.
  - Dễ training trên GPU phổ thông.
- **Nhược điểm:**
  - Codebase không đồng nhất (Darknet framework, không phải PyTorch).
  - Không hỗ trợ tốt custom training như Ultralytics.

1. Ưu nhược điểm của từng model Yolo

🔹 YOLOv5 (2020, của Ultralytics – không phải của tác giả gốc)

- **Ưu điểm:**
  - Viết bằng PyTorch → dễ dùng, dễ custom.
  - Có nhiều kích thước model (n, s, m, l, x).
  - Hỗ trợ AutoML, quantization, export ONNX/TFLite.
- **Nhược điểm:**
  - Không chính thức từ tác giả YOLO (Joseph Redmon).
  - Một số người cho rằng "không phải YOLO thật".

🔹 YOLOv6 (2022, Meituan)

- **Ưu điểm:**
  - Tối ưu cho edge devices (như điện thoại, camera).
  - Dễ deployment trên TensorRT.
- **Nhược điểm:**
  - Ít cộng đồng hỗ trợ.
  - Không nổi bật so với YOLOv5/8 về user experience.

1. Ưu nhược điểm của từng model Yolo

YOLOv7 (2022, Chien-Yao Wang)

- **Ưu điểm:**
  - Rất nhanh và chính xác, đứng top trong nhiều benchmark.
  - Hỗ trợ cả task object detection và instance segmentation.
- **Nhược điểm:**
  - Không dễ sử dụng như YOLOv5/8 nếu bạn là người mới.
  - Codebase ít được cập nhật sau này.

YOLOv8 (2023, Ultralytics)

- **Ưu điểm:**
  - Sạch, hiện đại, hỗ trợ cả detection, segmentation, pose estimation.
  - Training, validation, deployment cực dễ.
  - Được Ultralytics hỗ trợ tốt.
- **Nhược điểm:**
  - Code mặc định nặng hơn YOLOv5 một chút.
  - Cần GPU tương đối mạnh nếu training bản lớn (YOLOv8x).

1. Ưu nhược điểm của từng model Yolo

YOLOv9 (2024, Ultralytics)

- **Ưu điểm:**
  - Giới thiệu **RT-DETR-like architecture** → bỏ head truyền thống.
  - Chính xác hơn YOLOv8 trong nhiều tác vụ.
  - Tối ưu inference tốt hơn.
- **Nhược điểm:**
  - Mới nên chưa có nhiều tài liệu cộng đồng.
  - Chưa tương thích 100% các flow cũ YOLOv5/8.

YOLOv10 (2024, Ultralytics)

- **Ưu điểm:**
  - Tập trung vào tốc độ inference cực nhanh (trên edge devices).
  - Sử dụng kỹ thuật mới như **re-parameterization** và **NAS (neural architecture search)**.
- **Nhược điểm:**
  - Trade-off nhỏ về độ chính xác so với v9 nếu chọn bản cực nhẹ.

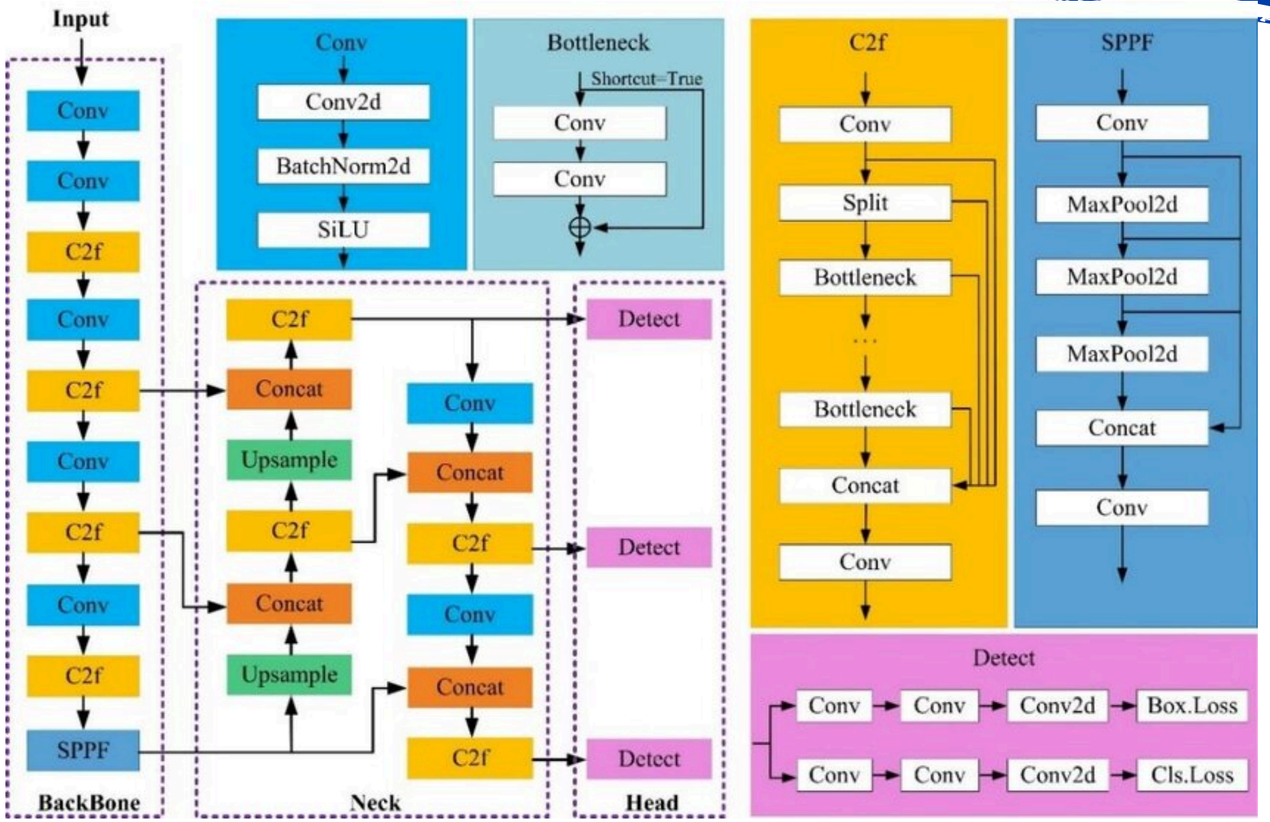
YOLOv11 (2025, Ultralytics - mới ra mắt)

- **Ưu điểm:**
  - Tiếp tục cải thiện tốc độ và độ chính xác.
  - Hỗ trợ **multi-modal input** (hình + text).
  - Tích hợp sâu với các nền tảng như Roboflow, Hugging Face.
- **Nhược điểm:**
  - Rất mới, cần thêm thời gian để kiểm chứng độ ổn định và benchmark.
  - Một số tính năng nâng cao chỉ hoạt động tốt trong hệ sinh thái Ultralytics.

2. Kiến trúc Yolov8:

a, Backbone:

Sử dụng kiến trúc CSPNet(Cross Stage Partial Network) dùng để tối ưu hóa trích xuất đặc trưng với hiệu xuất cao. CSPNet chia mạng nơ-ron thành hai thành phần chính, 1 dữ nguyên dòng thông tin và một thực hiện phép biến đổi sâu hơn để giảm thiểu độ trùng lặp thông tin giúp giảm thiểu lượng tính toán và duy trì độ chính xác cao



Kiến trúc của CSPNet có thể được biểu diễn như sau:

$$F_{\text{out}} = \text{CSP}(F_{\text{in}}) = F_{\text{in}}^1 + G(F_{\text{in}}^2)$$

Trong đó:

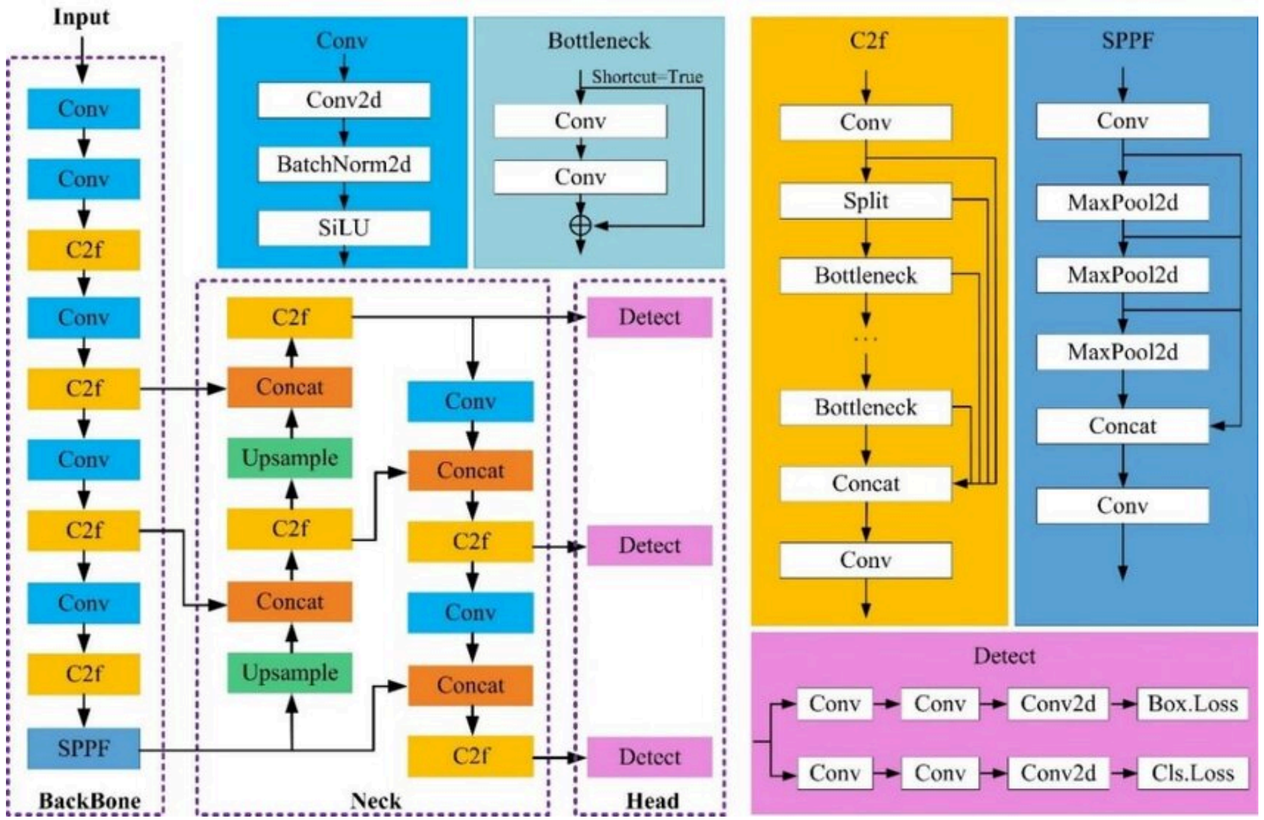
- F_{in} là đầu vào của một khối CSPNet.
- F_{in}^1 là dòng thông tin không thay đổi.
- $G(F_{\text{in}}^2)$ là phần áp dụng các phép biến đổi (như convolution) lên dòng thông tin thứ hai.
- F_{out} là đầu ra sau khi kết hợp hai dòng.

2. Kiến trúc Yolov8:

b. Neck:

Là thành phần kết hợp các đặc trưng từ các mức phân giải khác nhau để tối ưu việc phát hiện vật thể ở các kích thước khác nhau từ 2 công nghệ chính:

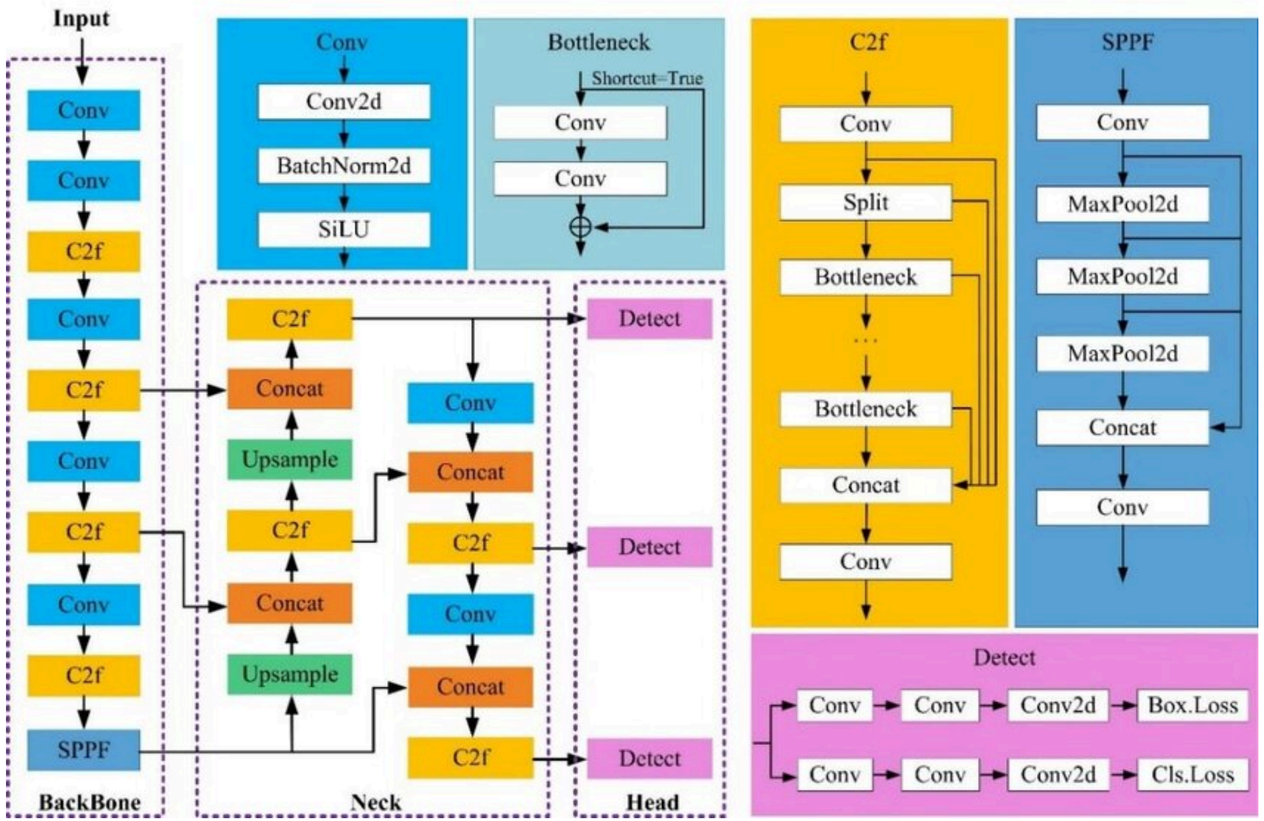
- FPN (Feature Pyramid Network): Giúp tổng hợp thông tin từ các tầng có độ phân giải khác nhau trong backbone, giúp mô hình có thể nhận diện vật thể lớn nhỏ đa dạng.
- PANet (Path Aggregation Network): PANet tối ưu hoá việc tổng hợp đặc trưng từ nhiều tầng khác nhau, tăng cường khả năng phát hiện vật thể nhỏ và mờ nhạt trong ảnh.



FPN và PANet giúp Yolov8 xử lý được nhiều kích thước và kiểu vật thể khác nhau, từ đó nâng cao độ chính xác trong các bài toán nhận diện phức tạp.

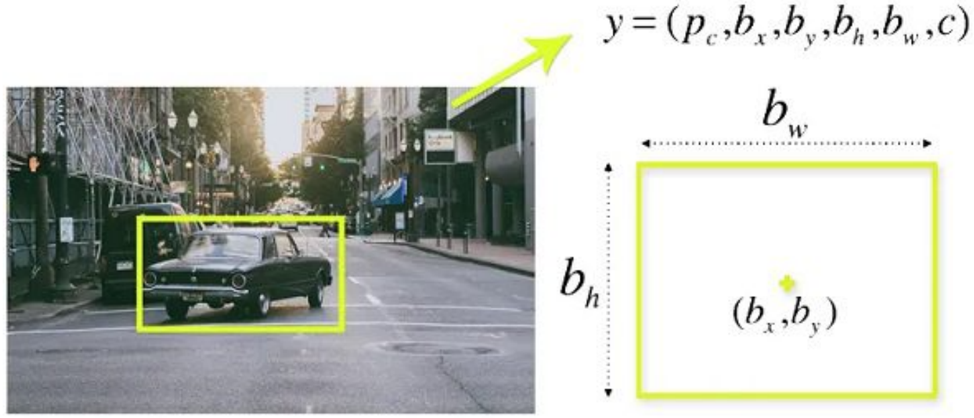
2. Kiến trúc Yolov8:

c. Head: Anchor-free Prediction:
Áp dụng cơ chế Anchor-free để dự đoán bounding box.
Cơ chế anchor-free giúp giảm độ phức tạp, giảm lượng tính toán và cải thiện độ chính xác trong phát hiện vật thể đặc biệt là vật thể có hình dạng không đồng nhất hoặc ở các vị trí bất thường.



2. Kiến trúc Yolov8:

Bounding box:



YOLO v8 sử dụng một hàm hồi quy để dự đoán các bounding box từ các đặc trưng đã tổng hợp. Giả sử x_c, y_c là toạ độ trung tâm của bounding box, và w, h là chiều rộng và chiều cao, ta có:

$$
\begin{aligned}
x_c &= \sigma(t_x) + c_x \\
y_c &= \sigma(t_y) + c_y \\
w &= p_w e^{t_w} \\
h &= p_h e^{t_h}
\end{aligned}
$$

Trong đó:

- σ là hàm sigmoid để đảm bảo giá trị trong khoảng [0, 1].
- t_x, t_y, t_w, t_h là các giá trị mà mạng dự đoán.
- c_x, c_y là các toạ độ của ô lưới hiện tại.
- p_w, p_h là các giá trị tham chiếu về kích thước.

2. Kiến trúc Yolov8:

Loss function:

Loss function của YOLO v8 được tối ưu để cân bằng giữa ba thành phần: localization loss, **Objectness** loss và classification loss. Công thức tổng quát có dạng:

$$\mathcal{L} = \lambda_{box} \cdot \mathcal{L}_{box} + \lambda_{obj} \cdot \mathcal{L}_{obj} + \lambda_{cls} \cdot \mathcal{L}_{cls}$$

Trong đó:

- $\mathcal{L}_{box}$: Box Loss – đo độ chính xác của vị trí và kích thước bounding box. Sử dụng **CIoU Loss** hoặc **GIoU Loss** để đánh giá độ chính xác của vị trí và kích thước hộp bao quanh đối tượng. Những loss này giúp mô hình học cách khớp chính xác bounding box với đối tượng cần phát hiện.
- $\mathcal{L}_{obj}$: Objectness Loss – để đánh giá xác suất có đối tượng trong mỗi ô lưới. Thành phần này được thiết kế để phân biệt giữa các ô lưới chứa đối tượng và các ô không chứa đối tượng, giúp mô hình dự đoán tốt hơn các đối tượng trong ảnh.
- $\mathcal{L}_{cls}$: Classification Loss – đo độ chính xác của việc phân loại đối tượng. Thường sử dụng **Binary Cross-Entropy (BCE) Loss** hoặc các hàm mất mát khác như **Focal Loss** để đánh giá khả năng mô hình phân loại đúng các đối tượng trong ảnh.
- $\lambda_{box}, \lambda_{obj}, \lambda_{cls}$: Các hệ số điều chỉnh trọng số cho từng thành phần loss.

3. Training Yolov8n:

Data:

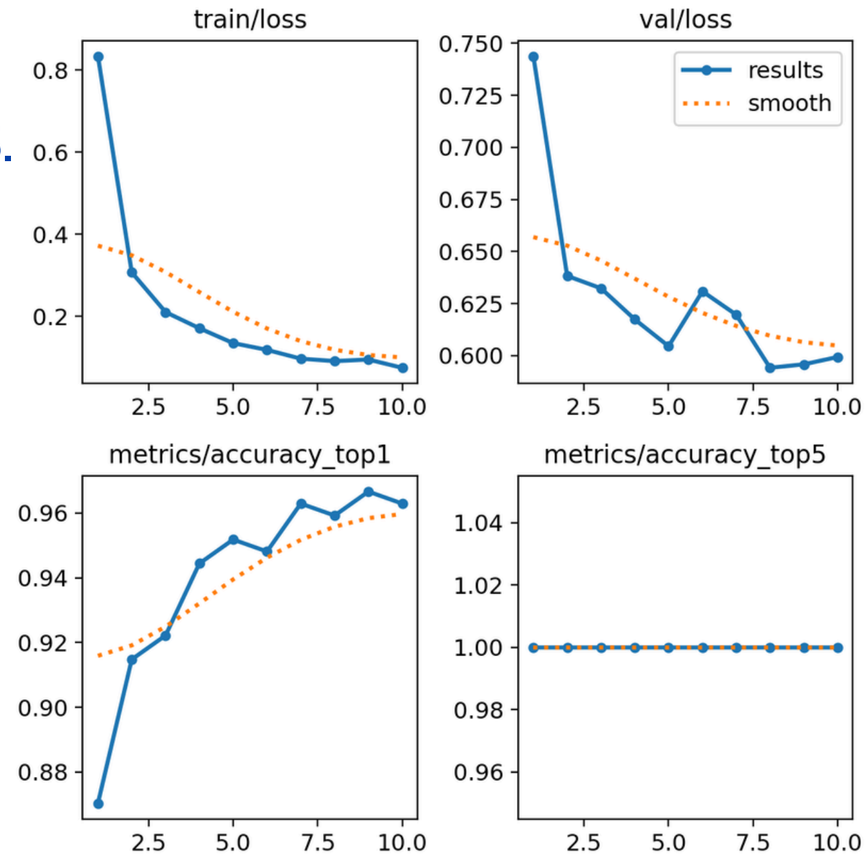
- 3 nhãn là Adult(Ảnh hở hang, gợi cảm) , Normal(Ảnh bình thường), Violent(Ảnh bạo lực) tỉ lệ nhãn 1:1:1.
- Nguồn dữ liệu:
  - Nhãn Adult thu thập trên web bằng selenium, pyautogui.
  - Nhãn Normal và Violent lấy từ tập dữ liệu trên kaggle
- Phân chia tập dữ liệu thành 3 phần train, val, test với tỉ lệ 70:15:15.

```
! cat {HOME}/data.yaml
```

```
train: train/
val: val/
test: test/
nc: 3
names: ["Normal", "Violent", "Adult"]
```

```
%cd {HOME}

!yolo task=classify mode=train model=yolov8n-cls.pt data={HOME} epochs=10 imgsz=256 batch=32
```



3. Training Yolov8n:

Kết quả với tập val qua từng epoch:

	epoch	train/loss	metrics/accuracy_top1	metrics/accuracy_top5	val/loss	lr/pg0	lr/pg1	lr/pg2
2	1	0.83442	0.87037	1	0.74385	0.00023205	0.00023205	0.00023205
3	2	0.30695	0.91481	1	0.63828	0.00042352	0.00042352	0.00042352
4	3	0.21011	0.92222	1	0.63232	0.00056786	0.00056786	0.00056786
5	4	0.1707	0.94444	1	0.61738	0.00050194	0.00050194	0.00050194
6	5	0.13445	0.95185	1	0.60449	0.00043126	0.00043126	0.00043126
7	6	0.11798	0.94815	1	0.63086	0.00036057	0.00036057	0.00036057
8	7	0.09649	0.96296	1	0.61963	0.00028988	0.00028988	0.00028988
9	8	0.0906	0.95926	1	0.59414	0.0002192	0.0002192	0.0002192
10	9	0.09442	0.96667	1	0.5958	0.00014851	0.00014851	0.00014851
11	10	0.07431	0.96296	1	0.59932	7.7826e-05	7.7826e-05	7.7826e-05

Kết quả trên tập test(95.9%):

```
[ ] %cd {HOME}

    !yolo task=classify mode=val model=/content/drive/MyDrive/ImageGuard/training_tuanthanh/best.pt data=/content/drive/MyDrive/ImageGuard/dataset_classification split=test
```

```
/content/drive/MyDrive/ImageGuard/dataset_classification
Ultralytics YOLOv8.2.103 🚀 Python-3.11.11 torch-2.6.0+cu124 CPU (Intel Xeon 2.20GHz)
YOLOv8n-cls summary (fused): 73 layers, 1,438,723 parameters, 0 gradients, 3.3 GFLOPs
train: /content/drive/MyDrive/ImageGuard/dataset_classification/train... found 1260 images in 3 classes ✅
val: /content/drive/MyDrive/ImageGuard/dataset_classification/val... found 270 images in 3 classes ✅
test: /content/drive/MyDrive/ImageGuard/dataset_classification/test... found 270 images in 3 classes ✅
test: Scanning /content/drive/MyDrive/ImageGuard/dataset_classification/test... 270 images, 0 corrupt: 100% 270/270 [00:58<00:00,  4.62it/s]
test: New cache created: /content/drive/MyDrive/ImageGuard/dataset_classification/test.cache
               classes   top1_acc   top5_acc: 100% 17/17 [00:13<00:00,  1.30it/s]
                   all      0.959          1
Speed: 0.0ms preprocess, 17.4ms inference, 0.0ms loss, 0.0ms postprocess per image
Results saved to runs/classify/val2
💡 Learn more at https://docs.ultralytics.com/modes/val
```

4. Triển khai:

Triển khai trên backend dùng Nestjs và FastAPI với luồng của người dùng như sau:
→ Người dùng đăng bài
→ Validate data đầu vào
→ Thêm id bài đăng vào BullMQ(trong backend Nestjs) và trả về đăng bài thành công
→ Processor lấy id trong BullMQ và xử lý xem ảnh, video đó có phù hợp với chính sách ảnh 16+ hay không(dùng FastAPI)
→ Nếu không thì gửi thông báo cho người dùng và gỡ bài

Source code:
Model phân loại ảnh: https://github.com/ntthanh2603/ImageGuard.git
Backend mạng xã hội: https://github.com/ntthanh2603/SNet-Backend.git