HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

BỘ MÔN THỰC TẬP CƠ SỞ

Báo Cáo Thực Tập Cơ sở

Đề tài:

Xây dựng mô hình học sâu áp dụng cách học chắt lọc kiến thức để nhận dạng người bị ngã

Giảng viên hướng dẫn	**: TS. Nguyễn Tất Thắng**
Họ và tên sinh viên	**: Nguyễn Thành Chung**
Mã sinh viên	**: B22DCKH014**
Lớp	**: D22CQKH02-B**
Nhóm	**: 47**

Hà Nội – 2025

I. Mô tả bài toán

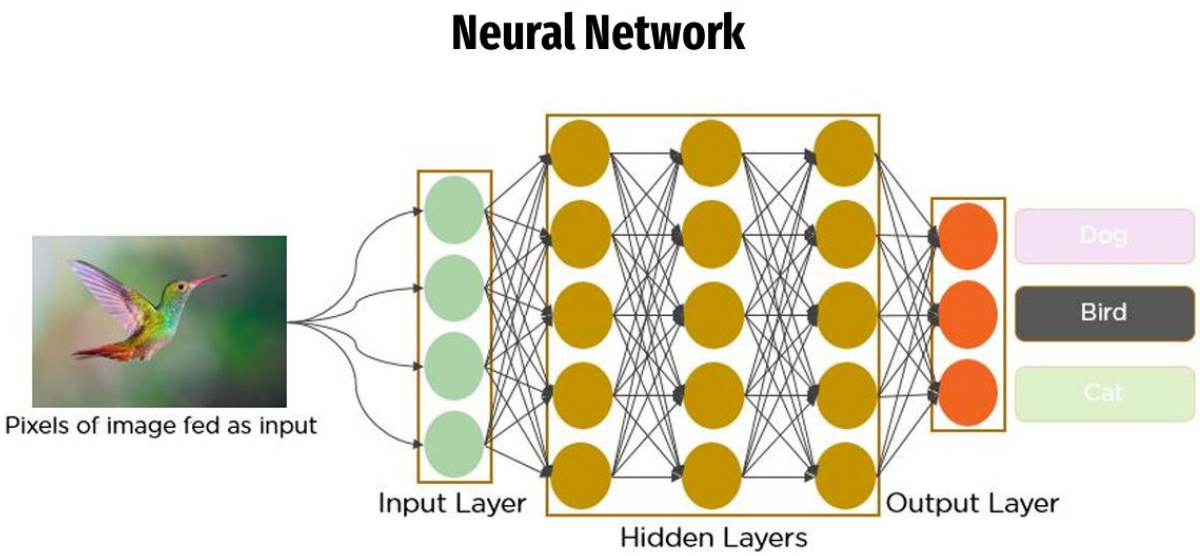
- Bài toán phát hiện ngã (Fall Detection) được mở rộng triển khai trong môi trường ngoài trời như công viên, vỉa hè, khu dân cư, hoặc khu vực công cộng. Mục tiêu của hệ thống là tự động phát hiện các trường hợp người bị ngã thông qua camera giám sát và gửi cảnh báo kịp thời đến trạm y tế hoặc công an phường gần nhất để can thiệp. Đây là giải pháp ứng dụng AI phục vụ cộng đồng, nhằm nâng cao khả năng giám sát an toàn tại nơi công cộng, đặc biệt hữu ích với người già, người khuyết tật, hoặc người đi bộ một mình vào ban đêm.

- Hệ thống được thiết kế hoạt động tự động và độc lập tại chỗ nhờ sử dụng thiết bị biên sử dụng GPU là NVIDIA Jetson TX2, giúp xử lý video thời gian thực ngay tại camera mà không cần gửi dữ liệu lên máy chủ trung tâm.

- NVIDIA Jetson TX2 yêu cầu mô hình có tối đa số lượng tham số (Parameters): ≤ 13 triệu tham số, FLOPs (Floating Point Operations): ≤ 20–30 GFLOPs

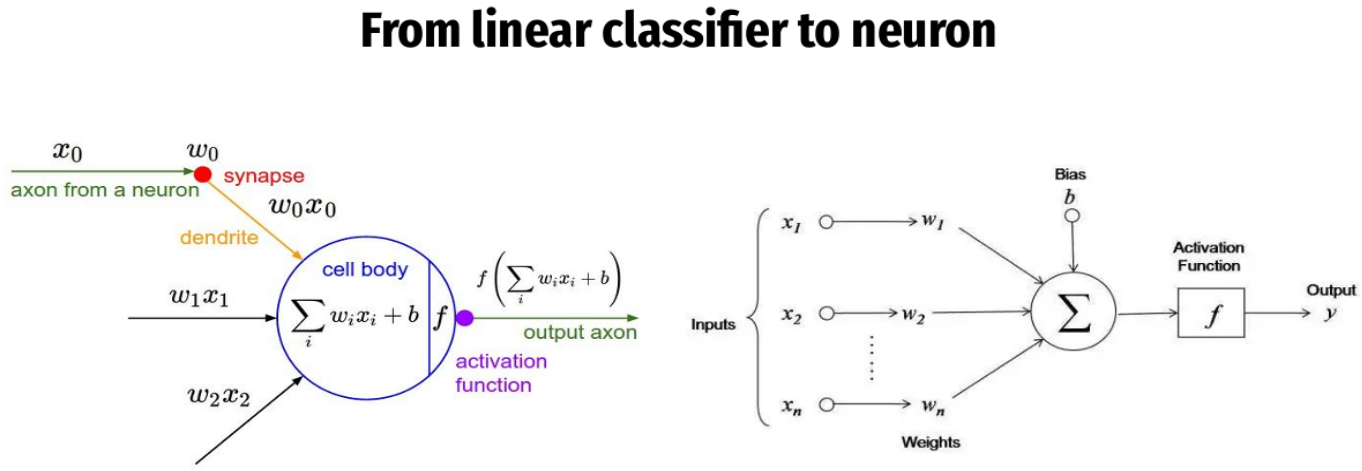
II. Lý thuyết được sử dụng

0. Tổng quan về Neural Network



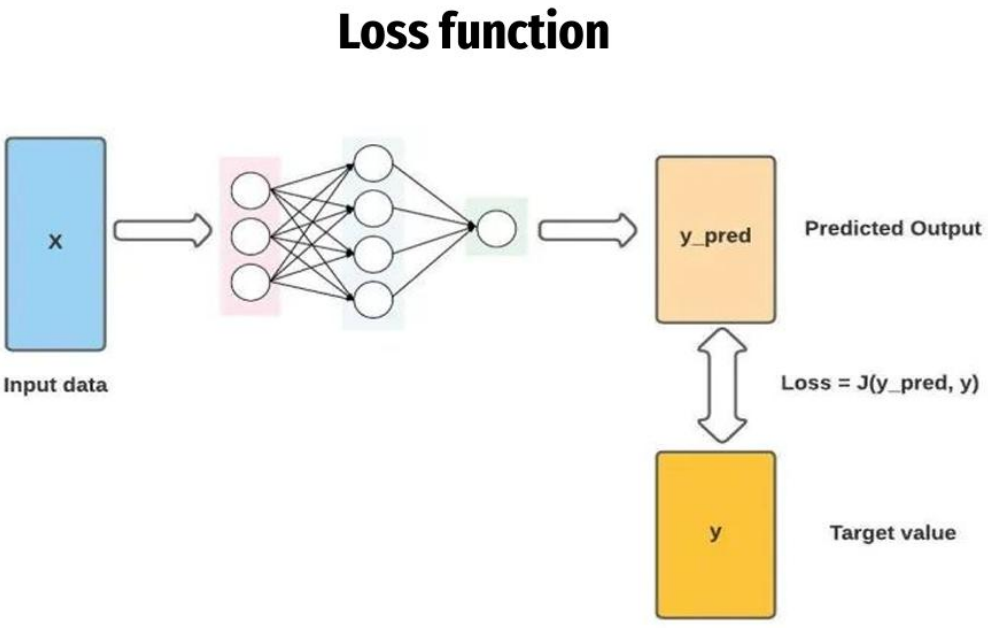
Ảnh 0: Neural Network

- Neural network (mạng nơ-ron nhân tạo) là một mô hình tính toán lấy cảm hứng từ cách hoạt động của bộ não con người. Nó bao gồm nhiều lớp (layers) các đơn vị xử lý nhỏ gọi là *neurons* (nơ-ron), được kết nối với nhau thông qua các trọng số (weights).
- Một mạng nơ-ron cơ bản thường bao gồm:
    - Lớp đầu vào (Input layer): Nhận dữ liệu đầu vào (ví dụ: ảnh, văn bản, số liệu).
    - Lớp ẩn (Hidden layers): Thực hiện các phép biến đổi phức tạp lên dữ liệu nhờ vào các trọng số và hàm kích hoạt (*activation functions*). Một mạng có thể có một hoặc nhiều lớp ẩn.
    - Lớp đầu ra (Output layer): Trả về kết quả dự đoán (ví dụ: nhãn phân loại, giá trị số...).
- Cách hoạt động:
    - Lan truyền xuôi (Forward Propagation): Dữ liệu đầu vào đi qua các lớp, mỗi lớp tính toán đầu ra dựa trên đầu vào và các trọng số, rồi áp dụng hàm kích hoạt.



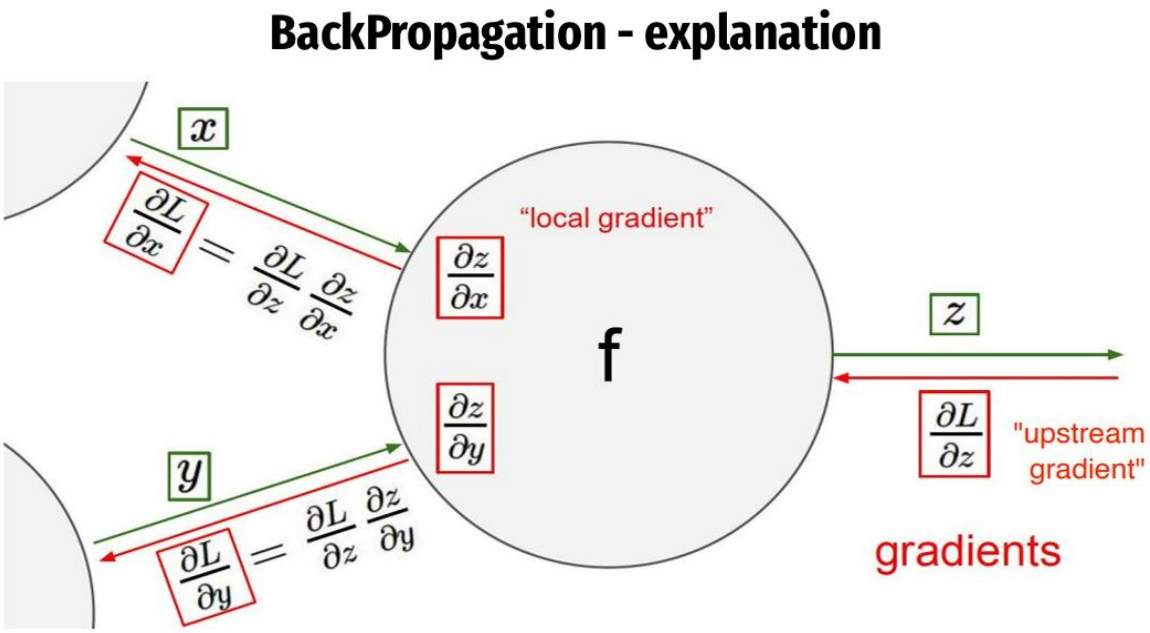
Ảnh 1: Quá trình Forward Propagation

- Tính toán lỗi: So sánh đầu ra của mạng với nhãn thật để tính *loss* (hàm mất mát).



Ảnh 2: Quá trình tính hàm mất mát Loss

- Lan truyền ngược (Backpropagation): Tính gradient của loss theo từng trọng số và cập nhật chúng bằng các thuật toán tối ưu (ví dụ: *Stochastic Gradient Descent*, *Adam*).



Ảnh 3: Quá trình BackPropagation

- Ưu điểm của Neural Network:
  - Khả năng học phi tuyến (Non-linear learning)
  - Tự động trích xuất đặc trưng (feature learning)
  - Linh hoạt và mở rộng
- Nhược điểm của Neural Network:
  - Yêu cầu dữ liệu lớn
  - Khó giải thích (Black box)
  - Thời gian huấn luyện dài (chứa nhiều parameters so với các kiến trúc khác)
  - Phụ thuộc nhiều vào hyperparameter

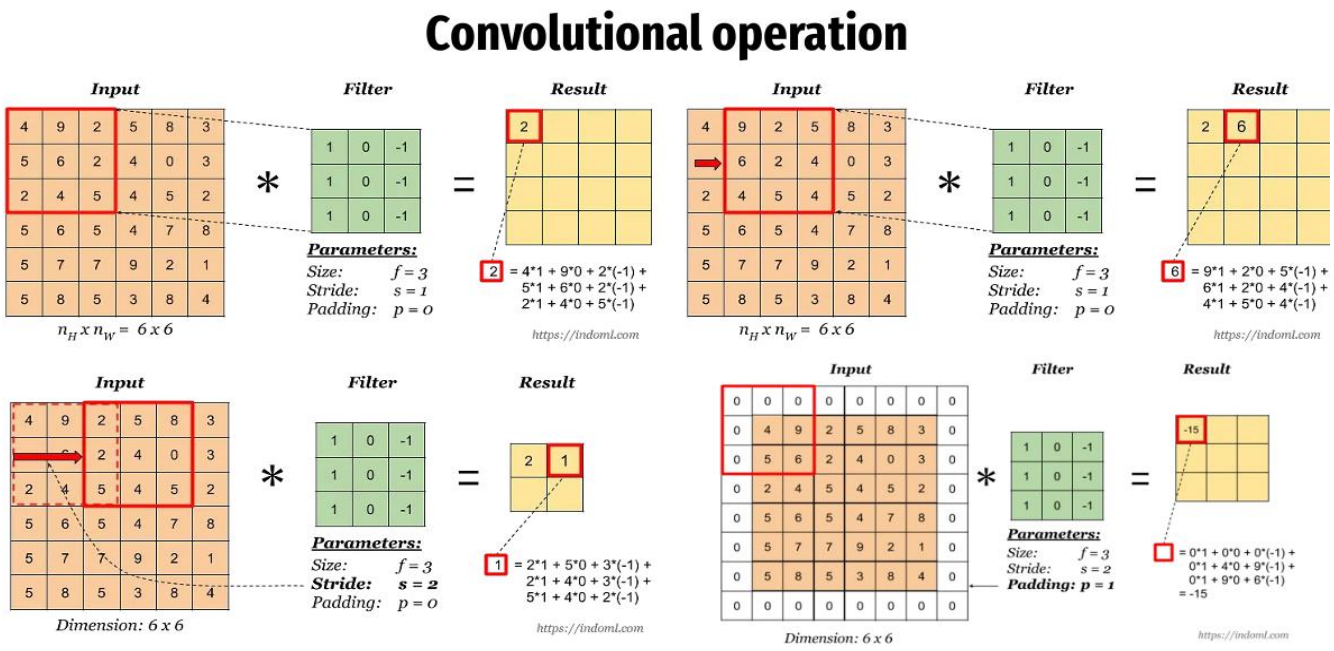
1. Lý thuyết Convolutional Neural Network

1.1 Tổng quan

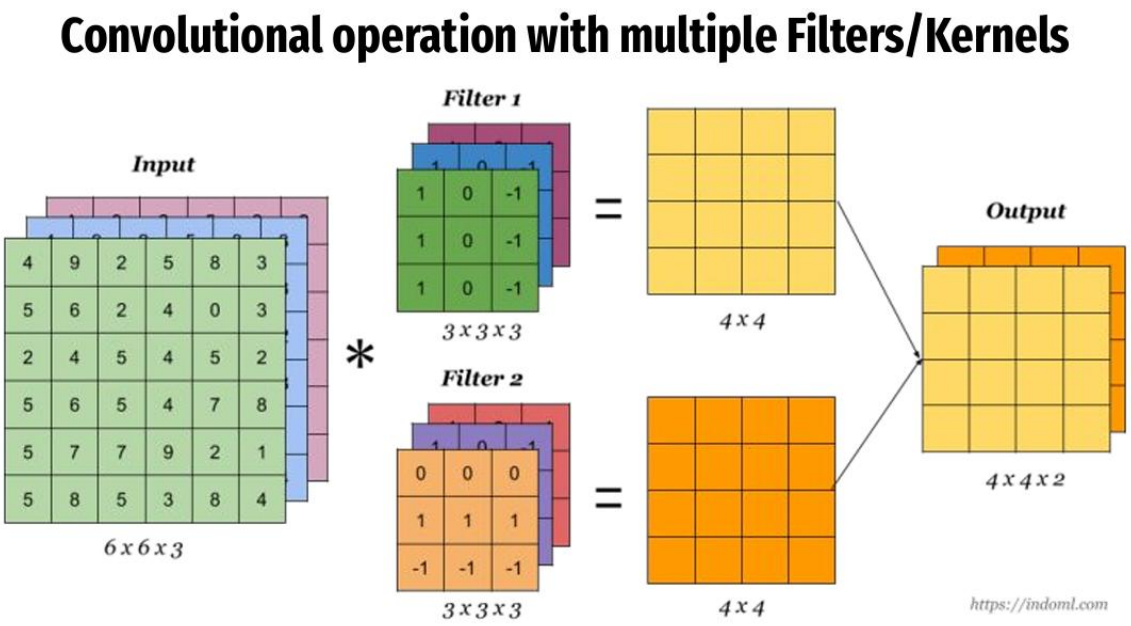
- Convolutional Neural Network - CNN (Mạng nơ-ron tích chập) là một loại mạng nơ-ron chuyên biệt, rất hiệu quả trong việc xử lý dữ liệu có cấu

trúc dạng lưới như hình ảnh và dữ liệu không gian. Nó được thiết kế để tự động học ra các đặc trưng (features) từ dữ liệu hình ảnh thông qua các phép tích chập (*convolution*), giúp tăng độ chính xác và giảm số lượng tham số cần học.

- Kiến trúc của CNN thông thường bao gồm:
  - Convolutional Layer (Lớp tích chập):
    + Dùng các bộ lọc (filter/kernel) trượt qua ảnh đầu vào để trích xuất đặc trưng như cạnh, đường, góc, họa tiết, ...
    + Mỗi bộ lọc học ra một đặc trưng riêng.

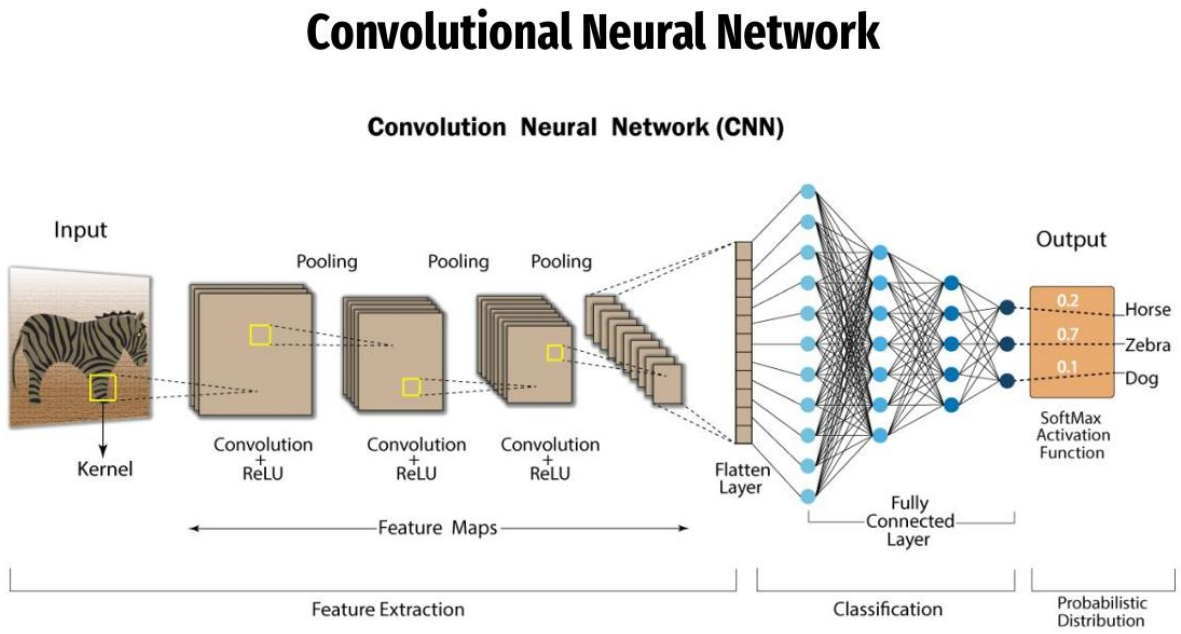


Ảnh 4: Kernel/ Filter của CNN



Ảnh 5: Phép tích chập với nhiều Kernels/ Filters

- Activation Layer (Hàm kích hoạt, thường là ReLU): Áp dụng sau lớp tích chập để đưa tính phi tuyến vào mô hình.
- Pooling Layer (Lớp gộp, ví dụ: Max Pooling): Giảm kích thước đặc trưng (downsampling), tăng khả năng khái quát, giảm số lượng tham số.
- Fully Connected Layer (Lớp kết nối đầy đủ): Ở cuối mạng, dùng để đưa ra dự đoán như phân loại ảnh
- Output Layer: Trả về kết quả cuối cùng (ví dụ: nhãn ảnh thuộc mèo, chó, v.v.).



Ảnh 5: Kiến trúc CNN

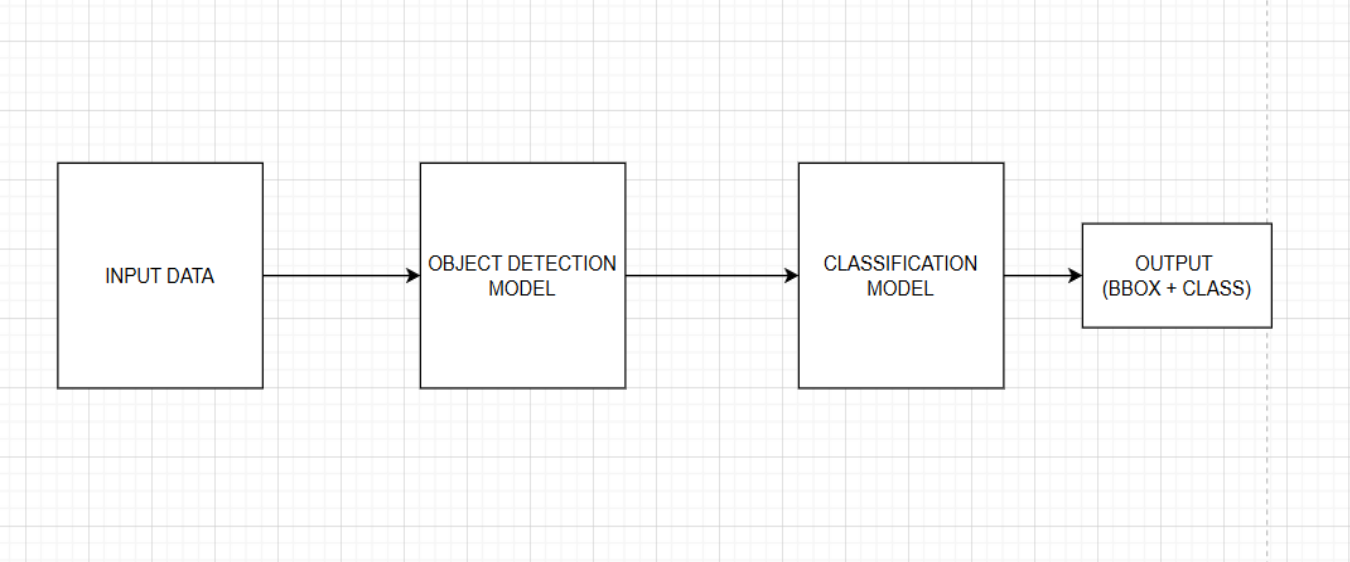
1.2 Những ưu và nhược điểm

- Ưu điểm:
    - Parameter Sharing (chia sẻ tham số): Nhờ cơ chế chia sẻ trọng số trong các lớp tích chập, CNN sử dụng ít tham số hơn nhiều so với mạng fully connected → tiết kiệm bộ nhớ và tăng hiệu suất.
    - Local Connectivity (kết nối cục bộ): Mỗi nơ-ron trong lớp tích chập chỉ liên kết với một vùng nhỏ trong ảnh → **khai thác thông tin cục bộ hiệu quả.**
    - Translation Invariance (bất biến dịch chuyển): là khả năng nhận biết đặc trưng dù nó bị dịch chuyển trong ảnh → Bảo toàn thông tin không gian (thời gian) của ảnh
- Nhược điểm:
    - Yêu cầu dữ liệu lớn
    - **Thiếu khả năng suy luận không gian toàn cục**
    - Không linh hoạt với biến đổi hình học
    - **Black box (khó giải thích)**

2. Mô hình CNN được áp dụng trong dự án

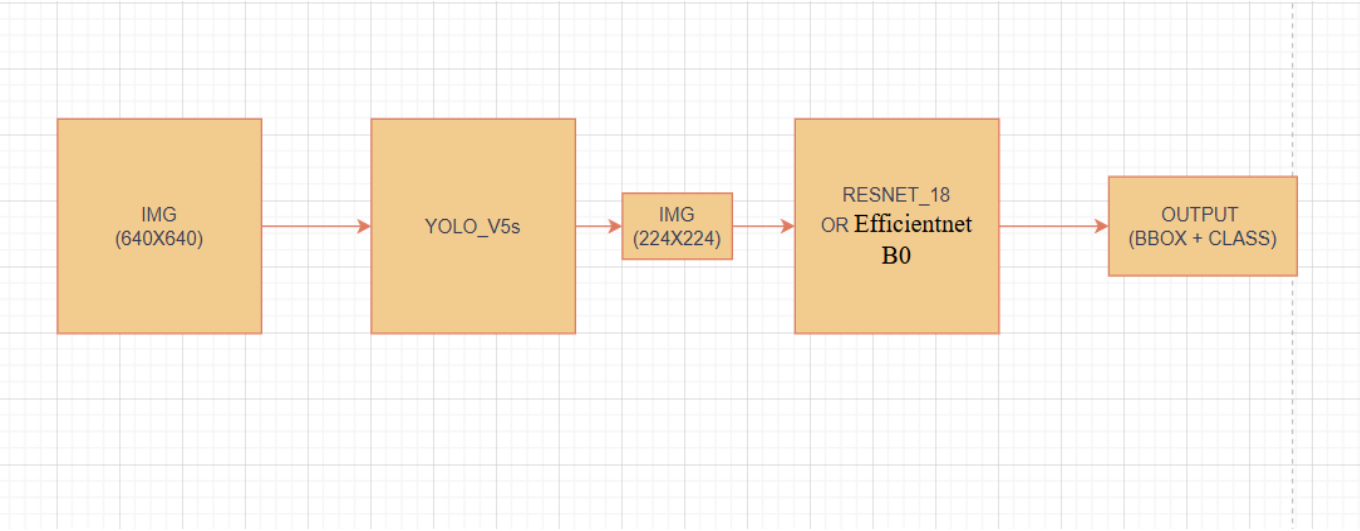
2.1 Mô tả khái quát:

- Ta nhận dạng đây là 1 bài toán object detection điển hình trong lĩnh vực Thị giác máy tính (Computer Vision). Kết hợp với yêu cầu bài toán là triển khai trên thiết bị biên có sử dụng gpu NVIDIA Jetson TX2 → ta cần phải lựa chọn mô hình học sâu hiệu quả cũng như phù hợp với yêu cầu bài toán
- Trong dự án của mình, em sử dụng 2 mô hình chính (1 mô hình object detection + 1 mô hình classification) kết hợp với nhau thành 1 pipeline.
- Tại sao phải chọn 2 mô hình?
    - Mô hình object detection rất mạnh với khả năng phát hiện vị trí của vật thể, tuy nhiên nhược điểm của những mô hình detection là khả năng dự đoán những classes (lớp) của vật thể đó (trong dự án classes sẽ gồm ‘fall’ và ‘normal’) không quá cao
    - Trong khi đó mô hình classification lại chuyên về phân loại và có độ chính xác cao hơn rất nhiều so với mô hình object detection
- Cách thức hoạt động: Ảnh đầu vào (640x640) sau khi được thiết bị biên thu thập sẽ được đưa vào mô hình detection đầu tiên. Mô hình detection sẽ phát hiện vị trí của người, sau đó vị trí đó sẽ được lưu lại thành 1 ảnh và thay đổi kích cỡ (224x224) rồi được đưa vào mô hình Phân loại (Classification). Mô hình phân loại tiến hành phân loại nhãn cho người là ngã (fall) hoặc bình thường (normal). Kết quả đầu ra sẽ bao gồm vị trí và trạng thái của người đó



Ảnh 6: Khái quát hoạt động của dự án

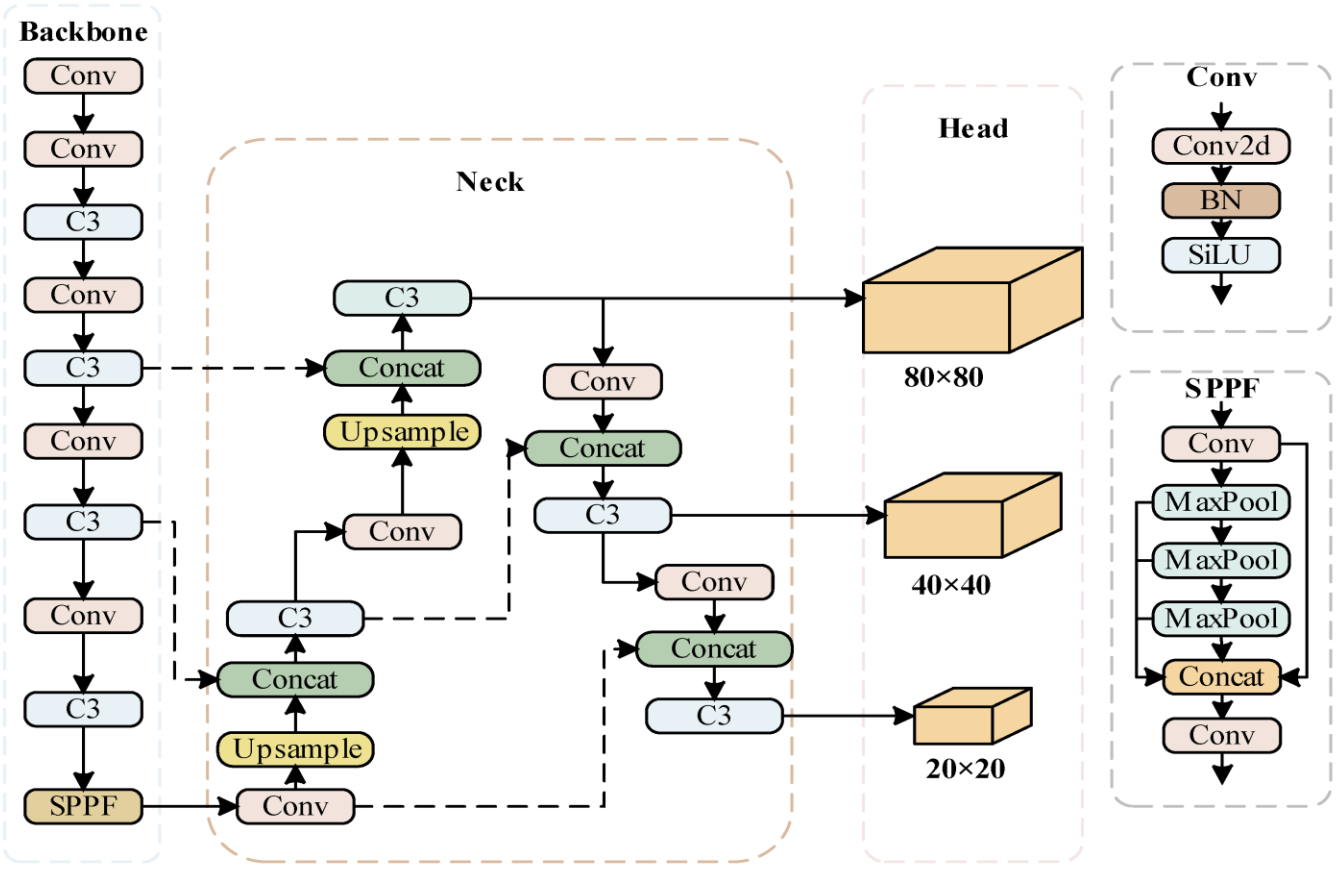
- Lựa chọn mô hình: Dựa trên yêu cầu của dự án là triển khai trên thiết bị sử dụng gpu NVIDIA Jetson TX2: em đề xuất sử dụng
    - Object detection: Yolov5s (7,2 triệu tham số)
    - Classification: Resnet 18 (11,3 triệu tham số) hoặc Efficientnet B0 (5 triệu tham số)



Ảnh 7: Chi tiết pipleline mô hình

2.2 Mô hình Yolov5

2.2.1 Kiến trúc Yolov5



Ảnh 8: Kiến trúc Yolov5

- Kiến trúc của YOLOv5 gồm **3 thành phần chính**:

  - Backbone – Trích xuất đặc trưng:

    + Sử dụng CSPDarknet53 (Cross Stage Partial Darknet), là phiên bản cải tiến từ Darknet trong YOLOv4.

    + CSPDarknet53 có thành phần chính là CSP Blocks: nó chia tensor đầu vào thành 2 phần (theo số lượng kênh chia đôi): 1 phần giữ nguyên phần còn 1 phần sẽ đi qua residual network. Cuối cùng nối 2 phần này lại.

    + Nhiệm vụ: trích xuất đặc trưng từ ảnh đầu vào.

  - Neck – Tăng cường đặc trưng:

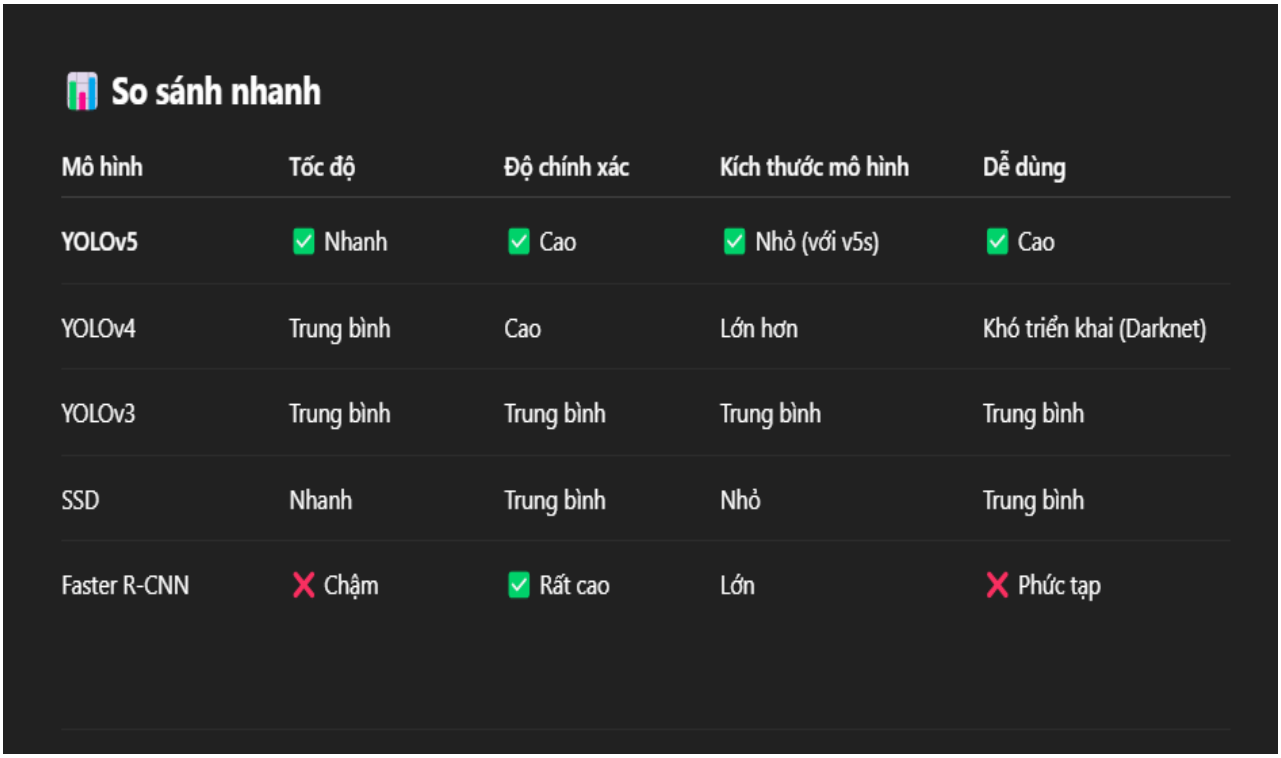
+ SPPF (Spatial Pyramid Pooling - Fast): thu nhận thông tin ở nhiều thang đo mà không tăng độ phức tạp.

+ PANet (Path Aggregation Network): giúp kết hợp đặc trưng từ nhiều tầng để phát hiện vật thể ở các kích thước khác nhau.

- Head – Dự đoán đầu ra: Dự đoán bounding boxes và Dự đoán nhãn lớp và độ tin cậy (confidence)

2.2.2 Tại sao lại chọn Yolov5?

- Ưu điểm của Yolov5 so với những mô hình object detection đi trước:
  - Nhẹ hơn, nhanh hơn và dễ triển khai
  - Hiệu quả vượt trội
  - Tự động hóa và tiện dụng
  - Training và inference linh hoạt
  - Hỗ trợ tốt cộng đồng và cập nhật thường xuyên



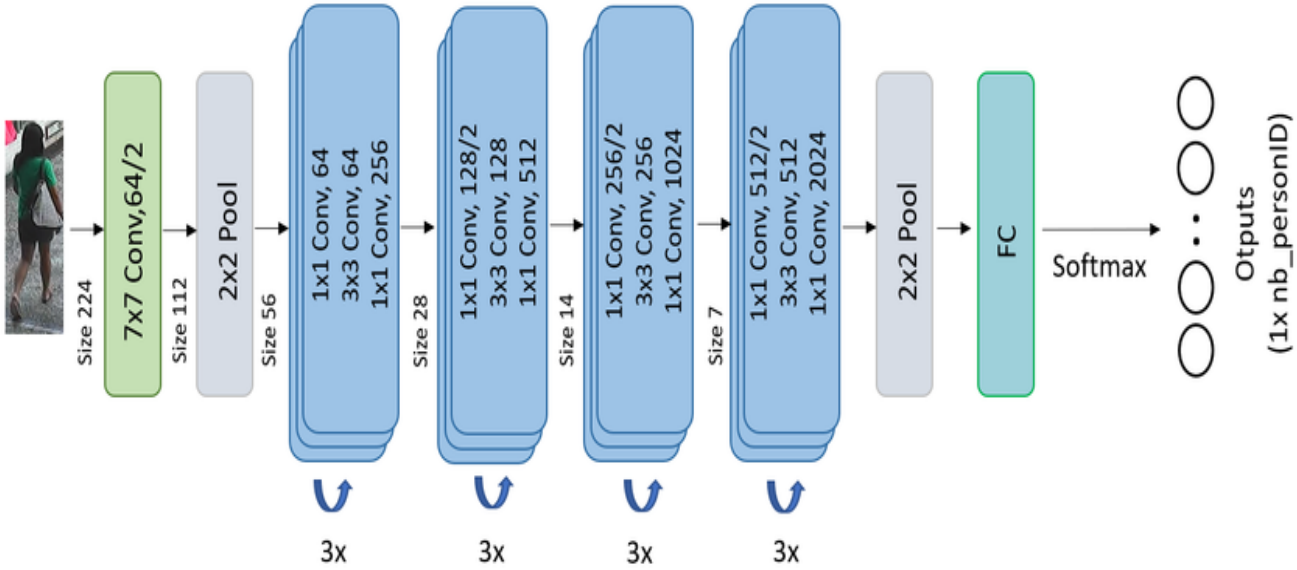
📊 So sánh nhanh

Mô hình	Tốc độ	Độ chính xác	Kích thước mô hình	Dễ dùng
YOLOv5	✅ Nhanh	✅ Cao	✅ Nhỏ (với v5s)	✅ Cao
YOLOv4	Trung bình	Cao	Lớn hơn	Khó triển khai (Darknet)
YOLOv3	Trung bình	Trung bình	Trung bình	Trung bình
SSD	Nhanh	Trung bình	Nhỏ	Trung bình
Faster R-CNN	❌ Chậm	✅ Rất cao	Lớn	❌ Phức tạp

Ảnh 9: So sánh Yolov5 với các mô hình tiêu biểu trước nó

2.3 Mô hình Resnet 18

2.3.1 Kiến trúc Resnet 18

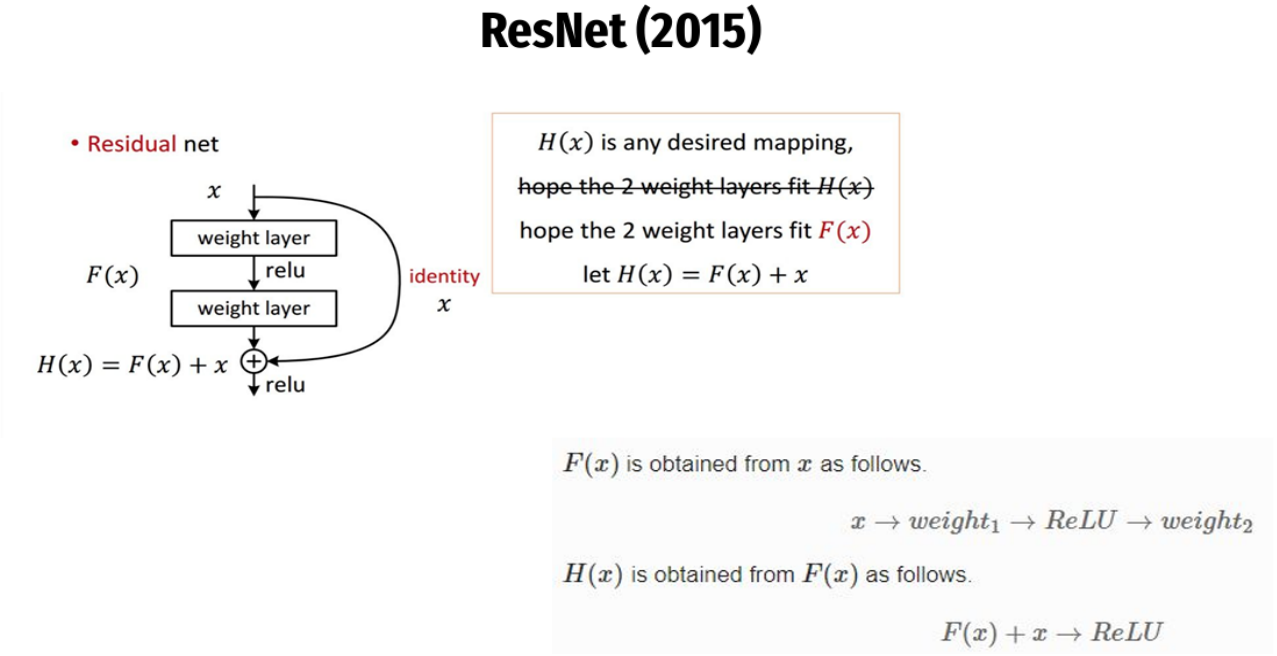


Ảnh 10: Kiến trúc mô hình Resnet 18

- **Giới thiệu**: ResNet là một kiến trúc mạng nơ-ron sâu nổi tiếng được giới thiệu bởi Microsoft Research trong bài báo *"Deep Residual Learning for Image Recognition"* (2015), và đã giành giải nhất trong cuộc thi ImageNet 2015.
- Trong dự án em sử dụng Resnet 18 với thành phần chính gồm có khối CNN, 4 khối Residual Block (bao gồm các khối CNN, hàm kích hoạt activation functions và lối tắt skip connection), khối booling và khối softmax

2.3.2 Mục tiêu của RestNet

- ResNet có thể:
    - Giải quyết vấn đề suy giảm hiệu suất khi mạng càng sâu (vanishing/exploding gradients).
    - Cho phép huấn luyện các mô hình rất sâu (lên tới hàng trăm lớp).



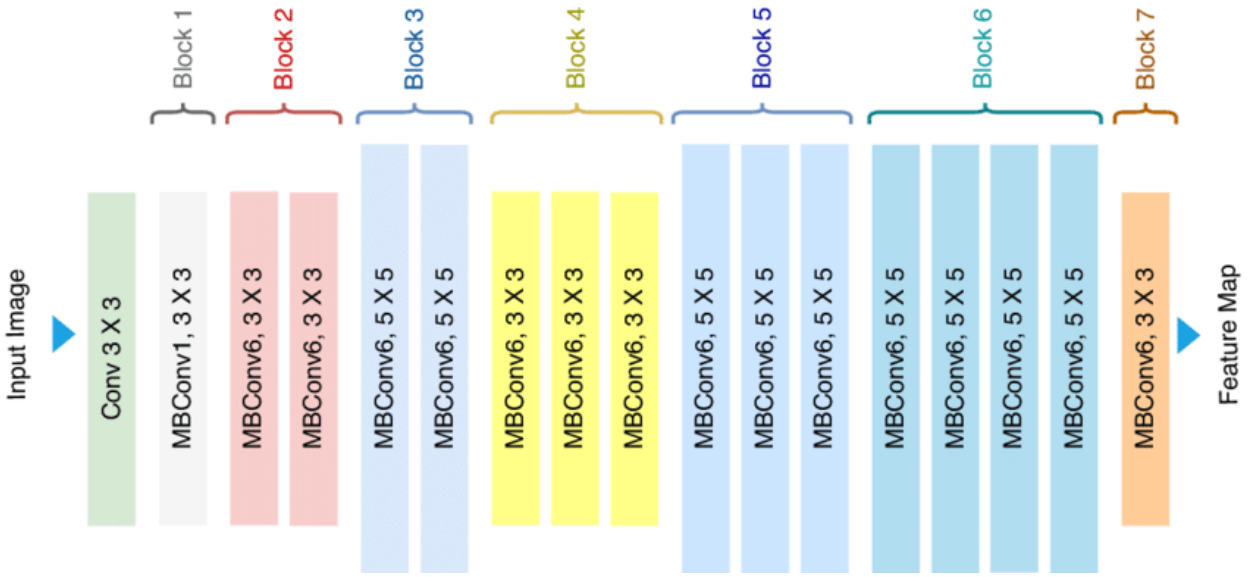
Ảnh 11: Ý tưởng của Residual Block

- Mục đích tạo ra *Residual Block* để giảm thiểu khả năng vanishing gradient trong quá trình BackPropagations khi mô hình được huấn luyện:
    - Vanishing Gradient (Gradient biến mất): xảy ra khi trong quá trình BackPropagations Gradient được áp dụng quy tắc đạo hàm chuỗi và nếu như 1 đạo hàm địa phương trở lên bé, đặt biệt sau khi đi qua các hàm kích hoạt như Sigmoid hoặc Tanh thì giá trị của đầu ra trở lên rất bé và khiến cho giá trị của Gradient càng tiến gần về 0. Do đó mô hình gặp khó khăn trong quá trình cập nhật tham số. Nếu giá trị Gradient về 0 thì tham số tại nơ-ron đó sẽ gần như không được cập nhật
    - Ý tưởng của Residual Block là: Residual Block là khối nền tảng của mạng ResNet. Thay vì học trực tiếp đầu ra H(x), nó học phần chênh lệch (residual) giữa đầu vào và đầu ra: output = F(x) + x
    - Vanishing Gradient rất dễ xảy ra với những model có chiều sâu lớn (nhiều layer). Từ trước năm 2015, vấn đề này rất nhức nhối và chúng ta cũng không có cách để giảm thiểu đáng kể tình trạng này, cho dù các hàm kích hoạt thân thiện với quá trình BackPropagations như Relu và các biến thể của nó được phát triển.

Do đó ta biết được *Residual Block hay Residual Network* đóng 1 vai trò quan trọng rất lớn trong quá trình phát triển của AI

2.4 Mô hình Efficientnet-B0

2.4.1 Kiến trúc của Efficientnet-B0



Ảnh 12: Kiến trúc Efficientnet-B0

- **Giới thiệu**: Kiến trúc của EfficientNet-B0 là một mạng nơ-ron tích chập (CNN) được thiết kế bởi Google với mục tiêu đạt được hiệu suất cao và tối ưu tài nguyên. EfficientNet-B0 là phiên bản cơ bản trong dòng EfficientNet, sử dụng chiến lược compound scaling để cân bằng giữa chiều sâu (depth), chiều rộng (width), và độ phân giải ảnh (resolution) khi mở rộng mạng.
- Có thể chia kiến trúc EfficientNet-B0 thành 3 phần chính:
  - Stem: 1 lớp Conv3x3
  - Body: 7 khối MBConv, mỗi khối sẽ có những nhân (kernel) và bước dịch (stride) cũng như số lượng khác nhau
  - Head: 1 lớp Conv1x1 (để giảm số lượng kênh mà vẫn giữ nguyên kích thước) + Pooling layer + Fully Connected Layer

- Kiến trúc của khối MBConv (Mobile Inverted Bottleneck Convolution) gồm:
    - Expansion (Conv1x1): Tăng số lượng kênh
    - Depthwise Convolution (k=3 hoặc 5)
    - Projection (Conv1x1): đầu ra của khối.
    - Skip/Residual Connection
    - Squeeze-and-Excitation (SE block) (trong EfficientNet): Một thành phần bổ sung để trọng số hóa kênh quan trọng, giúp tăng khả năng học đặc trưng.

2.4.2 Mục tiêu của Efficientnet

- Tối đa hóa hiệu suất mô hình (accuracy) trong khi giảm thiểu chi phí tính toán (thời gian, bộ nhớ, FLOPs, tham số).
- EfficientNet hướng đến:
    - Hiệu quả cao về tính toán (Efficiency): Đạt độ chính xác cao với ít tham số hơn nhiều so với các kiến trúc CNN truyền thống như ResNet, VGG, Inception.
    - Tối ưu hóa quá trình scale-up mạng: EfficientNet sử dụng chiến lược Compound Scaling để tăng đồng thời 3 yếu tố: độ sâu (số lượng lớp), độ rộng (số lượng kênh), độ phân giải ảnh (chiều dài, rộng của ảnh được gộp chung lại) theo một tỷ lệ tối ưu được tìm ra bằng thuật toán tìm kiếm kiến trúc (neural architecture search – NAS). Chiến lược Compound Scaling sẽ được áp dụng vào khối MBConv trong kiến trúc của EfficientNet.

In this paper, we propose a new **compound scaling method**, which use a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way:

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{3}$$

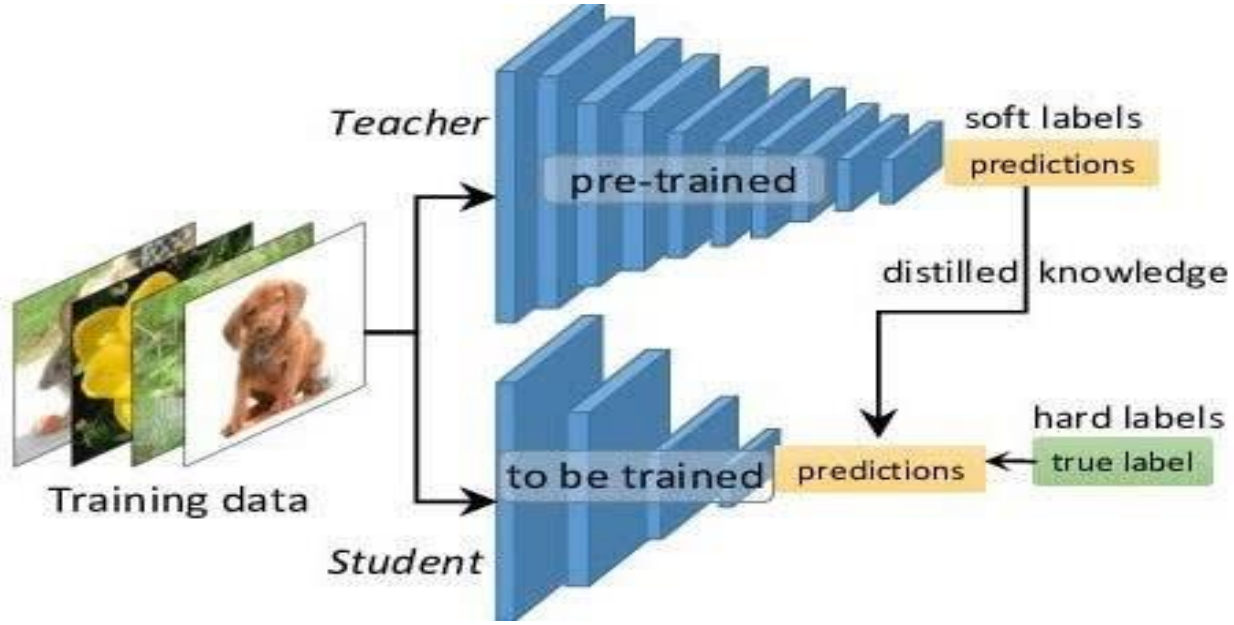
where α, β, γ are constants that can be determined by a small grid search. Intuitively, ϕ is a user-specified coefficient that controls how many more resources are available for model scaling, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution respectively. Notably, the FLOPS of a regular convolution op is proportional to d, w^2, r^2, i.e., doubling network depth

Ảnh 13: Công thức của Compound Scaling

- Khả năng mở rộng dễ dàng
- Tối ưu hóa cho cả inference và training: Thân thiện với deployment trên thiết bị thật nhờ mô hình nhỏ gọn, dễ dàng fine-tune trên các tập dữ liệu khác (transfer learning)

3. Lý thuyết Knowledge Distillation

3.1 Lý thuyết tổng quan



Ảnh 12: Ý tưởng của knowledge distillation

- Knowledge Distillation là một kỹ thuật trong học sâu (deep learning) dùng để huấn luyện một mô hình nhỏ (Student) học lại kiến thức từ một mô hình lớn mạnh hơn (Teacher).
- Mô hình Teacher không chỉ cung cấp nhãn cứng (hard labels, ví dụ: "ảnh này là con mèo") mà còn đưa ra phân phối xác suất mềm (soft labels) – biểu hiện kiến thức về độ tự tin giữa các lớp. → Student học không chỉ từ nhãn thật, mà còn bắt chước cách Teacher suy nghĩ**.**
- Ví dụ:

    - Hard labels (ground truth): [dog: 1.0, cat: 0.0, rabbit: 0.0]
    - Teacher's soft prediction: [dog: 0.7, cat: 0.2, rabbit: 0.1]

    → Student học được rằng: "con vật này giống chó, nhưng cũng hơi - giống mèo".

3.2 Tác dụng của Knowledge Distillation

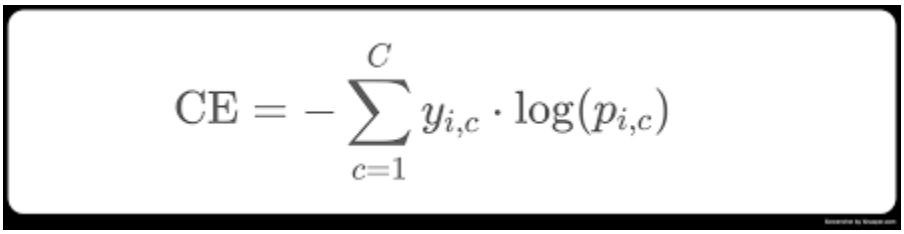
- Giảm kích cỡ của mô hình (Model Compression): KD cho phép bạn tạo ra một mô hình nhỏ gọn hơn (Student) nhưng vẫn giữ được hiệu năng gần với mô hình lớn (Teacher).
- Tăng tốc độ suy luận (Inference Speed): Mô hình Student thường đơn giản hơn nhiều → suy luận nhanh hơn, phù hợp cho:

    - Thiết bị di động (mobile, edge)
    - Hệ thống thời gian thực

- Cải thiện hiệu suất học (Better Generalization): KD giúp Student học được kiến thức mềm từ Teacher (soft labels) → biết được các mối quan hệ ngầm giữa các lớp.
  → Student sẽ:
    - Generalize tốt hơn trên dữ liệu chưa thấy
    - Giảm overfitting (so với chỉ học nhãn cứng)
- Truyền tri thức sâu từ mô hình mạnh (Knowledge Transfer):
    - Teacher có thể là mô hình phức tạp như Transformer, ViT, Ensemble, v.v.
    - KD giúp Student học lại các biểu diễn ẩn (độ tự tin, xu hướng...) mà mô hình đơn giản không thể tự học được
- Tăng độ chính xác cho mô hình nhỏ

3.3 Cách huấn luyện mô hình với Knowledge Distillation

- Bước 1: Chuẩn bị tập dữ liệu
- Bước 2: Lựa chọn mô hình: mô hình học sinh (student model) và mô hình giáo viên (teacher model)
- Bước 3: Finetuning lại mô hình giáo viên đã được huấn luyện sẵn
- Bước 4: Thực hiện huấn luyện mô hình học sinh: Dữ liệu sẽ được đưa vào mô hình học sinh và mô hình giáo viên. Mô hình học sinh dự đoán các lớp (classes) và đưa ra kết quả dưới dạng phân bố xác suất. Mô hình giáo viên cũng sẽ dự đoán và đưa ra kết quả dưới phân bố xác suất, kết quả đó được gọi là soft labels. Từ 2 phân bố xác suất trên ta tiến hành tính giá trị mất mát (loss function) của mô hình học sinh dựa trên:
    - Hard Loss (Loss_ce) – từ nhãn thật: Loss_ce = CrossEntropy (student_logits, ground_truth_labels)
    - Soft Loss (Loss_kd) – từ phân phối mềm của Teacher: teacher_soft = Softmax (teacher_logits / T), student_soft = Softmax (student_logits / T)
      → L_kd = KLDiv (student_soft, teacher_soft) * (T ** 2): T là độ nhiệt, độ nhiệt càng cao mô hình học sinh sẽ càng học được nhiều từ mô hình giáo viên
    - Loss tổng = Loss = α * L_ce + (1 - α) * L_kd: trong đó α là trọng số giữa soft và hard labels
- Giải thích khái quát về 2 hàm loss chính:
    - Cross-Entropy Loss (CE Loss): Đo độ sai lệch giữa phân phối dự đoán của mô hình và nhãn thật (ground truth)



$$\mathrm{CE} = -\sum_{c=1}^{C} y_{i,c} \cdot \log(p_{i,c})$$

Ảnh 13: Cross-Entropy Loss

    - Kullback-Leibler Divergence (KLDiv): Đo khoảng cách giữa hai phân phối xác suất (thường là soft labels)

$$KL(f,g) = \sum_{x \in \text{Support}(f)} f(x) \log\left(\frac{f(x)}{g(x)}\right)$$

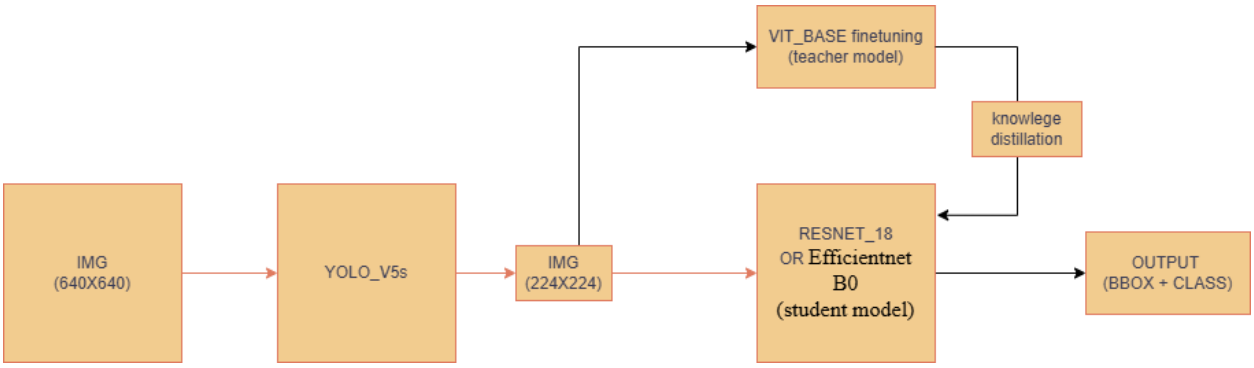
Ảnh 14: Kullback-Leibler Divergence Loss

3.4 Áp dụng Knowledge Distillation trong dự án

- Trong dự án Fall detection: em muốn sử dụng knowledge distillation nhằm nâng cao hiệu quả và độ chính xác của mô hình học sinh (Resnet18 hoặc Efficientnet-B0)
- Bởi vì hạn chế lớn nhất của những mô hình CNN là không có khả năng học được những đặc trưng toàn cục (global features), nên nếu áp dụng KD một cách hợp lý thì mô hình học sinh có thể học được những đặc trưng toàn cục. Từ đó nâng cao khả năng tổng quan hóa (Generalization) và giảm overfiting của mô hình học sinh

→ Cần chọn mô hình giáo viên phù hợp và đáp ứng được nhu cầu trên: em đề xuất ***Mô hình Visison Tranformer (VIT).***

- Sơ đồ pipleline của dự án khi có KD sẽ là:

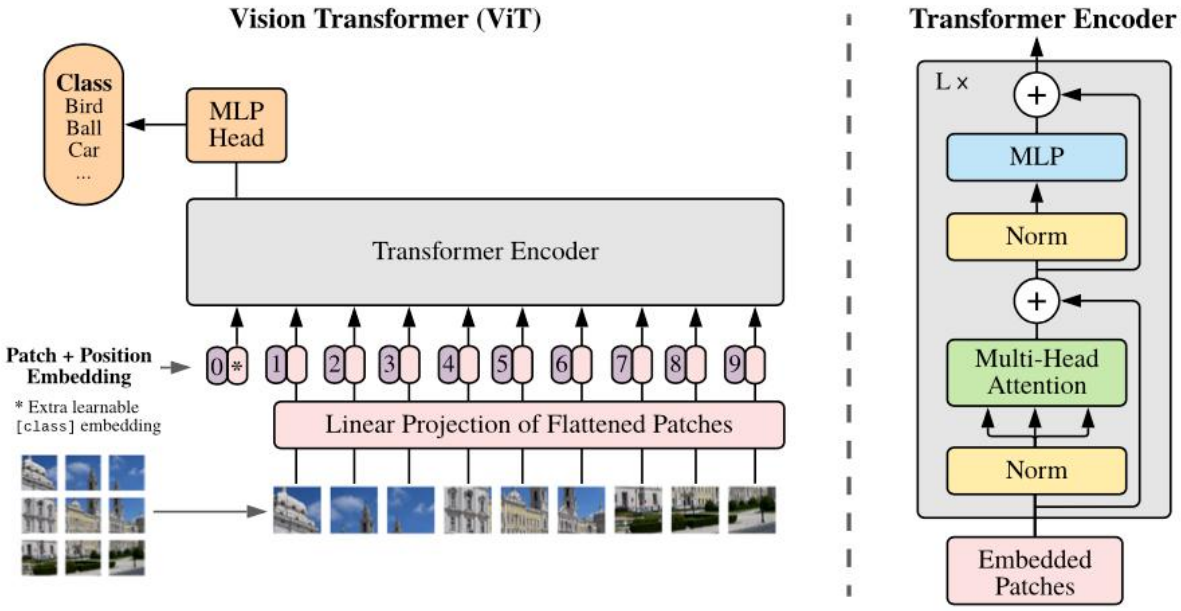


Ảnh 15: Chi tiết pipleline dự án với knowledge distillation

4. Lý thuyết Vision Tranformer

4.1 Kiến trúc mô hình Vision Tranformer

- **Giới thiệu**: Vision Transformer (ViT) là một mô hình mạng sâu ứng dụng kiến trúc Transformer – vốn nổi tiếng trong xử lý ngôn ngữ tự nhiên – vào lĩnh vực thị giác máy tính (computer vision) được xuất hiện lần đầu trong bài báo vào năm 2020, đặc biệt là các tác vụ như phân loại ảnh, phát hiện đối tượng, v.v.



Ảnh 15: Kiến trúc mô hình Vision Tranformer

- ViT là sự kết hợp giữa ý tưởng xử lý ảnh và Transformer gốc từ NLP. Thay vì dùng các lớp tích chập (CNN), ViT chia ảnh thành các mảnh nhỏ (patches) và xử lý như chuỗi từ.:
    - Patch Embedding Layer: Ảnh đầu vào được chia thành các patch có kích thước cố định (ví dụ 16x16), Mỗi patch sẽ được làm phẳng và đưa qua linear projection thành các vector có kích thước D
    - Positional Encoding: Vì Tranformer không có khái niệm vị trí, nên ta phải embedding vị trí và từng patch. Đi qua khối Positional Encoding sẽ tạo ra 1 vector cùng kích thước với vector patch đã làm phẳng rồi cộng 2 vector lại. Trong bài báo nghiên cứu về VIT, các nhà nghiên cứu đã thực nghiệm với nhiều kiểu position

embedding khác nhau, họ nhận thấy position embedding 2 chiều là phù hợp với mô hình nhất

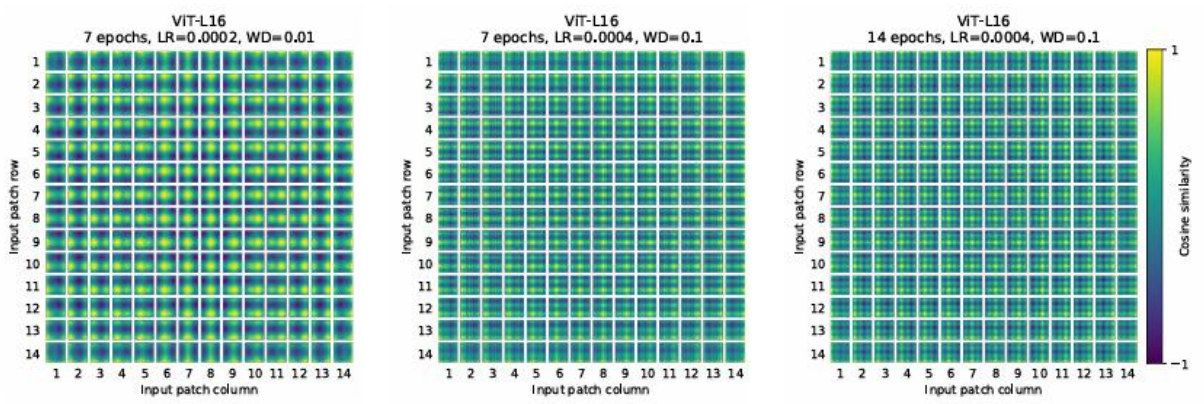


Figure 10: Position embeddings of models trained with different hyperparameters.

Ảnh 16: Minh họa sự thay đổi của position embedding 2 chiều với các tham số

- [CLS] Token: Thêm một token đặc biệt gọi là [CLS] vào đầu chuỗi patch → kích thước (1 × D). Token này được dùng để tóm tắt toàn bộ ảnh, và được đưa vào head phân loại sau này.
- Transformer Encoder Blocks: lấy cảm hứng từ khối encoder trong mô hình Tranformer. Bao gồm:
  + Multi-Head Self-Attention (MSA)
  + Layer Normalization + Residual
  + Feed-Forward Network (MLP): 2 lớp Linear + GELU
  + Layer Normalization + Residual
- MLP Head (Classification Head): Token [CLS] sau khi qua encoder được tách ra và đưa vào MLP để phân loại. Gồm một hoặc hai lớp fully connected, ra số lớp (num_classes).

4.2 Mục tiêu mô hình Vision Tranformer

- Thay thế CNN bằng Tranformer trong thị giác máy tính: ViT nhằm mục đích loại bỏ kiến trúc CNN truyền thống, vốn phụ thuộc vào kernel cục

bộ và pooling, bằng Transformer – một kiến trúc có khả năng học mối quan hệ toàn cục (global attention).

- Tận dụng sức mạnh của Self-Attention:
    - Học quan hệ giữa các vùng ảnh bất kỳ (dù ở xa nhau)
    - Giải quyết các giới hạn của CNN trong việc học ngữ cảnh toàn ảnh
- Mở rộng quy mô mô hình hiệu quả (Scalability): ViT được thiết kế để dễ dàng mở rộng kích thước mô hình (ViT-Base, ViT-Large, ViT-Huge...) mà vẫn duy trì hiệu quả huấn luyện khi có:
- Đơn giản hóa pipeline thị giác:
    - CNN yêu cầu nhiều thủ thuật: pooling, normalization, augmentation phức tạp...
    - ViT sử dụng cấu trúc đơn giản: chia ảnh thành patch + Transformer encoder + MLP head.
- Khám phá khả năng học vị trí (learnable position): ViT cho thấy Transformer có thể tự học cấu trúc không gian ảnh thông qua positional embedding, thay vì dùng cấu trúc cứng như CNN.

III. Huấn luyện và đánh giá mô hình

5. Lựa chọn bộ dữ liệu

- Kết hợp 2 bộ dataset về fall detection trên roboflow:
    - https://universe.roboflow.com/hong-nguyn-lixag/fall_detection-8vxyk/browse?queryText=class%3Afall&pageSize=50&startingIndex=0&browseQuery=true
    - https://universe.roboflow.com/roboflow-universe-projects/fall-detection-ca3o8/dataset/4
- Tổng quan: Sau khi kết hợp và tiền xử lý dữ liệu ta có bộ dataset với 30000. Mỗi ảnh sẽ thuộc về 1 trong 2 nhãn là fall (ngã) và normal (bình thường)

6. Huấn luyện và đánh giá mô hình

- ***Bước 1. Huấn luyện mô hình Yolov5s***
    - Chi tiết các bước:

DOWNLOAD ENVIROMENT YOLOV5

```
# @title DOWNLOAD ENVIROMENT YOLOV5

# Install the ultralytics package
!pip install ultralytics
# Clone the YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5

# Navigate to the cloned directory
!cd yolov5

# Install required packages
!pip install -r requirements.txt
```

Ảnh 17: Cài đặt môi trường Yolov5

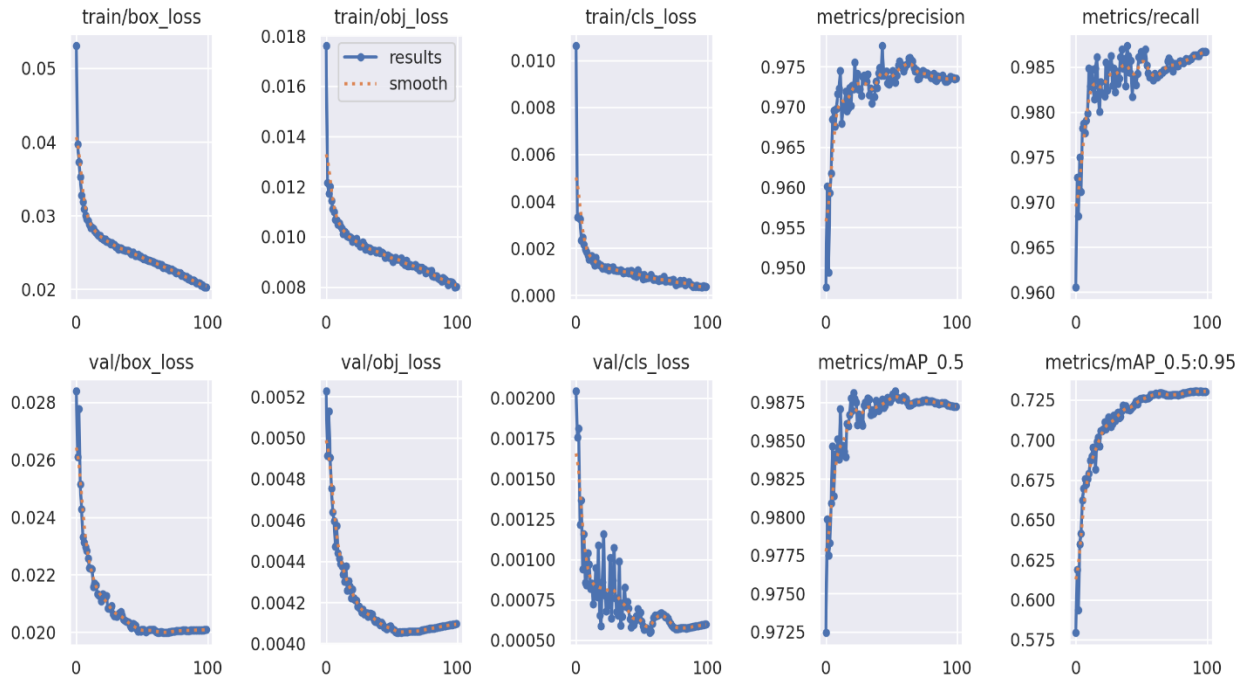
Trainning YOLOV5s

```
# @title Trainning YOLOV5s
!python /kaggle/working/yolov5/train.py \
    --img 640 \
    --batch 8 \
    --epochs 50 \
    --data /kaggle/input/data-yaml-falling/data.yaml \
    --cfg models/yolov5s.yaml \
    --weights yolov5s.pt \
    --name my-yolov5s-fall-detection \
    --project runs/train \
    --cache
```

Ảnh 18: Huấn luyện mô hình Yolov5

- Đánh giá: Trong quá trình train các loss giảm đều. Loss tại tập validation cũng giảm một cách ổn định và metric tăng khá tốt và đạt điểm khá cao

→ Mô hình được học hiệu quả

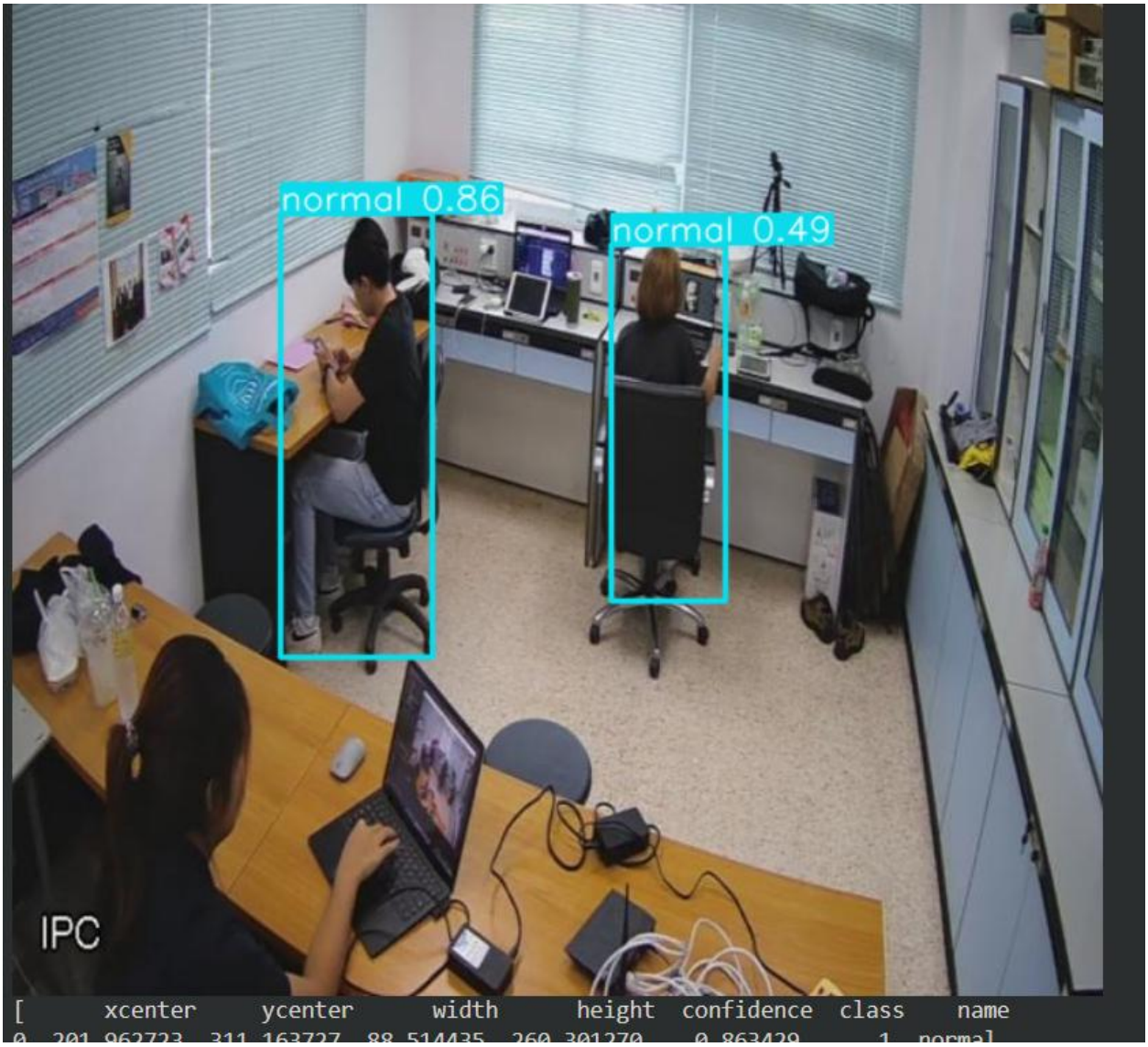


Ảnh 19: Huấn luyện mô hình Yolov5

INTERFACE YOLOv5_FINETUNING

```
# @title INTERFACE YOLOv5_FINETUNING
import torch
from PIL import Image
import matplotlib.pyplot as plt

# Load mô hình YOLOv5 từ file best.pt
model = torch.hub.load('ultralytics/yolov5', 'custom', path='/content/drive/MyDrive/BTL_TTCS/Model_yolov5_finetuning/best.pt')

# Đọc ảnh từ file
img = Image.open('/content/drive/MyDrive/BTL_TTCS/dataset_raw/dataset4/valid/images/0_jpg.rf.ab6865f09a98f46175982a12e4a9db21.jpg')

# Thực hiện dự đoán
results = model(img)

# Hiển thị kết quả
results.show()

# Lấy kết quả dưới dạng dataframe (Pandas)
print(results.pandas().xywh)  # In ra thông tin các đối tượng phát hiện (toạ độ, nhãn, độ tin cậy)
```

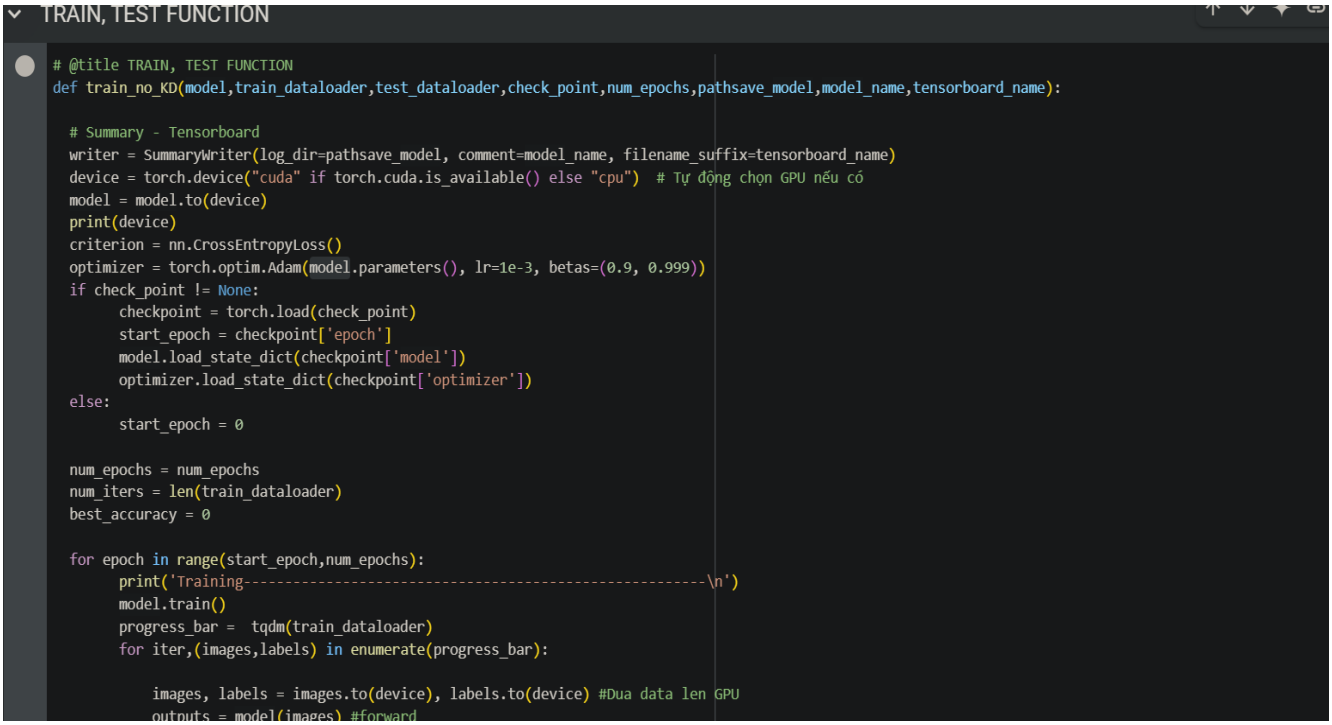
Ảnh 20: Code demo mô hình với 1 ảnh bất kì



Ảnh 21: Kết quả demo 1

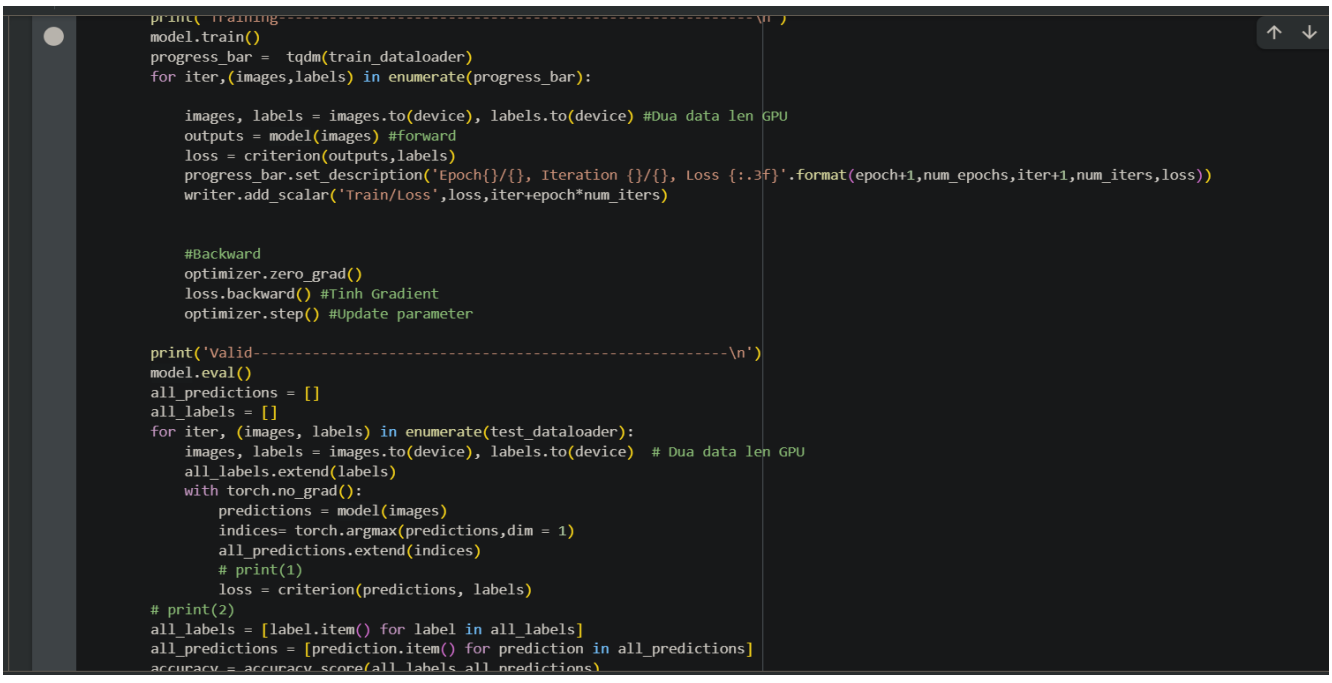
→ Mô hình đã nhận định được bbox của vật thể tuy nhiên confidence score trên mỗi class còn chưa cao. Ta sẽ tiến hành huấn luyện tiếp các mô hình phân loại để tăng onfidence score trên mỗi class.

- ***Bước 2. Huấn luyện và đánh giá mô hình Resnet 18 và Efficientnet B0***



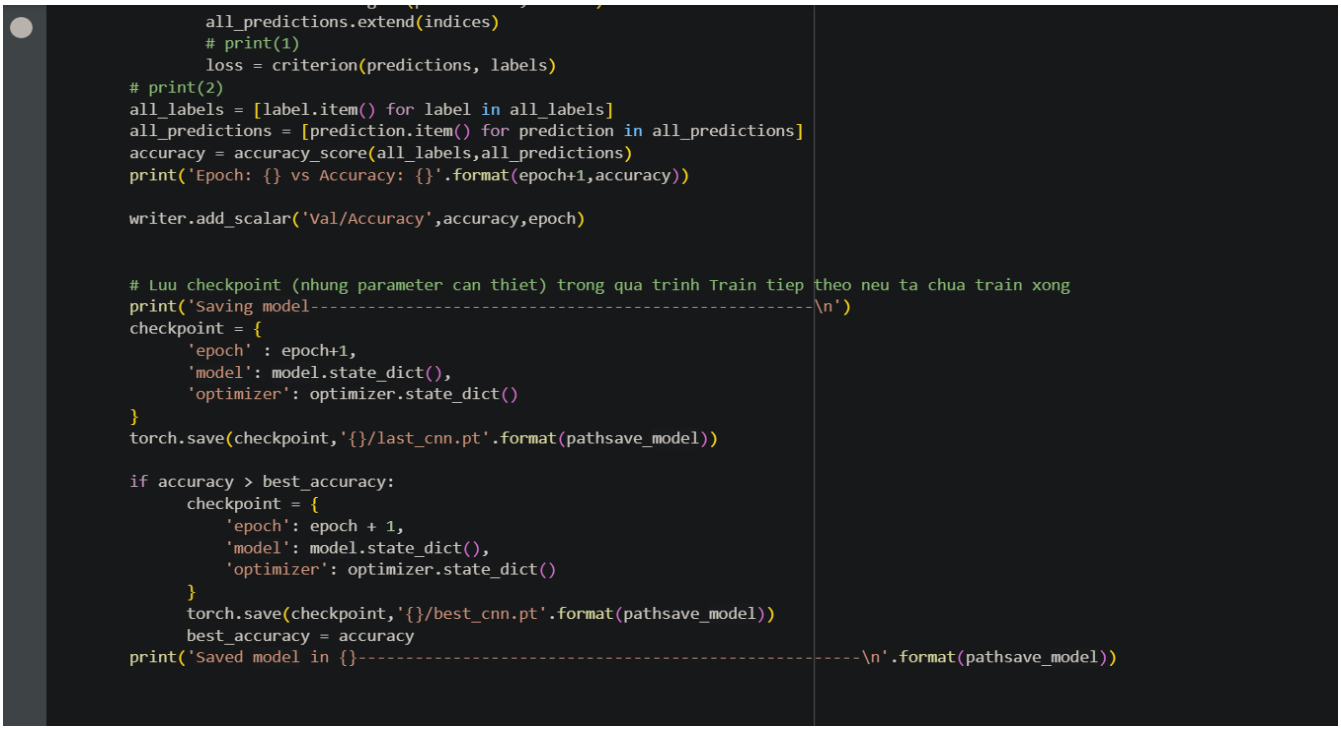
```
TRAIN, TEST FUNCTION

# @title TRAIN, TEST FUNCTION
def train_no_KD(model,train_dataloader,test_dataloader,check_point,num_epochs,pathsave_model,model_name,tensorboard_name):

  # Summary - Tensorboard
  writer = SummaryWriter(log_dir=pathsave_model, comment=model_name, filename_suffix=tensorboard_name)
  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  # Tự động chọn GPU nếu có
  model = model.to(device)
  print(device)
  criterion = nn.CrossEntropyLoss()
  optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, betas=(0.9, 0.999))
  if check_point != None:
        checkpoint = torch.load(check_point)
        start_epoch = checkpoint['epoch']
        model.load_state_dict(checkpoint['model'])
        optimizer.load_state_dict(checkpoint['optimizer'])
  else:
        start_epoch = 0

  num_epochs = num_epochs
  num_iters = len(train_dataloader)
  best_accuracy = 0

  for epoch in range(start_epoch,num_epochs):
        print('Training----------------------------------------------------------\n')
        model.train()
        progress_bar =  tqdm(train_dataloader)
        for iter,(images,labels) in enumerate(progress_bar):

            images, labels = images.to(device), labels.to(device) #Dua data len GPU
            outputs = model(images) #forward
```

Ảnh 22: Hàm huấn luyện không áp dụng KD 1



```
        print('Training----------------------------------------------------------\n')
        model.train()
        progress_bar =  tqdm(train_dataloader)
        for iter,(images,labels) in enumerate(progress_bar):

            images, labels = images.to(device), labels.to(device) #Dua data len GPU
            outputs = model(images) #forward
            loss = criterion(outputs,labels)
            progress_bar.set_description('Epoch{}/{}, Iteration {}/{}, Loss {:.3f}'.format(epoch+1,num_epochs,iter+1,num_iters,loss))
            writer.add_scalar('Train/Loss',loss,iter+epoch*num_iters)


            #Backward
            optimizer.zero_grad()
            loss.backward() #Tinh Gradient
            optimizer.step() #Update parameter

        print('Valid----------------------------------------------------------\n')
        model.eval()
        all_predictions = []
        all_labels = []
        for iter, (images, labels) in enumerate(test_dataloader):
            images, labels = images.to(device), labels.to(device)  # Dua data len GPU
            all_labels.extend(labels)
            with torch.no_grad():
                predictions = model(images)
                indices= torch.argmax(predictions,dim = 1)
                all_predictions.extend(indices)
                # print(1)
                loss = criterion(predictions, labels)
        # print(2)
        all_labels = [label.item() for label in all_labels]
        all_predictions = [prediction.item() for prediction in all_predictions]
        accuracy = accuracy_score(all_labels,all_predictions)
```

Ảnh 23: Hàm huấn luyện không áp dụng KD 2



```
            all_predictions.extend(indices)
            # print(1)
            loss = criterion(predictions, labels)
    # print(2)
    all_labels = [label.item() for label in all_labels]
    all_predictions = [prediction.item() for prediction in all_predictions]
    accuracy = accuracy_score(all_labels,all_predictions)
    print('Epoch: {} vs Accuracy: {}'.format(epoch+1,accuracy))

    writer.add_scalar('Val/Accuracy',accuracy,epoch)


    # Luu checkpoint (nhung parameter can thiet) trong qua trinh Train tiep theo neu ta chua train xong
    print('Saving model----------------------------------------------------------\n')
    checkpoint = {
          'epoch' : epoch+1,
          'model': model.state_dict(),
          'optimizer': optimizer.state_dict()
    }
    torch.save(checkpoint,'{}/last_cnn.pt'.format(pathsave_model))

    if accuracy > best_accuracy:
          checkpoint = {
              'epoch': epoch + 1,
              'model': model.state_dict(),
              'optimizer': optimizer.state_dict()
          }
          torch.save(checkpoint,'{}/best_cnn.pt'.format(pathsave_model))
          best_accuracy = accuracy
    print('Saved model in {}----------------------------------------------------------\n'.format(pathsave_model))
```

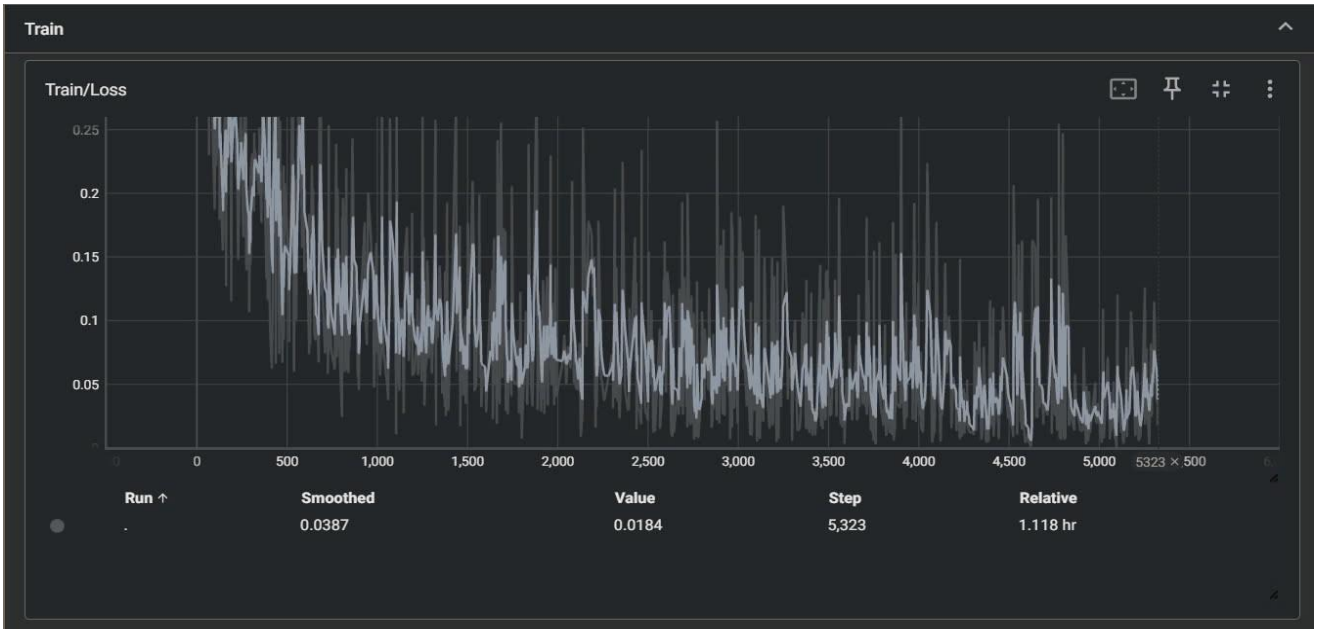
Ảnh 24: Hàm huấn luyện không áp dụng KD 3

- Bước 2.1 Huấn luyện mô hình Resnet 18:

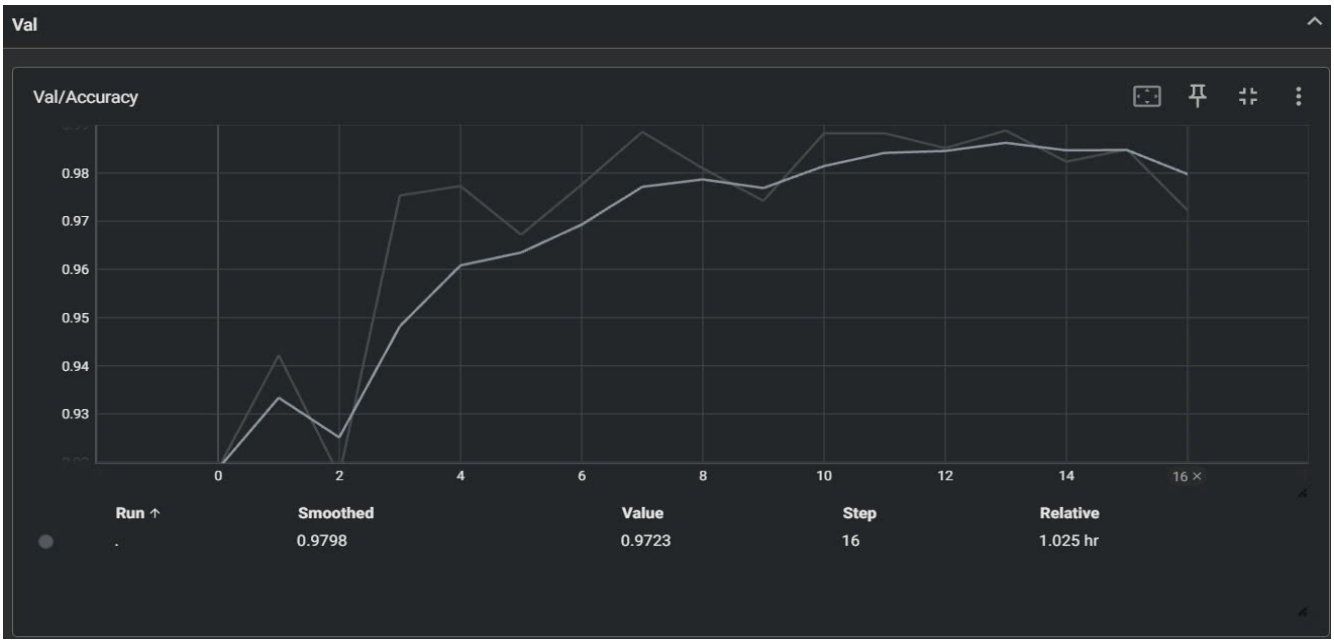
  + Code và test mô hình Resnet 18 với dataset Cifar10: https://colab.research.google.com/drive/1PQvuF6YF81kM3_q8_P17kx7cZDMLFlXb#scrollTo=HLosqso5ACkH

  + Huấn luyện mô hình:

  https://colab.research.google.com/drive/1t4b6yFvA-tCjtFbsp5sJeIyeiuEc_EBv#scrollTo=zhwiTCxOTEFy



Ảnh 25: Tensorboard của Loss_train trong quá trình train Resnet



Ảnh 26: Tensorboard của Accuracy_val trong quá trình validation Resnet

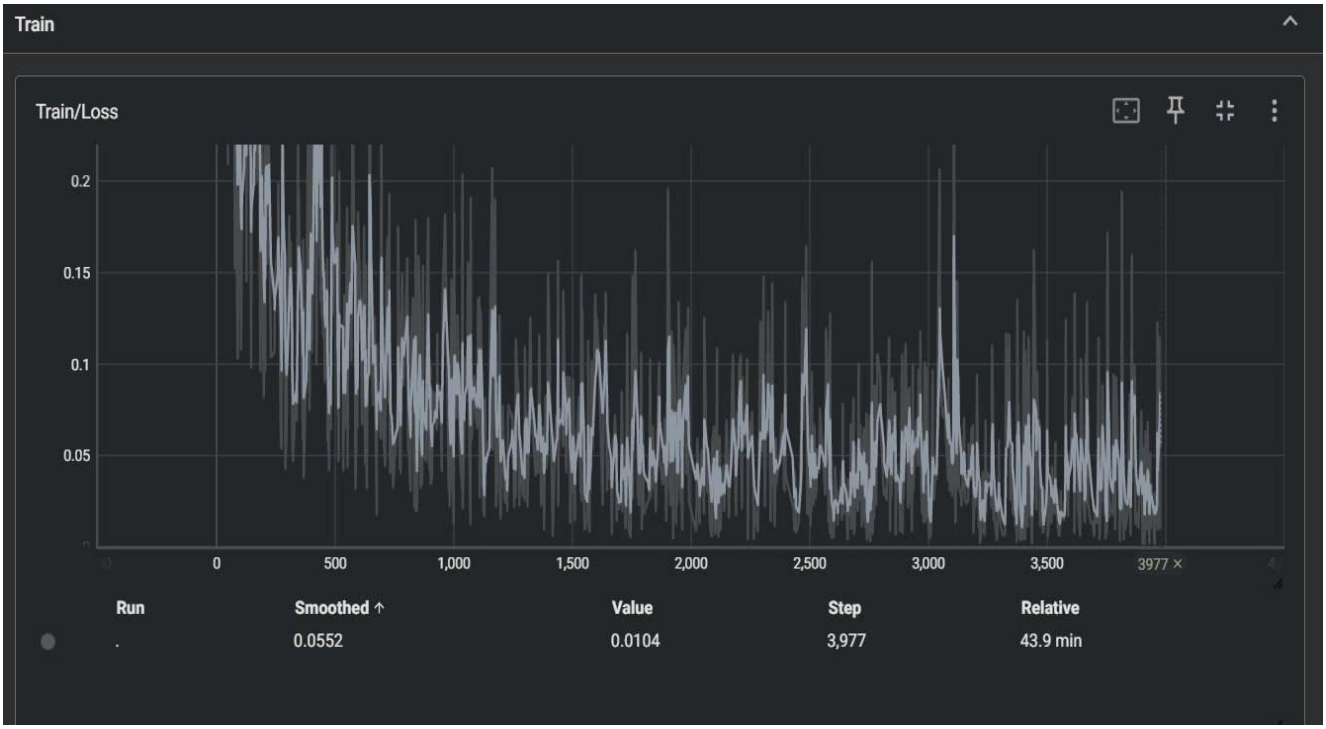
+ Đánh giá mô hình: Loss trong quá trình train giảm mạnh ở 4 Epochs đầu và giảm nhẹ trong những epochs sau. Accuracy trên tập validations tăng ổn định, đạt đỉnh tại epoch 14 và giảm nhẹ sau

epoch 14 và chững lại từ epoch 16→ *Resnet 18: accuracy 0.9823*→ Mô hình đạt độ chính xác cao nhưng overfiting nhẹ.

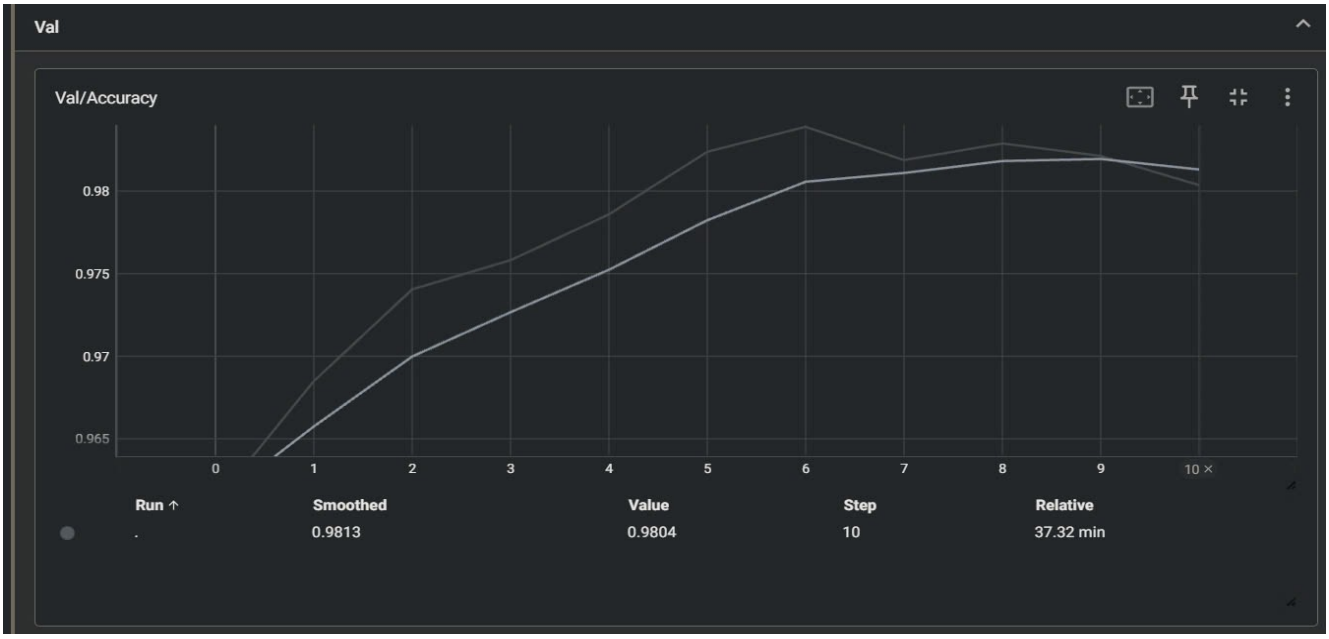
- Bước 2.2 Huấn luyện mô hình Efficientnet B0
  + Code và test mô hình Efficientnet B0 với dataset Cifar10: https://colab.research.google.com/drive/1_GlvLAHIUtKuLOmLYLpaY4p_DUgLqYFN

  + Huấn luyện mô hình:

  https://colab.research.google.com/drive/1t4b6yFvA-tCjtFbsp5sJeIyeiuEc_EBv#scrollTo=zhwiTCxOTEFy



Ảnh 27: Tensorboard của Loss_train trong quá trình train Efficientnet



Ảnh 28: Tensorboard của Accuracy_val trong quá trình validation Efficientnet

+ Đánh giá mô hình: Loss trong quá trình train giảm mạnh ở 2 Epochs đầu và giảm nhẹ trong những epochs sau. Accuracy trên tập validations tăng ổn định, đạt đỉnh tại epoch 9 và giảm nhẹ sau epoch 10 và chững lại từ epoch 11→ *Efficienet B0: accuracy 0.9838*. Mô hình đạt độ chính xác cao nhưng overfiting.

+ Bước 2.3 Đánh giá chung: Cả 2 mô hình đều có độ chính xác cao nhưng đồng thời cũng bị overfitting. Efficienet có độ chính xác cao hơn Resnet, cũng là mô hình có ít tham số hơn 1 nửa so với Resnet. → ***Efficientnet phù hợp với yêu cầu dự án*** là triển khai trên thiết bị GPU NVIDIA Jetson TX2 → Sử dụng thêm ***knowledge distillation*** để mô hình tăng độ tổng quát hóa và giảm overfiting

- ***Bước 3. Huấn luyện Efficientnet B0 với phương pháp knowledge distillation (KD)***

```python
# --------- STEP 2: Train student với soft labels đã lưu ----------
def train_with_KD_soft_labels(model_student, train_dataset, test_dataloader, soft_labels_path,
                              check_point, num_epochs, pathsave_model, model_name, tensorboard_name, T=2):

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model_student = model_student.to(device)
    writer = SummaryWriter(log_dir=pathsave_model, comment=model_name, filename_suffix=tensorboard_name)

    optimizer = torch.optim.Adam(model_student.parameters(), lr=1e-3, betas=(0.9, 0.999))
    criterion = nn.CrossEntropyLoss()

    if check_point is not None:
        checkpoint = torch.load(check_point)
        start_epoch = checkpoint['epoch']
        model_student.load_state_dict(checkpoint['model'])
        optimizer.load_state_dict(checkpoint['optimizer'])
    else:
        start_epoch = 0

    soft_labels = torch.load(soft_labels_path)  # (N, num_classes)

    train_loader = DataLoader(
        dataset=train_dataset,
        batch_size=64,
        shuffle=False,
        num_workers=2,
        drop_last=False,
        pin_memory=True
    )

    best_accuracy = 0
```

Ảnh 29: Hàm huấn luyện với phương pháp KD 1

```python
    for epoch in range(start_epoch, num_epochs):
        model_student.train()
        progress_bar = tqdm(train_loader, desc=f"Training Epoch {epoch + 1}/{num_epochs}")

        for i, (images, labels) in enumerate(progress_bar):
            images = images.to(device)
            labels = labels.to(device)
            soft_targets = soft_labels[i * 64:(i + 1) * 64].to(device)

            outputs = model_student(images)

            ce_loss = criterion(outputs, labels)
            kd_loss = nn.KLDivLoss(reduction='batchmean')(
                torch.log_softmax(outputs / T, dim=1), soft_targets
            )
            loss = 0.85 * ce_loss + 0.15 * (T ** 2) * kd_loss

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            writer.add_scalar('Train/Loss', loss.item(), epoch * len(train_loader) + i)

        # Validation
        model_student.eval()
        all_preds, all_gts = [], []

        with torch.no_grad():
            for images, labels in test_dataloader:
                images = images.to(device)
                labels = labels.to(device)
                outputs = model_student(images)
```

Ảnh 30: Hàm huấn luyện với phương pháp KD 2

```
            writer.add_scalar('Train/Loss', loss.item(), epoch * len(train_loader) + i)

        # Validation
        model_student.eval()
        all_preds, all_gts = [], []

        with torch.no_grad():
            for images, labels in test_dataloader:
                images = images.to(device)
                labels = labels.to(device)
                outputs = model_student(images)
                preds = torch.argmax(outputs, dim=1)
                all_preds.extend(preds.cpu().numpy())
                all_gts.extend(labels.cpu().numpy())

        acc = accuracy_score(all_gts, all_preds)
        writer.add_scalar('Val/Accuracy', acc, epoch)

        # Save model
        checkpoint = {
            'epoch': epoch + 1,
            'model': model_student.state_dict(),
            'optimizer': optimizer.state_dict()
        }
        torch.save(checkpoint, f'{pathsave_model}/last_cnn.pt')

        if acc > best_accuracy:
            torch.save(checkpoint, f'{pathsave_model}/best_cnn.pt')
            best_accuracy = acc

        print(f"Epoch {epoch + 1} | Accuracy: {acc:.4f} | Best: {best_accuracy:.4f}")
```

Ảnh 31: Hàm huấn luyện với phương pháp KD 3

- ***Bước 3.1 Finetuning model VIT_Base 16 với bộ dữ liệu của dự án***

  \+ Finetuning Vit_Base 16 bằng cách đóng băng (frezzing) toàn bộ backbone và chỉ để lại khối head để cập nhật tham số

```
[10]:
    # @title FINTUNING VIT
    from torchvision import  models
    from torchsummary import summary

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = models.vit_b_16(pretrained=True)
    model.heads.head = nn.Linear(model.heads.head.in_features, 2)  # binary classification

    for param in model.parameters():
        param.requires_grad = False

    for param in model.encoder.layers[-1].parameters():
        param.requires_grad = True

    for param in model.encoder.ln.parameters():
        param.requires_grad = True

    for param in model.heads.parameters():
        param.requires_grad = True

    # Huấn luyện
    model = model.to(device)
    train(
        model=model,
        train_dataloader=train_dataloader,
        test_dataloader=test_dataloader,
        check_point=None,
        num_epochs=15,
        pathsave_model='/kaggle/working/',
        model_name='VIT',
        tensorboard_name='run1'
    )
```

Ảnh 32: Code finetuning VIT_BASE 16

- ***Bước 3.2 Tiến hành huấn luyện Efficientnet B0 với phương pháp KD:***

  + Tạo trước 1 file soft labels theo bộ dữ liệu để trong quá trình huấn luyện mô hình giáo viên không phải dự đoán lại soft labels sau mỗi epoch. Điều nay làm giảm đáng kể thời gian huấn luyện tuy nhiên phải cẩn thận khi huấn luyện mô hình học sinh, không được trộn ngẫu nhiên dữ liệu vì điều này có thể làm sai khác nhãn ban đầu.

```python
# --------- STEP 1: Tính và lưu soft labels ----------
def compute_and_save_soft_labels(model_teacher, train_dataset, save_path, batch_size=64, T=2):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model_teacher.to(device)
    model_teacher.eval()

    dataloader = DataLoader(
        dataset=train_dataset,
        batch_size=batch_size,
        shuffle=False,
        num_workers=2,
        drop_last=False,
        pin_memory=True
    )

    all_soft_labels = []

    with torch.no_grad():
        for images, _ in tqdm(dataloader, desc="Computing soft labels in memory"):
            images = images.to(device, non_blocking=True)
            outputs = model_teacher(images)
            soft = torch.softmax(outputs / T, dim=1)
            all_soft_labels.append(soft.cpu())
            torch.cuda.empty_cache()


    soft_labels_tensor = torch.cat(all_soft_labels, dim=0)
    torch.save(soft_labels_tensor, save_path)
    print(f"Saved soft labels to: {save_path}")
```

Ảnh 33: Tạo trước soft labels từ mô hình giáo viên

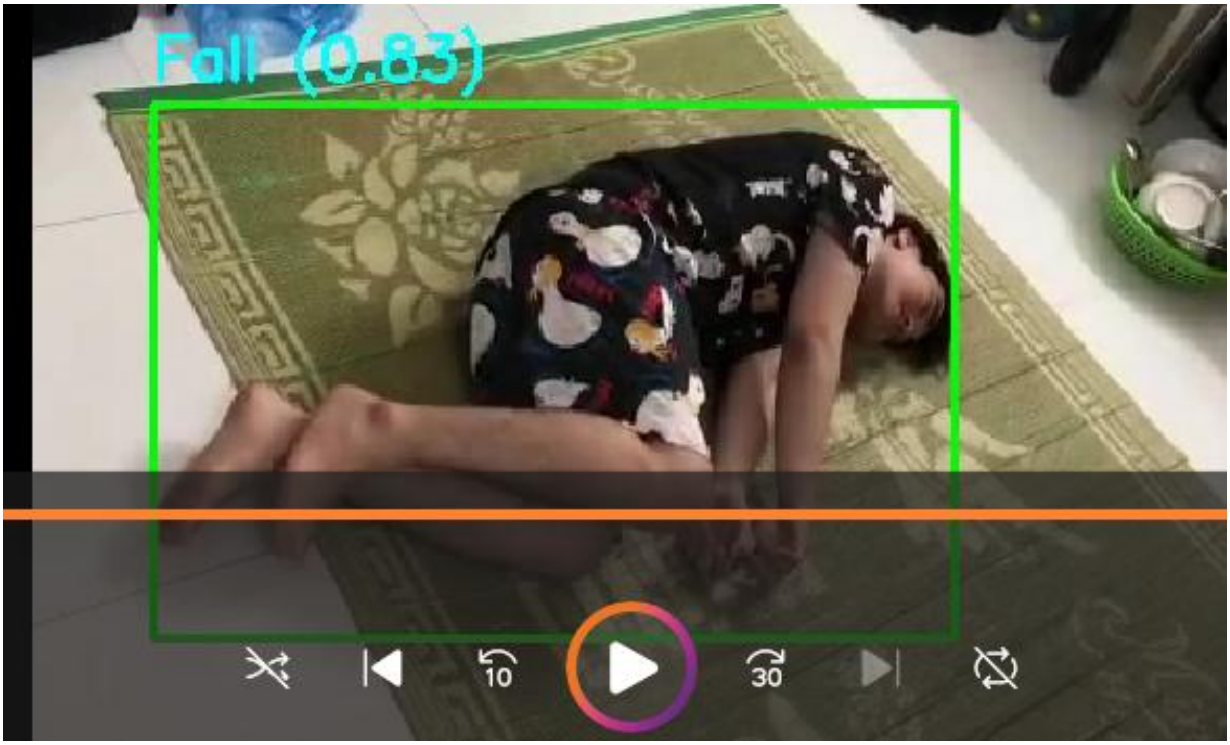
+ Code tiến hành huấn luyện mô hình mô hình Efficientnet B0 với phương pháp KD:

https://colab.research.google.com/drive/1t4b6yFvA-tCjtFbsp5sJeIyeiuEc_EBv#scrollTo=c89LcUWYIG0P

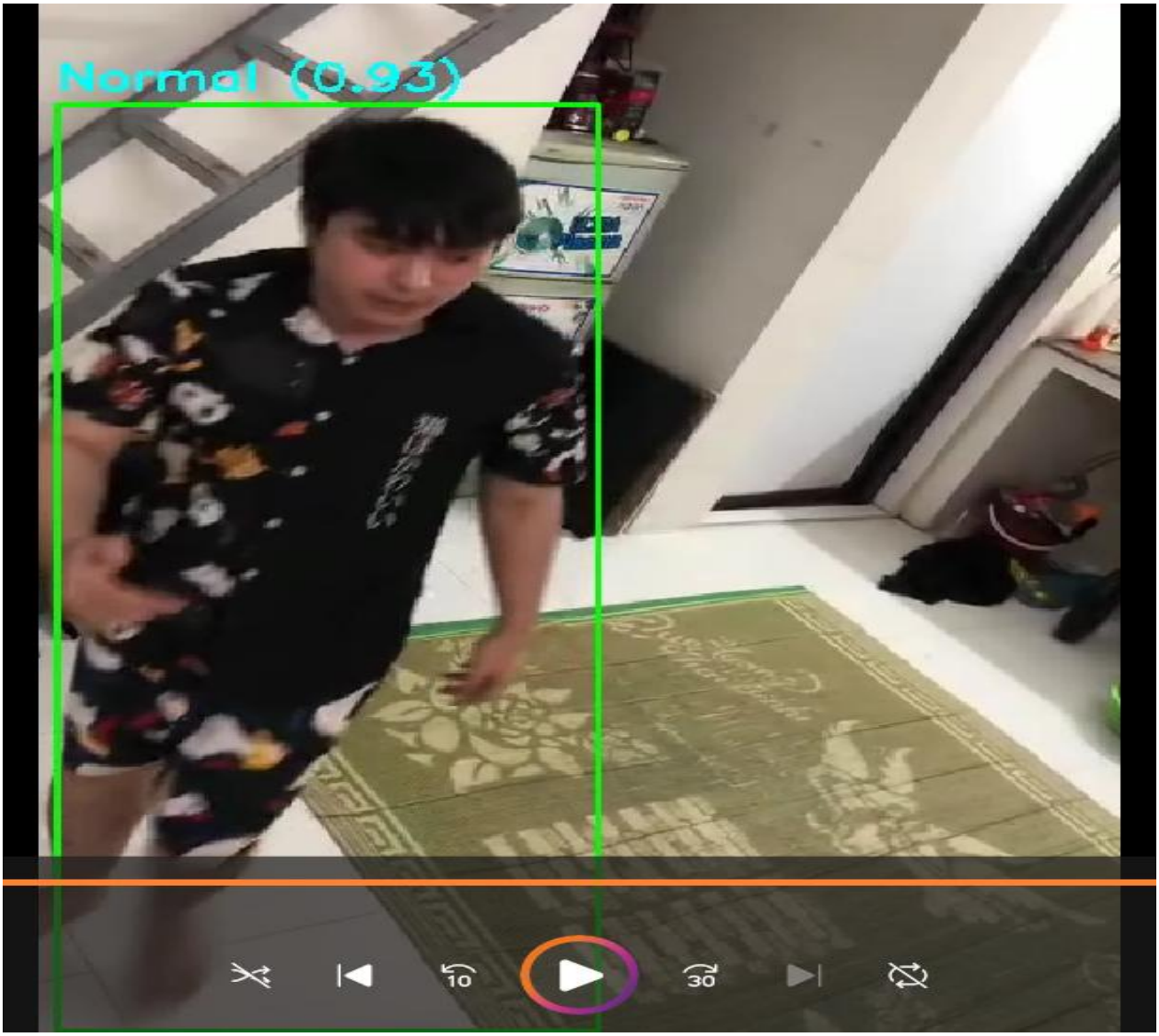
- ***Bước 3.3 Đánh giá:*** Sau khi huấn luyện với phương pháp KD, mô hình Efficientnet B0 có accuracy giảm nhẹ (0.9743) tuy nhiên mô hình đã học được những đặc trưng tổng quát và tránh bị overfiting → tốt hơn khi triển khai thực tế

IV. Demo dự án

- Kết quả khi chạy mô hình trên tính cá nhân:



Ảnh 34: Kết quả thử nghiệm 1



Ảnh 35: Kết quả thử nghiệm 2

Ảnh 36: Kết quả thử nghiệm 3

V. Kết luận và hướng phát triển

- Kết Luận: Qua dự án trên chúng ta đã thiết kế 1 pipeline mô hình sử dụng trong việc dự đoán tình trạng ngã của người ở nơi công cộng. Đồng thời trong dự án chúng ta đã tìm hiểu và nghiên cứu những mô hình phù hợp với bài toán đề ra, cùng với cách huấn luyện Knowledge Distillation đã cho thấy hiệu quả khi huấn luyện mô hình nhỏ được sử dụng trong thiết bị biên.
    - Tầm quan trọng: Bảo vệ người dễ tổn thương (người già, khuyết tật).
    - Hiệu quả: Yolov5s và EfficientNet B0 tối ưu trên Jetson TX2.
    - Knowledge Distillation: Giảm overfitting, tăng tính tổng quát.
    - Ứng dụng mở rộng: Giám sát, bảo vệ cộng đồng.
- Hướng phát triển:
    - Triển khai thực tế trên thiết bị biên
    - Áp dụng quantization để mô hình nhẹ và phù hợp hơn
    - Chuyển đổi về TensorRT: tôi ưu hóa với Jetson Tx2

- Nghiên cứu và tìm hiểu những mô hình phù hợp hơn

VI. Tài liệu nghiên cứu

- Bài báo về Efficientnet: https://arxiv.org/pdf/1905.11946
- Bài báo về Vision Transformer: https://arxiv.org/pdf/2010.11929
- Bài báo về knowledge distillation (KD): https://arxiv.org/pdf/1503.02531
- Tài liệu về Yolov5: https://docs.ultralytics.com/vi/models/yolov5
- Bài báo về SCPnet: https://arxiv.org/pdf/2303.06884