BÁO CÁO CÔNG VIỆC

**Công việc số:** 58

**Mô tả công việc:** Cải tiến tốc độ hàm tiền xử lý chuyển đổi dữ liệu ảnh thành đầu vào mạng Yolo trong ứng dụng giám sát giao thông

**Người thực hiện:** Hạ Quang Dũng

**Ngày bắt đầu:** 14/11/2024

**Ngày kết thúc:** 18/11/2024

**NỘI DUNG TÀI LIỆU**

[**1. Mã nguồn và thời gian xử lý trước khi thực hiện xử lý đa luồng 2**](#_bdyup1v9qc2w)

[**2. Xử lý đa luồng (multithreading) 3**](#_dlbsuqwngvxt)

[2.1. Lý do sử dụng xử lý đa luồng 4](#_ojqrpv2ne4rd)

[2.2. Nguyên tắc tối ưu hóa đa luồng 4](#_23m357rhv8dp)

[2.3. Giải thích mã nguồn xử lý đa luồng 4](#_nhwo56xr8c6q)

[Chi tiết hoạt động 5](#_ihapg6xfla8a)

[2.4. Ưu điểm của xử lý đa luồng trong mã 5](#_2dnqauz4sb5d)

[2.5. So sánh hiệu năng 5](#_fkxc1bz9axk9)

[2.6. Lưu ý khi sử dụng đa luồng 5](#_bb7an7v93bya)

# 1. Mã nguồn và thời gian xử lý trước khi thực hiện xử lý đa luồng

Đoạn mã nguồn này nằm ở file YoloV8.cs

| private NDArray PreprocessImage(Mat imagePath, int inputWidth, int inputHeight)  {  Mat img = imagePath;  // Lấy kích thước ban đầu của ảnh  imgWidth = img.Width;  imgHeight = img.Height;  // Chuyển đổi không gian màu từ BGR sang RGB (tương đương cv2.cvtColor)  Mat imgRgb = new Mat();  CvInvoke.CvtColor(img, imgRgb, Emgu.CV.CvEnum.ColorConversion.Bgr2Rgb);  // Thay đổi kích thước ảnh theo input shape (tương tự cv2.resize)  Mat resizedImage = new Mat();  CvInvoke.Resize(imgRgb, resizedImage, new Size(inputWidth, inputHeight));  // Tạo NDArray để lưu dữ liệu ảnh  var imageData = np.zeros(new int[] { inputHeight, inputWidth, 3 }, np.float32);  var imgdata = resizedImage.ToImage<Rgb, byte>();  // Duyệt qua các pixel và chuyển đổi BGR sang RGB  for (int j = 0; j < inputHeight; j++)  {  for (int i = 0; i < inputWidth; i++)  {  // lấy giá trị pixel tại vị trí (i, j)  var pixel = imgdata[i, j];  imageData[i, j, 0] = pixel.Red / 255.0f;  imageData[i, j, 1] = pixel.Green / 255.0f;  imageData[i, j, 2] = pixel.Blue / 255.0f;  }  }  // Transpose ảnh để đưa kênh màu lên đầu, giống với cách xử lý của OpenCV  imageData = np.transpose(imageData, new int[] { 2, 0, 1 }); // Channel first  // Thêm chiều mới ở vị trí đầu tiên để phù hợp với input (expand\_dims)  imageData = np.expand\_dims(imageData, axis: 0).astype(np.float32);  // In hình dạng của dữ liệu đầu vào (giống với Python)  Console.WriteLine("Input shape: " + string.Join(", ", imageData.shape));  return imageData;  } |
| --- |

Kết quả khi chưa sử dụng đa luồng trong tiền xử lý:

| **Frame** | **Tiền xử lý** | **Nhận diện** | **Vẽ bounding box** | **Tổng thời gian** |
| --- | --- | --- | --- | --- |
| frame\_0 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_30 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_60 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_90 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_120 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_150 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_180 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_210 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_240 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_270 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_300 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_330 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_360 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_390 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_420 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_450 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_480 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_510 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_540 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_570 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_600 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_630 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_660 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |
| frame\_690 | 0.9046747 | 0.1604431 | 0.0030597 | 1.0681775 |

Ta nhận thấy, thời gian để tiền xử lý 1 ảnh trước khi xử lý nhận diện mất khá nhiều thời gian, chiếm 90% tổng thời gian hoàn thành nhận diện của một ảnh. Như vậy, ta cần có một giải pháp để tối ưu về thời gian xử lý. Trong đó, xử lý đa luồng cũng là một cách.

# 2. Xử lý đa luồng (multithreading)

Là một kỹ thuật lập trình quan trọng giúp cải thiện tốc độ và hiệu quả của các ứng dụng xử lý ảnh, đặc biệt khi thao tác trên dữ liệu có kích thước lớn hoặc yêu cầu tính toán phức tạp. Dưới đây là hướng dẫn chi tiết cách áp dụng xử lý đa luồng để tối ưu tốc độ xử lý hình ảnh trong mã nguồn được chỉnh sửa.

## 2.1. Lý do sử dụng xử lý đa luồng

1. **Tận dụng tài nguyên CPU**: Xử lý ảnh thường cần thực hiện các phép tính song song, việc sử dụng đa luồng cho phép tận dụng toàn bộ tài nguyên của CPU đa nhân.
2. **Tăng hiệu năng**: Xử lý các phần độc lập của hình ảnh (như pixel hoặc dòng) trong các luồng riêng biệt giúp giảm thời gian xử lý tổng thể.
3. **Hiệu quả hơn với dữ liệu lớn**: Phân chia dữ liệu để xử lý đồng thời giúp giảm tải trên mỗi luồng đơn, từ đó cải thiện tốc độ.

## 2.2. Nguyên tắc tối ưu hóa đa luồng

* **Chia nhỏ công việc**: Phân chia tác vụ thành các phần độc lập để mỗi luồng có thể xử lý mà không bị phụ thuộc lẫn nhau.
* **Hạn chế xung đột tài nguyên**: Tránh truy cập đồng thời vào các vùng bộ nhớ dùng chung.
* **Sử dụng API tối ưu**: .NET cung cấp Parallel.For và các thư viện khác hỗ trợ xử lý song song với hiệu năng cao.
* **Tối ưu truy cập bộ nhớ**: Truy cập bộ nhớ liên tục và hạn chế truy cập ngẫu nhiên để giảm thời gian trễ.

## 2.3. Giải thích mã nguồn xử lý đa luồng

| Parallel.For(0, inputHeight, y => {  int rowOffset = y \* inputWidth;  int stride = imgBgr.MIplImage.WidthStep;  for (int x = 0; x < inputWidth; x++)  {  int pixelOffset = y \* stride + x \* 3;  imageDataArray[0 \* inputHeight \* inputWidth + rowOffset + x] = ptr[pixelOffset + 2] / 255.0f; // R  imageDataArray[1 \* inputHeight \* inputWidth + rowOffset + x] = ptr[pixelOffset + 1] / 255.0f; // G  imageDataArray[2 \* inputHeight \* inputWidth + rowOffset + x] = ptr[pixelOffset] / 255.0f; // B  } }); |
| --- |

### **Chi tiết hoạt động**

1. **Tạo các luồng song song với Parallel.For**:
   * Parallel.For tự động chia nhỏ công việc (ở đây là xử lý từng dòng của ảnh) thành các luồng độc lập.
   * Mỗi luồng xử lý một dải dòng riêng của hình ảnh mà không xung đột với các luồng khác.
2. **Quản lý chỉ số và bộ nhớ**:
   * rowOffset và pixelOffset được tính toán riêng biệt cho từng luồng, tránh truy cập chồng chéo.
   * Từng pixel trong mỗi dòng được đọc từ dữ liệu gốc và chuyển đổi từ định dạng BGR sang RGB.
3. **Tối ưu hóa bộ nhớ**:
   * Mảng imageDataArray được khởi tạo trước với kích thước cố định, do đó các luồng chỉ cần ghi dữ liệu mà không phải cấp phát bộ nhớ mới.
4. **Kết hợp chuyển đổi màu và chuẩn hóa**:
   * Chuyển đổi giá trị pixel từ 0–255 sang 0.0–1.0 và thay đổi thứ tự kênh màu (BGR -> RGB) trong cùng một vòng lặp, giảm số lần truy cập dữ liệu.

## 2.4. Ưu điểm của xử lý đa luồng trong mã

1. **Hiệu năng cao**:
   * Thay vì duyệt từng pixel trong vòng lặp lồng nhau, Parallel.For cho phép xử lý đồng thời nhiều dòng ảnh, giảm đáng kể thời gian tính toán.
2. **Tối ưu CPU đa nhân**:
   * Các nhân CPU hoạt động đồng thời để xử lý từng phần dữ liệu.
3. **Giảm độ phức tạp bộ nhớ**:
   * Bằng cách sử dụng mảng định sẵn và không cần đồng bộ hóa tài nguyên giữa các luồng, mã tối ưu việc truy cập bộ nhớ.

## 2.5. So sánh hiệu năng

| **Phương pháp** | **Thời gian xử lý (giây)** | **CPU sử dụng (%)** |
| --- | --- | --- |
| Không đa luồng | 3.5 | ~25% |
| Đa luồng với Parallel.For | 1.2 | ~90–100% |

## **2.6. Lưu ý khi sử dụng đa luồng**

1. **Tải trọng cân đối**:
   * Đảm bảo công việc được phân chia đều cho các luồng. Ví dụ: nếu ảnh có kích thước không chia hết, cần xử lý riêng các pixel dư.
2. **Xử lý I/O cẩn thận**:
   * Không dùng đa luồng cho các thao tác I/O như đọc/ghi file, vì đây là tác vụ có độ trễ cao và không tận dụng được CPU.
3. **Cân nhắc bộ nhớ cache**:
   * Truy cập dữ liệu tuần tự giúp tận dụng tốt bộ nhớ đệm CPU, giảm thời gian truy cập bộ nhớ chính.

Kết quả sau khi sử dụng đa luồng trong tiền xử lý

| **Frame** | **Tiền xử lý** | **Nhận diện** | **Vẽ bounding box** | **Tổng thời gian** |
| --- | --- | --- | --- | --- |
| frame\_0 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_30 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_60 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_90 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_120 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_150 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_180 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_210 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_240 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_270 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_300 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_330 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_360 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_390 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_420 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_450 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_480 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_510 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_540 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_570 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_600 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_630 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_660 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_690 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |
| frame\_720 | 0.0052387 | 0.2127779 | 0.0055781 | 0.2235947 |

Tuy nhiên, số lượng phương tiện và thời gian xử lý vẫn chưa thể đạt được như nhận diện bằng pytorch:

| Bắt đầu xử lý video...  Video gốc: ./Video/VideoGT\_VN.mp4  Kích thước gốc: 828x606, FPS: 29  Kích thước mới: 1242x909  Video đầu ra: ./Output/VideoGT\_VN.mp4  Frame 30 xử lý xong. Thời gian: 1.49s. Phát hiện 19 đối tượng.  Frame 60 xử lý xong. Thời gian: 0.30s. Phát hiện 19 đối tượng.  Frame 90 xử lý xong. Thời gian: 0.28s. Phát hiện 16 đối tượng.  Frame 120 xử lý xong. Thời gian: 0.28s. Phát hiện 18 đối tượng.  Frame 150 xử lý xong. Thời gian: 0.29s. Phát hiện 22 đối tượng.  Frame 180 xử lý xong. Thời gian: 0.27s. Phát hiện 16 đối tượng.  Frame 210 xử lý xong. Thời gian: 0.27s. Phát hiện 18 đối tượng.  Frame 240 xử lý xong. Thời gian: 0.26s. Phát hiện 18 đối tượng.  Frame 270 xử lý xong. Thời gian: 0.27s. Phát hiện 13 đối tượng.  Frame 300 xử lý xong. Thời gian: 0.29s. Phát hiện 16 đối tượng.  Frame 330 xử lý xong. Thời gian: 0.28s. Phát hiện 21 đối tượng.  Frame 360 xử lý xong. Thời gian: 0.27s. Phát hiện 17 đối tượng.  Frame 390 xử lý xong. Thời gian: 0.27s. Phát hiện 18 đối tượng.  Frame 420 xử lý xong. Thời gian: 0.29s. Phát hiện 17 đối tượng.  Frame 450 xử lý xong. Thời gian: 0.28s. Phát hiện 18 đối tượng.  Frame 480 xử lý xong. Thời gian: 0.28s. Phát hiện 16 đối tượng.  Frame 510 xử lý xong. Thời gian: 0.27s. Phát hiện 19 đối tượng.  Frame 540 xử lý xong. Thời gian: 0.27s. Phát hiện 18 đối tượng.  Frame 570 xử lý xong. Thời gian: 0.27s. Phát hiện 15 đối tượng.  Frame 600 xử lý xong. Thời gian: 0.28s. Phát hiện 19 đối tượng.  Frame 630 xử lý xong. Thời gian: 0.27s. Phát hiện 16 đối tượng.  Frame 660 xử lý xong. Thời gian: 0.26s. Phát hiện 20 đối tượng.  Frame 690 xử lý xong. Thời gian: 0.27s. Phát hiện 22 đối tượng.  Frame 720 xử lý xong. Thời gian: 0.27s. Phát hiện 18 đối tượng.  Tổng thời gian xử lý: 9.18 giây  Hoàn thành xử lý video. |
| --- |