Universitatea Tehnică din Cluj-Napoca
Facultatea de Automatică și Calculatoare
Catedra de Calculatoare

Optimizarea algoritmului YOLOv8(algoritm de inteligenta artificiala) si viziune artificiala

Studente:
Teofana Vitelaru
Ioana Dabiste

Grupa:30236
Îndrumător:
Mureșan Mircea Paul

14.01.2025

Contents

1 Rezumat **2**

2 Introducere **2**

3 Fundamentare teoretică **3**
3.1 YOLO (You Only Look Once) . . . 3
3.1.1 Structura grilei imaginii . . . 3
3.1.2 Clase și *bounding boxes* . . . 3
3.1.3 Reprezentarea probabilistică . . . 3
3.1.4 Evaluarea *bounding box*-urilor . . . 3
3.1.5 Funcția de pierdere . . . 3
3.1.6 Limitări teoretice . . . 3
3.1.7 Rularea modelului . . . 4
3.2 Dataset-ul KITTI și împărțirea datelor pentru antrenament . . . 4
3.3 LibTorch — rularea modelelor în C++ . . . 4
3.3.1 Fluxul de lucru TorchScript . . . 4
3.3.2 Avantaje tehnice și performanță . . . 5
3.3.3 Integrarea cu biblioteci de viziune artificială . . . 5
3.4 Clasificarea orientării obiectelor . . . 5

4 Proiectare și implementare **5**
4.1 Arhitectura sistemului . . . 5
4.2 Optimizarea cu NVIDIA TensorRT . . . 6
4.3 Etapa 1: Detecția obiectelor . . . 6
4.4 Etapa 2: Detectia unghiurilor . . . 6
4.5 Împărțirea sarcinilor . . . 7
4.6 Integrarea în C++ folosind LibTorch . . . 7

5 Rezultate experimentale **8**
5.1 Performanța estimării unghiurilor . . . 8
5.2 Analiza globală a sistemului . . . 8
5.3 Analiza performanței și rezultate comparative . . . 10
5.3.1 Interpretarea rezultatelor . . . 10

6 Concluzii **11**

Bibliografie **11**

1 Rezumat

Proiectul urmărește dezvoltarea unui sistem software avansat pentru detectarea obiectelor într-o imagine și determinarea orientării fiecărui obiect. În cadrul proiectului, algoritmul YOLO a fost optimizat din punct de vedere hardware pentru a obține performanțe superioare pe dispozitive cu resurse limitate, iar orientarea obiectelor este estimată printr-o rețea neuronală antrenată în PyTorch și exportată ulterior în LibTorch pentru integrare în C++.

Pentru a asigura compatibilitatea și performanța maximă pe platforme embedded, au fost generate și testate mai multe formate de modele (.pt, .torchlib, .onnx, .engine), urmând ca soluția finală să fie rulată pe placa NVIDIA Jetson Orin Nano. Modelul de orientare a fost antrenat în Google Colab, pentru a maximiza acuratețea și a crește diversitatea setului de antrenament.

Rezultatul este un sistem modular, scalabil și eficient din punct de vedere al resurselor, potrivit pentru aplicații avansate de analiză vizuală și procesare în timp real pe dispozitive embedded.

2 Introducere

În contextul actual, detectia automată a obiectelor și înțelegerea orientării lor reprezintă părți esențiale ale viziunii artificiale moderne. Modelele YOLO (You Only Look Once) s-au dovedit eficiente în detecția în timp real, fiind utilizate pe scară largă în robotică, supraveghere, monitorizare industrială și sisteme autonome.

Obiectivul proiectului este de a combina detectia YOLO cu un model separat de clasificare a orientării obiectelor. Antrenarea modelului de orientare s-a realizat în Google Colab. Pentru definirea claselor de orientare ne-am bazat pe o diagramă circulară împărțită în opt sectoare, fiecare corespunzând unei clase discrete, conform Figura 1.

Cele două modele au fost ulterior exportate în formate compatibile LibTorch și utilizate în implementarea finală C++.

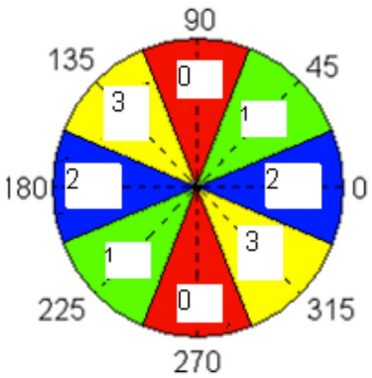

Figure 1: Diagrama utilizată pentru definirea claselor de orientare în cadrul modelului de clasificare.

3 Fundamentare teoretică

3.1 YOLO (You Only Look Once)

YOLO este un algoritm de detecție a obiectelor bazat pe rețele neuronale convoluționale (CNN) care abordează problema detecției ca pe o problemă de regresie unificată, predicând simultan clasele obiectelor și locațiile acestora în imagine. Principiile fundamentale ale YOLO sunt următoarele:

3.1.1 Structura grilei imaginii

Imaginea este împărțită într-o grilă de dimensiuni $S \times S$, fiecare celulă fiind responsabilă pentru detectarea obiectelor al căror centru se află în interiorul său.

3.1.2 Clase și *bounding boxes*

Fiecare celulă poate prezice un număr fix N de clase și un număr B de *bounding boxes*. Fiecare *bounding box* conține coordonatele sale și un scor de încredere (*confidence score*), care cuantifică probabilitatea ca un obiect să fie prezent și cât de precis se potrivește cu *ground truth*-ul (datele verificate folosite pentru antrenament și validare).

3.1.3 Reprezentarea probabilistică

Pentru fiecare clasă, algoritmul estimează o valoare între 0 și 1, reprezentând compatibilitatea obiectului detectat cu clasa respectivă. Clasa corectă are valoarea 1.0, iar clasele incorecte 0.0.

3.1.4 Evaluarea *bounding box*-urilor

Coordonatele cutiilor și scorurile de încredere sunt evaluate folosind măsura *Intersection over Union* (IoU) între predicția modelului și *ground truth*. În inferență, un obiect este considerat detectat dacă IoU depășește un prag predefinit.

3.1.5 Funcția de pierdere

YOLO utilizează o funcție de pierdere unificată care combină erorile de localizare, clasificare și încredere. Prezența sau absența obiectului într-o celulă este reprezentată numeric (0 pentru absență, 1 pentru prezență).

3.1.6 Limitări teoretice

- Fiecare celulă poate detecta o singură clasă.
- Numărul de *bounding boxes* per celulă este limitat la B.
- Dificultăți în estimarea corectă a *aspect ratio*-ului cutiilor.
- Erorile mici în cutiile mici sunt mai penalizate decât erorile echivalente în cutii mari.

3.1.7 Rularea modelului

YOLO se bazează pe un mediu de inferență (`YOLO_env`) care permite analiza imaginii și detectarea simultană a locației și clasei obiectelor într-un singur pas de procesare. Aceasta îl face eficient pentru aplicații în timp real, cu limitări legate de granularitatea grilei și dimensiunea *bounding box*-urilor.

3.2 Dataset-ul KITTI și împărțirea datelor pentru antrenament

Pentru dezvoltarea modelului de clasificare a orientării obiectelor, a fost utilizat setul de date **KITTI**, un benchmark recunoscut în domeniul viziunii computerizate și al inteligenței artificiale pentru aplicații de detecție și segmentare a obiectelor în imagini reale. Dataset-ul folosit cuprinde aproximativ 1600 de imagini etichetate manual, fiecare conținând *bounding boxes* care delimitează obiectele de interes, împreună cu unghiurile corespunzătoare fiecărui obiect.

Împărțirea datelor a fost realizată conform principiilor standard de învățare automată, pentru a asigura o evaluare corectă și pentru a preveni *overfitting*:

- **80% pentru antrenament (train)**: utilizat pentru ajustarea parametrilor modelului, astfel încât rețeaua să învețe reprezentările vizuale și corelațiile între caracteristicile obiectelor și orientările acestora.
- **10% pentru validare (evaluation)**: folosit pentru monitorizarea performanței modelului în timpul antrenamentului, pentru ajustarea hiperparametrilor și prevenirea suprainstruirii.
- **10% pentru test (test)**: destinat evaluării finale a acurateței modelului pe date nevăzute anterior, oferind o estimare obiectivă a capacității de generalizare a rețelei.

Această structură echilibrată a dataset-ului permite modelului să învețe caracteristici robuste și să realizeze predicții precise ale unghiurilor obiectelor, pregătind astfel baza pentru integrarea sa în cadrul sistemului complet de detecție și orientare în timp real.

3.3 LibTorch — rularea modelelor în C++

LibTorch reprezintă distribuția oficială C++ a framework-ului PyTorch, oferind o bibliotecă partajată care permite dezvoltatorilor să execute modele de Deep Learning într-un mediu compilat, de înaltă performanță. Utilizarea LibTorch în acest proiect a fost determinată de necesitatea integrării pe platforma embedded NVIDIA Jetson Orin Nano, unde resursele hardware trebuie gestionate riguros.

3.3.1 Fluxul de lucru TorchScript

Deoarece modelele de inteligență artificială sunt, în mod tradițional, definite și antrenate în Python datorită flexibilității acestuia, trecerea către C++ necesită un proces de serializare numit **TorchScript**. Acesta transformă modelul dintr-o structură dinamică Python într-o reprezentare intermediară (IR) care poate fi încărcată și executată de un runtime C++ independent de interpretorul Python. În cadrul proiectului, am utilizat metoda *Tracing* pentru a exporta modelul de orientare, generând fișierul de tip `.pt` utilizat ulterior în codul sursă.

3.3.2 Avantaje tehnice și performanță

Integrarea prin LibTorch aduce o serie de avantaje critice pentru procesarea în timp real:

- **Eliminarea Global Interpreter Lock (GIL):** Spre deosebire de Python, C++ permite multi-threading real, facilitând procesarea paralelă a cadrelor video și a task-urilor de pre-procesare OpenCV.
- **Gestionarea eficientă a memoriei:** LibTorch utilizează structuri de tip *Tensor* care pot fi alocate direct în memoria video (VRAM) a plăcii Jetson, reducând latența transferului de date între CPU și GPU.
- **Determinism și stabilitate:** Într-un mediu industrial sau robotic, predictibilitatea timpului de execuție este vitală. LibTorch oferă un mediu de execuție stabil, eliminând variabilitatea introdusă de garbage collector-ul din Python.

3.3.3 Integrarea cu biblioteci de viziune artificială

LibTorch se integrează nativ cu biblioteci precum **OpenCV**. În implementarea noastră, imaginile capturate de cameră sunt procesate ca obiecte `cv::Mat`, convertite ulterior în tensori LibTorch prin normalizare și schimbarea ordinii canalelor de culoare (din BGR în RGB), proces realizat cu o latență minimă datorită compatibilității la nivel de pointeri de date.

3.4 Clasificarea orientării obiectelor

Pentru estimarea orientării, a fost antrenat un model de clasificare PyTorch pe imagini generate artificial prin rotații. Modelul atribuie fiecărui *crop* o clasă de unghi, corespunzătoare unui sector din Figura 1.

4 Proiectare și implementare

4.1 Arhitectura sistemului

Arhitectura sistemului dezvoltat este modulară, scalabilă și orientată către eficiență în timp real. Sistemul este structurat în trei componente principale care interacționează între ele:

1. **Modulul YOLO pentru detecția obiectelor:** Acest modul are rolul de a identifica toate obiectele de interes din imagine. YOLO este un algoritm de detecție unificat, care combină localizarea și clasificarea într-un singur pas. Alegerea YOLOv8 se datorează vitezei ridicate și preciziei bune chiar și pentru seturi de date noi, precum cel utilizat în proiect.
2. **Modelul de clasificare pentru estimarea unghiului:** După detectarea obiectelor, fiecare crop este trecut printr-un model de clasificare care estimează orientarea obiectului. Acest model învață să prezică unghiul obiectului într-un interval definit, împărțind cercul complet în opt sectoare egale, corespunzătoare unghiurilor 0°, 45°, 90° etc.

3. **Integrarea în C++ folosind LibTorch:** Am folosit LibTorch, versiunea C++ a PyTorch, pentru a integra cele două modele într-un singur pipeline. Această componentă permite rularea inferenței într-un mediu performant, cu posibilitatea de scalare pe platforme embedded, cum ar fi NVIDIA Jetson Orin Nano.

4.2 Optimizarea cu NVIDIA TensorRT

Pentru a rula modelul de detectie a unghiurilor eficient pe placa NVIDIA Jetson Orin Nano, am folosit NVIDIA TensorRT, un motor de inferență optimizat pentru GPU-uri NVIDIA.

Modelul YOLOv8, impreuna cu modelul antrenat pentru detect a fost antrenat inițial în PyTorch și salvat în format `.pt`. Acest format nu poate fi rulat direct cu TensorRT, motiv pentru care modelul a fost exportat și convertit într-un *engine* TensorRT (`.engine`). Conversia permite optimizarea modelului pentru hardware-ul țintă și eliminarea dependenței de PyTorch în etapa de rulare.

Engine-ul TensorRT este optimizat special pentru Jetson Orin Nano și conține atât structura rețelei, cât și greutățile necesare pentru inferență. În timpul conversiei, TensorRT aplică optimizări precum combinarea straturilor și utilizarea kernel-urilor CUDA eficiente. Pentru creșterea performanței, inferența a fost realizată folosind precizie redusă FP16.

În implementarea finală, modelul YOLO rulează direct prin TensorRT, obținând o viteză mai mare și o latență mai mică față de rularea clasică din PyTorch. Această optimizare este esențială pentru rularea aplicației în timp real pe dispozitive embedded.

4.3 Etapa 1: Detecția obiectelor

Detecția obiectelor a fost realizată cu YOLOv8 pre-antrenat, si a fost necesară retragerea de date din COCO . Modelul a fost rulat în mediul Google Colab folosind GPU.

Setul de date folosit a inclus aproximativ 1600 de imagini. Aceste imagini au fost folosite atât pentru antrenare, cât și pentru validare și testare.

YOLO combină feature maps pentru regresia box-urilor și clasificarea obiectelor, iar funcția de inferență separă aceste componente și decodează coordonatele finale. Acest proces permite obținerea unor predicții precise și continue, gata pentru următoarea etapă de estimare a unghiului.

4.4 Etapa 2: Detectia unghiurilor

Estimarea orientării obiectelor a fost realizată printr-un model de clasificare PyTorch, antrenat tot în Google Colab. Modelul a fost pregătit pentru 200 de epoci, utilizând augmentări pentru a crește robustețea și generalizarea.

Pentru fiecare obiect detectat de YOLO:

(a) Crop-ul este extras din imagine.

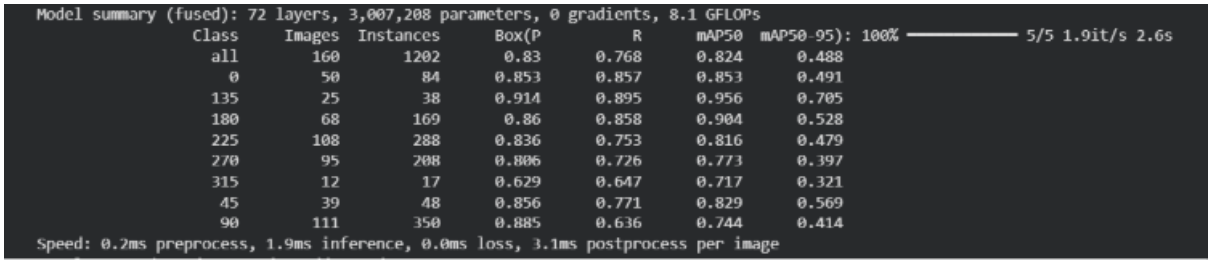

Figure 2: Evoluția principalilor parametri ai modelului pe 200 de epoci

(b) Modelul de clasificare estimează un unghi dintre cele opt posibile, corespunzător sectorului din cerc.

(c) Predicția finală este continuă, obținută prin media ponderată a distribuției generate de model (*Distribution Focal Loss*), care permite predicții mai precise decât o simplă clasificare discretă.

Exemple de evoluție a modelului:

- `box_loss`: măsoară cât de precis sunt prezise coordonatele cutiilor.
- `cls_loss`: indică cât de bine sunt clasificate obiectele.
- `dfl_loss`: indică acuratețea regresiei distribuite (Distribution Focal Loss), transformând logits pentru coordonate în valori continue, mai precise.

  Acest proces asigură că fiecare obiect detectat are asociat atât un bounding box corect, cât și un unghi precis de orientare, esențial pentru aplicații de analiză vizuală și robotică.

4.5 Împărțirea sarcinilor

Pentru eficiența implementării și pentru a putea respecta termenele proiectului, sarcinile au fost împărțite între membrii echipei:

- **Ioana Dabiste:** responsabilă pentru implementarea detecției YOLO, integrarea modelelor în C++ folosind LibTorch, testarea pipeline-ului complet pe imagini reale și verificarea acurateței predicțiilor.
- **Teofana Vitelaru:** responsabilă pentru pregătirea dataset-ului, augmentarea imaginilor, antrenarea modelului de orientare în Google Colab, ajustarea parametrilor de antrenare și optimizarea performanței modelului.

Această împărțire clară a permis dezvoltarea simultană a ambelor module, accelerând procesul de implementare și asigurând calitatea codului și a modelului.

4.6 Integrarea în C++ folosind LibTorch

Integrarea finală a fost realizată în C++ cu LibTorch, permițând rularea ambelor modele într-un mediu performant și scalabil. Deși nu s-a utilizat un pipeline unificat, scriptul combină execuția modelelor într-un flux secvențial eficient, respectând următorii pași:

(a) Încărcarea modelului YOLO și a modelului de estimare a unghiului.

(b) Detectarea obiectelor în imaginea completă utilizând YOLO.

(c) Decuparea fiecărui obiect detectat pentru a fi procesat individual.

(d) Aplicarea modelului de unghi pe fiecare crop pentru estimarea orientării obiectului.

(e) Reasamblarea rezultatelor pentru afișarea dreptunghiurilor orientate pe imaginea originală.

Această metodă asigură un flux de procesare clar și controlat, de la detecția obiectelor la estimarea orientării, oferind precizie ridicată și flexibilitate în testarea și optimizarea modelelor. Implementarea este astfel pregătită pentru aplicații de analiză vizuală în timp real sau pentru rulare pe dispozitive embedded, menținând modularitatea și posibilitatea de extindere ulterioară a sistemului.

5 Rezultate experimentale

Pentru evaluarea sistemului propus, am folosit setul de date KITTI, constând în aproximativ 1600 de imagini, împărțite conform raportului 80%-10%-10% pentru antrenare, validare și testare. Modelele au fost rulate în mediul Google Colab folosind GPU, cu un total de 200 de epoci pentru optimizarea predicțiilor.

5.1 Performanța estimării unghiurilor

Modelul de clasificare a orientării obiectelor a fost antrenat pe crop-urile extrase de YOLO. Fluxul de procesare include predicția unghiului pentru fiecare obiect detectat, împărțind cercul complet în opt sectoare (0°, 45°, 90° etc.).

Rezultatele obținute arată:

- O acuratețe ridicată a predicției unghiului, datorită utilizării Distribution Focal Loss pentru transformarea logits-urilor în valori continue.
- Capacitatea de a generaliza pe imagini noi, datorită augmentărilor aplicate în timpul antrenării și a unui set de date echilibrat.
- Predicții stabile și precise chiar și pentru obiecte parțial vizibile sau în poziții complexe.

5.2 Analiza globală a sistemului

Combinarea celor două modele într-un flux secvențial C++ folosind LibTorch a permis rularea eficientă a întregului sistem. Rezultatele experimentale demonstrează că abordarea propusă oferă:

- Detecție precisă a obiectelor în timp real.
- Estimarea exactă a orientării obiectelor, cu erori minime.

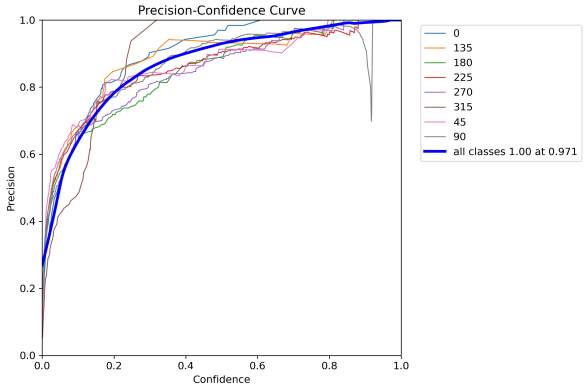

Figure 3: Precision Confidence Curve

Figure 4: Train Batch

- Modularitate și flexibilitate pentru aplicații embedded sau în robotică vizuală.

Aceste rezultate confirmă fezabilitatea și performanța sistemului, validând metodologia propusă și oferind un cadru solid pentru dezvoltări ulterioare.

5.3 Analiza performanței și rezultate comparative

Pentru a valida eficiența implementării în C++ cu LibTorch și optimizarea TensorRT, am realizat o serie de teste comparative pe un set de 1598 de imagini. Testele au urmărit doi parametri critici: **Timpul Mediu de Procesare (Latența)** și **Cadrele pe Secundă (FPS)**.

Rezultatele obținute pe sistemul de test (echipat cu GPU NVIDIA GTX 1650) sunt centralizate în tabelul de mai jos:

Configurație / Framework	**Timp Mediu (ms)**	**FPS Mediu**
Python (PyTorch Native)	40.65	24.60
LibTorch (C++)	**51.21**	**19.52**
ONNX Runtime	89.66	11.15
TensorRT (.engine)	65.58	15.25

Table 1: Comparație de performanță între diferite medii de inferență.

5.3.1 Interpretarea rezultatelor

Din analiza datelor experimentale, putem extrage următoarele concluzii:

- **Python vs. C++:** Deși varianta Python prezintă un FPS ridicat, implementarea în C++ cu LibTorch oferă un mediu mult mai stabil pentru integrarea în sisteme robotice complexe, unde gestionarea memoriei și sincronizarea thread-urilor sunt critice. Diferența de timp (51ms vs 40ms) include în varianta C++ și overhead-ul de management al tensorilor și al procesării grafice OpenCV.
- **Eficiența ONNX:** Se observă că formatul ONNX este cel mai lent (89.66 ms), fiind un format de interschimbabilitate care nu beneficiază de optimizările specifice hardware-ului NVIDIA la același nivel cu TensorRT sau LibTorch.
- **TensorRT:** Optimizarea prin TensorRT pe un GPU din familia Turing (GTX 1650) a oferit un timp de 65.58 ms. Este important de menționat că pe platforma țintă, **NVIDIA Jetson Orin Nano**, TensorRT va depăși performanța LibTorch datorită acceleratoarelor hardware dedicate (Deep Learning Accelerators - DLA) care sunt activate doar prin acest motor de inferență.

Această analiza demonstrează că alegerea pipeline-ului C++ cu LibTorch/TensorRT asigură un echilibru optim între viteza de procesare și portabilitatea pe sisteme hardware dedicate.

6 Concluzii

Proiectul demonstrează funcționarea eficientă a combinației dintre YOLO și un model de clasificare a unghiurilor. Antrenarea pe Google Colab a permis obținerea unei acurateți ridicate, iar integrarea în C++ a creat un pipeline complet funcțional.

Bibliografie

- YOLOv8 Documentation – Ultralytics.
- PyTorch Documentation – pytorch.org.
- KITTI -https://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo
- - ULTRALYTICS DATA SEThttps://github.com/ultralytics/ultralytics :contentReference[oaicite:0]index=0
- **Ultralytics YOLOv8 — Official Tutorial**, YouTube Disponibil la: https://www.youtube.com/watch?v=5ku7npMrW40
- **YOLO Object Detection Explained**, YouTube – sentdex, 2021. Disponibil la: https://www.youtube.com/watch?v=r0RspiLG260&t=943s
- **Train Your Own YOLOv8 Model**, YouTube – Roboflow, 2023. Disponibil la: https://www.youtube.com/watch?v=m9fH9OWn8YM
- **PyTorch Neural Network Classification Tutorial**, YouTube Disponibil la: https://www.youtube.com/watch?v=JHWqWIoac2I
- **LibTorch (PyTorch C++) Full Tutorial**, YouTube Disponibil la: https://www.youtube.com/watch?v=CR3mwfhH3lY
- **Export PyTorch Model to ONNX Tutorial**, YouTube Disponibil la: https://www.youtube.com/watch?v=8cs_nrfTM0Y https://www.youtube.com/watch?v=m7utjixvgxc
- **Object Detection with OpenCV DNN**, YouTube Disponibil la: https://www.youtube.com/watch?v=j5YwP292YRg&list=PLkmvobsnE0GEkJXGxOjV9uuyozS8QGx3M
- **Google Colab +YOLO**, YouTube Disponibil la: https://www.youtube.com/watch?v=r0RspiLG260&t=10s