DIY Video Filter for Jetson Nano

Group 4
Shelton Cyril K - 23103804
Rebeca Monis – 23103790
Class code – MSC-MAI (EE551)

Abstract

This project explores the implementation of video filters on the Jetson Nano platform using opencv_zoo. For human segmentation, PPHumanSeg[1] is used and the Viola-Jones Face Detection Technique, popularly known as Haar Cascades[2] is used for face detection. The objective is to create dynamic video effects, including background blur, background replacement, face distortion, and face replacement. The project leverages the computational capabilities of Jetson Nano to run these filters in real-time.

Introduction

The primary objective of this project is to leverage the computational power of the Jetson Nano to apply real-time video filters similar to those popularized by social media platforms like Instagram. The background replacement and blur techniques are also used by video conferencing platforms such as Zoom and Teams. It is also used in modern cameras/mobile-phones which have effects such as portrait mode which try to mimic the bokeh effect seen in higher end cameras with lenses which have really high-quality glass and design. Filters have become a ubiquitous feature in image-based applications, enhancing visual aesthetics and enabling users to express their creativity through personalized and stylized content. The Jetson Nano, with its GPU acceleration and parallel processing capabilities, provides an ideal platform to explore the implementation of such filters at the edge.

Project Objectives:

Implementation of the following filters and running them in real-time on the Jetson Nano

1) **Background Blur**:
   Focus on the subject and blur the background.
2) **Background Replacement**:
   Replace the background with an image or video stream, creating a dynamic environment.
3) **Face Distortion**:
   Apply facial landmarks detection and manipulation to distort facial features in a humorous or artistic way.
4) **Face Replacement**:
   Identify and replace the face in the video stream with a different image or video.
5) **A creative filter of our choosing:**
   We decided to replace the background with a GIF or video.

[1] https://github.com/opencv/opencv_zoo/tree/main/models/human_segmentation_pphumanseg
[2] https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

Methodology

Tools and Technologies[3]

1) Jetson Nano Kit
    - 1 x Jetson Nano 4GB Board
    - 1 x Power supply
    - 1 x Raspberry Pi camera 2
    - 1 x HDMI cable
    - 1 x 2.4Ghz USB Wi-Fi adapter
    - 1 x 64 GB microSD card
2) Jetpack OS
    - Customized Ubuntu OS
    - A Python 3.6.9 virtual environment
    - OpenCV 4.5.3 (CUDA enabled)
    - YoloV5 (revision 5.0)
    - TeamViewer (remote desktop application)
    - Example OpenCV and Yolo scripts
    - A python script to connect to eduroam
    - Tools and Technologies
3) Software used
    - OpenCV 4.8.1 built from source with CUDA support
    - Pillow Python imaging library
    - NumPy
    - Pipenv

Hardware Setup

1) Connect camera flex to the CSI port
2) Connect the barrel jack to a 5v 4A power supply
3) Insert the flashed SD card
4) Power on the Jetson

Software Setup

1) Ensure Python version > = 3.6.9 is installed
2) Build and install OpenCV 4.8.1 from source 4
3) Clone the code repo 5

[3] https://developer.nvidia.com/embedded/jetson-nano-developer-kit
[4] https://www.youtube.com/watch?v=BCNnqTFi-Gs
[5] https://github.com/sheltoncyril/ee-miniproj

System Design

Tech specs[6]:

- GPU: 128-core Maxwell based
- CPU: Quad-core ARM A57 @ 1.43 GHz
- Memory: 4 GB 64-bit LPDDR4 25.6 GB/s
- Video Encode performance: 4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
- Video Decode: 4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
- Camera: 2x MIPI CSI-2 DPHY lanes
- Connectivity: Gigabit Ethernet, M.2 Key E
- Display: HDMI and display port
- USB: 4x USB 3.0, USB 2.0 Micro-B
- Others: GPIO, I2C, I2S, SPI, UART

Modules:

- Camera: Raspberry Pi camera 2 [7] [8]

Code

GitHub: https://github.com/sheltoncyril/ee-miniproj
Also available locally in the same folder as this file.

Optimization

The most important reason to prefer a Jetson over a Raspberry Pi or any other SBC is the GPU. Utilizing parallel compute for such applications is paramount. Matrix operations are accelerated when using the GPU as they consist of simpler maths but just need to be performed repetitively millions or billions of times a second. Avoiding code which relies on the disk was also a big factor in improving performance, I don't think we should be reading or writing to disk if we are deploying this on the edge, especially if it's running off an SD Card. Caching data is also quite important, computing static content is pointless especially if we are resource constrained. While 4GB of memory and having a GPU is all nice, not every application will have a budget which allows for the use of a Jetson (of any kind). It's important to understand hardware limitations first before going all out on the model for peak (segmentation or classification) performance. We did experiment with the jetson-inference[9] library and while it yielded better FPS, we still decided to go with PPHumanSeg just because it was more accurate. For background replacement especially, the quality of segmentation matters a lot. Our code with a little bit more optimization can run really well without performance degradation, granted that it's not doing anything else but this. Probably using C instead of Python will yield better performance, at the cost of development time. Apart from this, keeping Big O in mind when coding is quite important.

"The loop is a powerful force, but it is up to us to decide whether we let it controls us or if we take control of it." - Unknown

---

[6] https://developer.nvidia.com/embedded/jetson-nano-developer-kit
[7] https://www.raspberrypi.com/products/camera-module-v2/
[8] https://www.raspberrypi.com/documentation/accessories/camera.html
[9] https://github.com/dusty-nv/jetson-inference

Future improvements

- Trying out libraries such as jetson-inference which are purpose built for Jetson.
- Leveraging HW support for other matrix operations.
- Using auxiliary sensors to gain additional information such as depth or using multiple cameras to infer depth information using stereo-vision. I had plans of using a radar sensor for depth information. For face replacement, it would be great if we could get a 3D map of the person's face so that we can "mould" the replacement image to their face. We can also infer directionality, blinking patterns, mouth movements and facial expressions and map it to the replacement image/model.
- Using a camera with AF can also help, makes it easier for the model to detect the subject when the subject is one of the things in focus.
- Using more optimized models for embedded platforms is also needed as with larger machines, performance becomes a matter of horizontal or vertical scaling (hardware) and for edge applications, cost, power-efficiency and performance are all defining factors of success.
- Utilizing newer platforms such as Jetson Orin will definitely help push boundaries. The C100 was released in 2019 and has 472 Giga-flops of performance whereas the newer Orin Nano has 20-40 Tera-flops. There are newer models which have even more performance like the Orin AGX with 200-275 TFLOPS.[10]
- Using more performant languages such as C, C++ or Rust can also greatly improve performance. While acquiring seasoned developers in such languages is not easy, the performance benefit for such applications is completely justifiable when it comes to the cost of hiring such developers.

Demo

YouTube: https://youtu.be/v4nKxlNwl-g
Also available locally in the same folder as this file.

Additional Information

- We ended up compiling OpenCV 4.8.1, opencv_zoo seemed like a very good option due to the quality of its models and support documentation available. The performance of HelloAIWorld demo from jetson-inference was extremely good but we didn't want to get into dockerization for this assignment due to the timeframe.[11] It also could classify different segments of the body so that could be used for additional information such as gesturing.[12]
- We decided to keep it simple and in general simple works better on such platforms. They are not meant to be Jacks of all trades and instead should only be used for specialized applications.
- The special filters feature can be enhanced by making the lightspeed-like streaks appear from behind the subject/s.
- The segmentation model needs some additional blurring around the edges to smoothen the transition between the background and subject.
- We had difficulty evaluating real world performance as our camera decided to die on us. Most of my testing was done using a virtual camera on OBS and piping that into the program.
- The face detection algorithm is not advanced enough and the face replacement performance is not good enough, smoothing needs to be implemented as the behaviour is currently very jumpy and even a single mis-classification can cause huge "faces" to appear out of artifacts in the image. Using a model that can distinguish features will be beneficial in also warping and conforming the image to the face.
- I had ideas of adding a ring of LEDs around the camera to give feedback to the person in the frame about staying in frame and also feedback about staying in focus. This was again abandoned due to the timeframe.

[10] https://en.wikipedia.org/wiki/Nvidia_Jetson
[11] https://github.com/dusty-nv/jetson-inference
[12] Using multi-human 640x360: https://github.com/dusty-nv/jetson-inference/blob/master/docs/segnet-console-2.md