CS 498 Final Project - FA20

RURAL HOME SECURITY
WITH OPENCV ON NVIDIA JETSON

Team MvF (Makers vs Fakers)

Ryan Rickerts (ryanjr3) and Shreyas Byndoor (byndoor2)

Video link: https://mediaspace.illinois.edu/media/t/1_21hlub1t
Github: https://github.com/theRocket/banditland-detection

Motivation:

In our project proposal, we highlighted our objective of building a Smart Home Security System. Ryan's house is located in a forest in a remote part of NW Washington state. Since the population in his neighborhood is sparse, home security is of utmost importance since he doesn't see many humans around unless he expects a visitor. The most popular smart security system in the United States is Google's Nest services and Ryan has the same system installed on his property. In addition to his safety, he would try to use Google Nest for verifying AirBnB rentals to determine the number of guests that arrive at his property and the exact time at which they would arrive or depart. This feature that Nest offered was not convenient because he would have to log on to Google's servers and view large portions of video footages to determine when they arrive. The Nest was also not efficient in terms of home security as there have been numerous occasions where Ryan has been alerted about false positives. Shadows from swaying tree branches, passing cars, and insects have triggered alerts and this illustrates the inaccuracy of the system. In addition to the installation costs, Ryan has been spending $10/month for this service - The Nest camera looking out for strangers, Mobile alerts, and video footage backups. Another thing to note is that the computer vision algorithms for computer vision are all running on the cloud which adds latency and is also not reliable in case network connectivity is compromised.

Our goal is to build a much more efficient and cheaper Smart Home Security System. Firstly, we plan to do all of the computer vision computation on the edge device itself. We opted for NVIDIA's Jetson suite of devices to aid in this effort. Ryan purchased the

mightier Xavier NX board (~$400) and a wide-angle lens so that he can legitimately replace his Google Nest. He also intends to use the Xavier NX for other machine learning projects in the future. Shreyas opted for the Jetson Nano 2GB which costs only $100 and comes with a 128 core Nvidia Maxwell GPU. These devices are small yet extremely powerful for Deep Learning on the edge and that is one of the primary reasons we decided to try out the potential of these devices. We also used Raspberry Pi v2 cameras which are equipped with the Sony IMX219 sensor. Shreyas had a NoIR camera with the same specs and this had better night vision capabilities.

We implemented an object detection neural network on the device (similar to lab1 mobilenet_ssd) and also leveraged the CUDA libraries for parallel processing. This would accelerate inference and the latency of classification would be far lower than the Google Nest. If a human was detected, we sent a text message to our phones by making use of AWS's Simple Notification Service (SNS). In addition to this, the captured video footage would be stored onto the Jetson so that we could view the activity at a later point in time. An important feature to note is that the video footage would be recorded only if a person is detected. This way, we can conserve a lot of storage space. Google Nest publishes video footage to the cloud constantly and this is expensive, so they only retain 10 rolling days of footage at the $10/month subscription price.

We identified multiple areas to make our smart security system more efficient and hopefully, these can replace the Google Nest which is currently in use at many houses.

Technical Approach:

After receiving the Jetson Nano/ Xavier, the first step was to flash the Jetpack 4.4.1 OS onto these devices. We explored the "Hello AI World" tutorials which showed how to accelerate DNN inference for object detection algorithms. Initially, we used test images and videos stored in the Jetson to evaluate the performance of the classification inference. The object detection model we settled with was mobilenet_ssd since it was the most lightweight and the feature where it reports the class an object belongs to along with a confidence percentage is profitable to us. We could trigger alerts when the confidence level is high so that false positives are reduced, which was one of the biggest problems of Google Nest.

The next step was to install the camera onto the Jetson. This was done through the MIPI-CSI 2 interface and once this was complete, we could run the object detection algorithms through the camera in real-time. The model we finalized on using was the mobilenet_ssd.

After a person was detected, it was important for us to assign an ID to him/her and track as long as he/she is in the view of the camera. This is because we are maintaining a count of the number of people detected and if a unique ID isn't assigned to a person, he/she might be calculated multiple times in future frames. To prevent such miscalculation, a person detected must be identified with the same ID as long as the camera can see the person. PyImageSearch has been a resourceful blog that helped us throughout the project and we borrowed the techniques from this specific blog post to aid in the unique tracking of people detected by the camera - https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/.

The next step to implement was storing the video frames where people are detected. The concept behind this is to write the OpenCV frames to a video file if an event of importance occurs in the frame. This must continue until the event ends and must therefore be a continuous stream of frames from the start of the event to the end of the event. In our case, this particular event is the detection of a person. Once again, a PyImageSearch blog, https://www.pyimagesearch.com/2016/02/29/saving-key-event-video-clips-with-opencv/#pyi-pyimagesearch-plus-optin-modal, aided us in implementing the feature. The above blog shows how the author is able to store 4 video files and the recording for each of them is triggered by a green ball coming within the view of the camera. We followed a similar approach and the recording would occur as long as there was at least one person in the view of the camera. When a person comes within the view of the camera, the timestamp is noted and a video file is created with this timestamp. This way, it is easy for the user to locate the video footage file based on the time they received a text message alert. Storage is also conserved since we do not store video files when no person is detected and this is the case for majority of the day.

Another feature we implemented was alerting the user when a person is detected through a text message by using the AWS Simple Notification Service. A message is sent when a person is immediately detected. Therefore, whenever a video output file is created for a timestamp, a message is sent to the user.

Implementation Details:

Nvidia's recommended OS to use on the Jetson is the Jetpack 4.4.1. The Jetson Nano ran into some problems upon installation of the OS: Trying to install software upgrades would break the bootloader and Shreyas had to reinstall a couple of times until finding a fix to hold off updates:

```
sudo apt update
sudo apt-mark hold systemd
sudo apt upgrade
Reboot
```

After installation, we played around with some Deep Learning models through Nvidia's Hello AI world tutorials. The main dependencies that were installed were provided in the repo and it consisted of `Jetson utilities and Jetson inference` (https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo-2.md). These packages enable Jetson to make use of its GPU for acceleration. Here, we were able to see the performance of object detection, mobilenet-ssd-v2 model on a camera stream. Setting up the MIPI-CSI-2 camera stream through a Jetson Utilities function was easy.

```
camera = jetson.utils.videoSource("csi://0")
```

After this, the object detection was very smooth and could detect accurately with an impressive frame rate of around 20-30 frames per second (fps). The Jetson utilities and Inference modules are able to achieve this by using their own objects and CUDA methods. The biggest drawback is that these methods are not scalable to other applications and it is not friendly to modification. In other words, it wasn't possible to write the Jetson Frame to an output file (based on events) or do other sorts of customized processing compared to what the capabilities that an OpenCV frame provides.

As a result, we explored methods to first stream using the OpenCV `cv2.VideoCapture`. The raspi camera has to be set up with the following gstreamer pipeline:

```
camSet='nvarguscamerasrc sensor-id=0 wbmode=2 tnr-mode=2 tnr-strength=1'
camSet+=' ! video/x-raw(memory:NVMM), width=3264, height=2464, framerate=21/1,
format=NV12'
camSet+=' ! nvvidconv flip-method=2 ! video/x-raw, width=800, height=600, format=BGRx'
camSet+=' ! videoconvert ! video/x-raw, format=BGR ! appsink'
camera=cv2.VideoCapture(camSet)
```

This is an important configuration to include when using MIPI-CSI cameras with Jetson. Through this OpenCV frames could be read, converted to a CUDA tensor and ssd-mobilenet configured through Nvidia's Jetson inference module could perform the detection on the frame. This would theoretically enable the GPU acceleration. However upon further processing of the image, like adding text boxes, etc, the final frames-per-second (FPS) was around 10-12 FPS. Adding the video storage and AWS Notify features would further decrease the FPS. We finally found a good way to do the detection without compromising on the FPS and the method was explained in this blog - https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/. The idea is to

drop some frames while running detection (but still directly stream them) because the results of analyzing these frames will be similar inside the same second (param is --skip-frames). This resulted in our entire video streaming and recording at 20 frames per second.

Once a frame is processed (i.e. passed through the neural network), an object is returned which consists of the various class IDs and the positions where they were detected. We are interested only by the "person" class and if we find it, we assign that object a unique ID and it is highlighted in the output frame with a green dot (center of the rectangular detection box). This person will now be tracked as long as he/she is visible to the camera (code in `people_counter.py`).

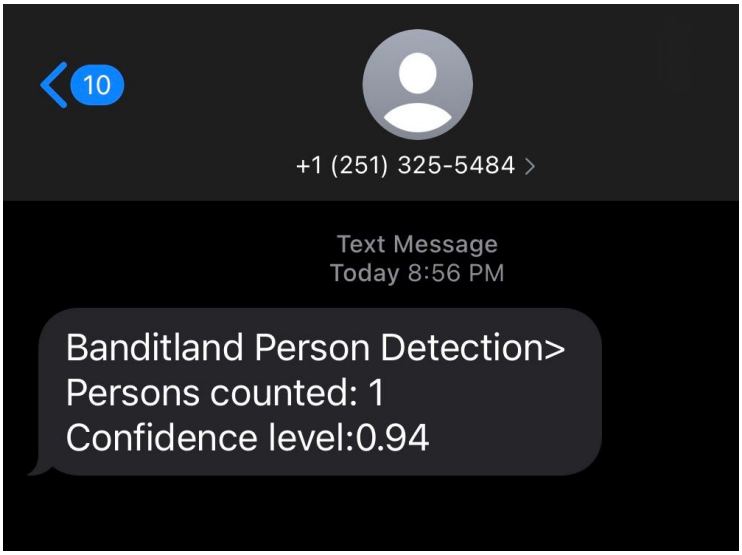
That is Ryan and he is being tracked by the camera. We also see that there is only 1 person in the frame.
https://www.youtube.com/watch?v=CAM4KO4BITI

A useful package that was provided through pyImageSearch was the `KeyClipWriter`. This provides an object that can be used to store a frame and also write to the output video file. So once a person is detected, we retrieve some of the frames before the current frame using the `KeyClipWriter` array of objects and keep writing frames to the video files until the person detection occurs and tracking begins. Python's `datetime` module was used to retrieve the timestamp and name the video file. Above is an example of one of

our video files uploaded to YouTube. Once the person falls out of view or the confidence level drops below the pre-configured value (our people_counter.py routine currently defaults to 0.80), the clip ends. If a second or third person joins the frame, the video should continue recording until they leave or become obscured. When new objects are detected, another clip will be saved with a new timestamp.

AWS Simple Notification Service (SNS):

This Amazon Web Services product is a simple way to set up a publish/subscribe architecture for forwarding messages to a phone number. Once the region is selected, a topic is added and given a name, such as “Banditland Person Detection” which prepends the message as shown in this screenshot from Ryan’s phone:



As many phone numbers as desired can become subscribers to this topic. Once the aws-cli is configured on the Xavier NX or Nano to authenticate with an access key/secret key (and/or session token), the python boto3 library can easily publish to this topic with this class utilized by our people_counter.py method:

```
class AwsNotifier:
   def __init__(self,
region='us-west-2',topicArn='arn:aws:sns:us-west-2:876612415673:xavier_securitycam'):
       self.region = region
       self.topic = topicArn
       # Create an SNS client
       self.sns = boto3.client('sns', region_name=self.region)
```

```
def publish(self, message):
    # Publish a simple message to the specified SNS topic
    response = self.sns.publish(TopicArn=self.topic, Message=message)
```

Results

We were able to achieve an efficient and functional home security system with the help of our Jetson devices and raspberry pi version 2 cameras! As we mentioned in our approach and implementation, we were able to maintain a decently high FPS in our output video files (and while streaming) at around 20-30 FPS. The video recordings were happening as we expected, i.e. after the camera detects at least one person. Videos can quickly take up hundreds of MegaBytes of space and this particular feature can trim it down significantly. In addition to this, the AWS messages also reach the phone promptly giving the number of people detected, the timestamp and also the confidence level. Therefore, if it's in the night, the user is out of town and he gets a message that there are 3 people with a confidence level of 90%, it is best to alert the cops (compared to Google nest where it is possible that the alert could be the shadow of a swaying branch). We believe that this can be a lot more effective than the Google Nest and also cheaper. Ryan is already in the process of replacing the Nest with our Project!

We would love to extend this further. Currently, our video footage sits on the local machine. These can be sent to AWS S3 buckets and hence be accessed from any device (what if you are far from home and want to analyze a footage). Another feature is for the system to recognize the home-owner and his close friends so that video need not be captured in such scenarios. We could even include more smart devices - the security system could recognize the homeowner, send a message to your Alexa speaker, and Alexa can play welcoming music once you enter your home!

Contributions & Conclusions

All in all, we enjoyed working on this project where we were able to build upon the skills we developed through the course. We also learned new techniques and got comfortable by working on a whole new development platform (Jetson) and can also scale our system by adding more features to what we currently have. The best part is that we can actually deploy this in our house, save expenses, and remain safe!

Both team members purchased all hardware and implemented all software so they could assist each other through any difficulties. Ryan purchased more advanced hardware only

because he plans to use this as an actual security system and can use the Xavier NX for other machine learning problems when not acting as a person detector (this could probably be offloaded to a Nano once performance is stabilized). The dual camera slots could also be utilized to stitch together an even wider scene at high quality (although not likely compatible with a wide-angle lens).

Ryan prepared the video and Shreyas wrote the majority of this report. We really enjoyed working together on all five projects this semester and found a great groove on this final opportunity, even working on opposite sides of the globe. It was an amazing course!