Low-light Environment Neural Surveillance (LENS)

Henry Gridley, Kevin Hines, Noah Lichtenstein,
John Nguyen, Michael Potter, and Jacob Walsh

Advisor: Bahram Shafai

Northeastern University

December 10, 2019

Contents

1 Abstract

For our capstone project, we have designed and implemented a system for criminal activity detection in low-light environments. Unlike CCTV, which is a reactive system, the LENS system can be used by law enforcement to get real time crime alerts. The system uses a low-light camera and a NVIDIA Jetson TX2 embedded GPU running a FlowNet2-CSS neural network (NN), a spatial ResNet-101 and temporal ResNet-101 neural network, and a support vector machine (SVM) to identify shootings, assaults, and thefts as they occur. Law enforcement officials then receive a notification alerting them of the crime and have the opportunity to view a video of the crime should they deem the notification credible. Citizens may download a public facing app which enables law enforcement to push alerts for credible crimes based on user proximity to a crime.

To use LENS, law enforcement would first install a camera and GPU on a dedicated pole or light post in the area that is to be surveyed such as a dark park or alley. The system uses a stacked inference model architecture to do real time activity detection and determine if a crime is occurring. The spatial neural network takes a single RGB image as input and outputs a logit prediction over the actions. The temporal network takes 10 stacked optical flow frames (optical flow encodes the horizontal and vertical motion between frames) as input and outputs a logit prediction over the actions. The output from each model is concatenated to create the input to the SVM, which outputs a final probability over actions. If the action is determined to be criminal, a notification is sent from the GPU to the cloud and then to the monitoring app for law enforcement to evaluate. The video is then be sent from the module to the cloud and from the cloud to the law enforcement's application.

The system was trained and tested with the LENS dataset created by the team on a Google Cloud virtual machine with a NVIDIA Tesla P100. The team acted out three types of crimes wearing various clothing and ran the data through the stacked model architecture. The confusion matrices showed that the various types of crimes were predicted with high precision and high recall. The F1 scores, which are the harmonic mean between precision and recall, were typically in the range of 80%-95%. Using the Cloud GPU, the system achieved a runtime of 20 FPS for FlowNet2.0, 50 FPS for spatial ResNet, and 45 FPS for temporal ResNet. However, when transferred to the embedded GPU, performance degraded to 2 FPS, 10 FPS, and 10 FPS respectively due to the reduced processing power. We believe that faster hardware or a less precise inference pipeline would increase real time performance.

2 Introduction

Intelligent surveillance cameras have become increasingly popular in both commercial and private applications. Traditionally, CCTV systems are the de facto monitoring system, but more recent products such as Ring home security

systems use IoT connected surveillance video feeds to provide homeowners an extra degree of security through remote monitoring. As the number of internet connected devices continues to increase and dominate the market, the growth of these products depend on their smart capabilities. Most camera applications require some amount of manual operation and have limited use in low-light environments. Our project implements action recognition algorithms to detect crime in low-light conditions, effectively removing the responsibility of civilians to be able to report a crime, and provides an expedited response time from government entities in urgent situations.

Figure 1: Code Blue station (not in use)

Other similar solutions include the Code Blue emergency call system, shown in Figure 1, which many universities have deployed on their campuses. These systems provide a direct communication line to first responders, however the call station depends on the user being able to reach and physically push the call button. Furthermore, the rate of reported crimes is dependant on the victims or bystanders which does not always happen. LENS does not rely on the victims or bystanders and is intended to greatly reduce this human error.

Though there exist algorithms for action recognition [1][2][3][4][5], many are not applied in real-time or low-light environments. The existing benchmark action recognition datasets such as HMDB-51 (Human Motion DataBase), UCF-101 (University of Central Florida), and Sports-1M contain primarily daytime videos. There is also no standard benchmark dataset for crime detection, however UCF released a UCF-Crime dataset for general anomaly detection and recognizing 13 crimes, including *arrest, arson, assault, burglary, explosion, fighting, robbery, shooting, stealing, shoplifting, and vandalism.* This dataset consists of indoor video feeds or camera views focuses on buildings, which is not an applicable backdrop for this project.

Low-light cameras traditionally consist of IR sensors or CMOS sensors, which are commonly used in home surveillance and military applications. Modern day

cameras have high resolution, dynamic range and video frame rate, which can all be manually controlled through software. This is critical for strict system requirements.

To allow for reliable and secure communication, the camera is WiFi/Ethernet enabled. This way, the project can scale to incorporate many cameras with standard IoT techniques. A network of cameras can integrate with AWS IoT, which could allow for decision making and analysis to occur off of the hardware platform and instead be made in the cloud. However, most of the computation is offloaded to the local GPU or processor to minimize lag time to detection and to minimize network calls to the cloud. This design facilitates modularity and scalability, as well as create a separation of roles, wherein the hardware platform is responsible for crime detection, leaving the cloud responsible for decision making and alerts by pushing information to connected applications or web monitoring services.

LENS leverages these technologies into a deployable and robust embedded platform that is modular and is scalable. Future features could include a moving camera control system, continuous suspect tracking (facial recognition), or crime detection augmented with audio feed. Epipolar geometry could be incorporated to mitigate occlusion, or for pooling camera feeds for 3D reconstruction which would provide more robust models. Lastly, as hardware continues to improve, cameras with higher frame rate, resolution and SNR, in addition to board compute power would enhance processing and model accuracy.

3 Analysis

Prior to designing our system, it was imperative that we research and analyze the problem space, as well as possible solutions. This section details this exploration, as well as some of the possible extensions that we did not implement.

3.1 Hardware

When considering embedded applications for computer vision and machine learning projects, it is ideal to have a board with high compute power and modular I/O. A key component is having a GPU to process video and encode/decode video for cloud integration. Additionally, buying camera with off the shelf drivers and SDK avoids having to develop custom drivers. Other aspects to consider are if there are on board WiFi or Bluetooth modules, external memory for saving video and storing action recognition model parameters. The board also should to be able to host either a modem or router for network connectivity. A MIPI CSI bus lane or USB interface is also required to connect to the low-light camera.

CSI cameras have an advantage performance in parameters such as FPS, resolution, and CPU usage whereas USB cameras are cheaper but are typically plug and play out of the box. USB cameras often come with readily available

drivers and their own SDK, whereas CSI cameras offer lower level control of the camera/sensor. One major advantage of CSI cameras is bandwidth as USB3.0 is limited to 5Gbps. However, this may be misleading since USB connections have an impact on the CPU overhead which could lower processing power, especially if there are other tasks in progress.

In terms of image quality, various accessories could be used to improve video quality such as a lens hood and a lens filter. If the camera is placed directly beneath a lamp for example, a lens hood would prevent harsh lighting from different directions hitting the front of the lens and producing unwanted flare in images. An intensifier filter can be used to target the light emitted by street lamps and significantly reduces yellow glow as the light passes through the filter which increases the image's contrast and sharpness. Compression was also considered, because although raw video capture provides more data, compressed video performs comparable to raw video in researched applications such as object detection and facial recognition [6].

3.2 Network Architecture

Traditional IoT architecture separates computing into two categories: edge computing and cloud computing. For our project the cameras and processor responsible for running our computer vision models are considered to be the edge devices. In designing their architecture, there are two approaches that were considered. The first was to have all devices make separate cellular connections for direct access to the cloud, while the second was to have all devices join a local wireless network, which would maintain a single cellular connection for the group.

The benefits of the former are that it decentralizes the system and reduces the amount of infrastructure needed for deployment. If one camera loses its outward connection, the other cameras are unaffected. However, this would substantially increase the cost to maintain many active connections. The latter approach pools many devices into a local wireless network, with one outward connection to the cloud. This would reduce costs, as it would require fewer sim cards and reduce data usage. However, it does require extra infrastructure by way of additional routers to establish a wireless network, and increases the impact of connection failures.

3.3 Cloud Services

Our project also needed to utilize cloud services to handle decision making and routing of all messages that come from the cameras. Using the information computed on the edge of the network, cloud services then used that information to update the information consumers, like the apps installed on a user's cellphones.

In designing this component of the project, we considered Amazon Web Services (AWS), Google Cloud, and Microsoft Azure. AWS provides many services such as IoT Core, which is a managed cloud service that allows for the easy

communication between edge computers, the cloud, and IoT devices. It can interconnect with the rest of Amazon's computing platform, which allows the use of server-less architectures for data processing. AWS IoT Core also provides secure authentication methods to ensure network security.

Additionally, AWS provides two IoT device development platforms that simplify the application development process: AWS Greengrass and Amazon FreeRTOS. Greengrass is an application that runs on edge computers which allows them to communicate seamlessly with the cloud. Similarly, FreeRTOS is a microcontroller framework for small embedded devices.

Google Cloud offers similar platform services with the Cloud IoT Core, providing easy to use options for securely connecting, managing, and sending data into the cloud. Google Cloud uses the pub/sub architecture, which makes analysis on the platform easy to implement. They also provide a Cloud IoT Edge application, which allows edge computers to communicate easily with the cloud, similar to AWS. Unique to Cloud IoT Edge is the ability to run machine learning models trained in the cloud directly on the edge computing devices, which is very useful for training machine learning models for use on edge devices.

Microsoft Azure offers the same platform services as its competitors, with the added benefit of being built on Azure's enterprise grade suite of cloud services. It can support billions of edge devices and facilitate quick connection of new IoT devices. Azure IoT Edge also allows the same offline computing and seamless communication integration as its competitors. One unique advantage of Azure is its "solution accelerators." These are web based templates that are pre-constructed for common IoT scenarios. They may be useful for facilitating rapid prototyping, although are likely not reliable as permanent solutions. They also provide simulation tools to help test IoT architectures before deployment at scale.

3.4 Computer Vision

Prior to 2014 action recognition consisted of laboriously hand-crafting feature representations for images which were predefined by experts, such as histogram of gradients (HOG), histogram of flow (HOF), motion boundary histogram (MBH) and improved dense trajectories (IDT). These feature representations were input to classical statistical models such as support vector machines (SVM), multi-class logistic regression, or random forests.

Then, in 2014 Karen Simonyan and Andrew Zisserman pivoted the action recognition research into training deep Convolutional Neural Networks (CNN) for action recognition in video with [1]. This paper became the benchmark for action recognition research. Here, the main contribution was a two-stream CNN architecture which incorporated a spatial and temporal network.

The spatial stream was trained on RGB frames sampled from videos, while the temporal stream was trained on multi-frame dense optical flow. The spatial stream and temporal stream predictions on actions were averaged, or concatenated as an input feature to a SVM for final action predictions. The architecture was trained and evaluated on the UCF-101 and HMDB-51 datasets, which are

standard benchmark datasets now for Computer Vision action recognition. New datasets such as Kinetics-400 dramatically improve action recognition results on UCF-101 and HMDB-51, and is starting to become the new benchmark dataset.

In 2016 the Computer Vision Lab of ETH Zurich in Switzerland devised a novel framework, Temporal Segment Network (TSN), for video-based action recognition built on Simonyan and Zisserman's work [2]. This research contributed to a series of good practices in initializing and training the two-stream CNN architecture for state-of-the-art results. Two-stream CNN Architectures drastically improved action recognition results on benchmark datasets, however there was a bottleneck in calculating the optical flow which prevented true "real-time" action detection.

Due to the optical flow bottleneck in real-time action recognition, two new approaches for real-time action recognition were developed [3][4]. Facebook AI research and Dartmouth college [3] developed the C3D architecture, which was a simple yet effective approach for spatio-temporal feature learning using 3-dimensional CNNs, where the temporal information between video frames would be implicitly calculated. The C3D architecture was able to perform action recognition at an frame rate of 313.9. Unfortunately, this caused action detection to degrade.

ETH and Tongji University [4] were able to circumvent the calculation of optical flow by extracting pre-calculated motion vectors from compressed video to achieve a frame rate of 390.7 without a severe degradation to action-detection results (compared to the action recognition benchmark [1]). As evident from the research, there is a trade-off between speed and accuracy. Determining how much latency is acceptable in crime detection, and how accurate the action recognition software needs to be is key in creating a robust and useful system. If a more accurate, but slower action recognition software is desired, a simple Two-Stream CNN Architecture using optical-flow could be used. If a high frame rate, but less accurate action recognition software is desired, a 3D CNN architecture similar approach to [4] should be taken.

The previous research was developed in the time from 2014-2016, whereas new state-of-the-art approaches have been developed [5][7]. Since the two-stream CNN architecture has showed promising action recognition results, researchers developed two-stream Inflated 3D Convnet (I3D) architectures to take advantage of a spatial stream and a temporal stream but with 3D CNNs. The benefit here is that the 3D CNNs can implicitly calculate motion from stacked RGB images and change in movement speed from stacked optical flow images. I3D, and networks similar, have created an explosion in the number of network parameters, which leads to overfitting on small datasets such as UCF-101. Therefore, large datasets such as Kinetics-400 are used to train I3D, but the downside is that these datasets comprise several hundred gigabytes in memory. The next steps in action recognition research is to leverage spatio-temporal Features with 3D residual networks, which has shown promise.

4 Approach

Figure 2 shows the LENS system block diagram, which is detailed further in this section.

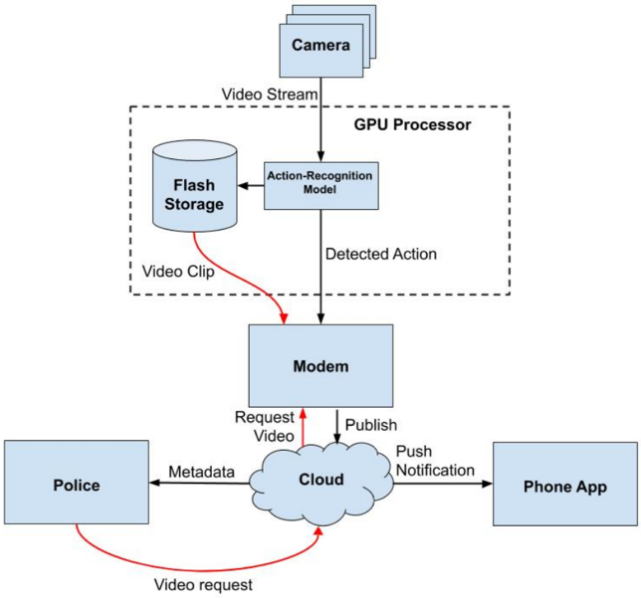

Figure 2: LENS system architecture

4.1 Hardware

To select hardware, an decision matrix was created, shown in Appendix A. The categories were cost, performance and peripherals which would reduce the amount of additional hardware needed. Out of the three boards, the Jetson TX2 and PICO-Pi earned the highest scores and merited further assessment, shown in Appendix B.

For selecting the development board, the NVIDIA Jetson TX2 was preferred over the PICO-PI. With the NVIDIA student discount and readily available developer kit, there was no reason to sacrifice computer power for cost. The Jetson has the highest performing interfaces for computer vision and machine learning applications, and the on-board graphics card makes its easy to leverage performance gains for video processing as opposed to relying on a CPU. In addition, the TX2 has higher bandwidth peripherals for storing and processing video which is crucial for improving the speed and accuracy of our algorithms and models. The NVIDIA JetPack SDK has a mature and well documented API which will helped speed up image installs, manage libraries and reduced the amount of time dedicated to system integration and debugging.

Choosing a camera had a higher number of trade-offs to be considered as demonstrated in the decision matrix. The key component we evaluated was low-light performance, which breaks down into many aspects. Low-light cameras can function in dark environments through IR, specialized CMOS sensors, or a combination of both. However, the most important factors of video acquisition are the dynamic range and signal to noise ratio. Dynamic range illustrates the brightest and darkest areas of a frame. In other words, sufficient contrast is crucial for computer vision models to detect object outlines. Furthermore, a high signal to noise ratio helps to isolate noise from events and stray unwanted signals from the environment. Challenging lighting conditions, sensor exposure time limitations and lens quality play the largest role in determining SNR and DR.

The next consideration was whether or not a thermal imaging or night vision camera would provide a better solution to the issue of a low-light environment. While thermal vision cameras are preferred when there is no light, the scope of our project is to provide crime detection in low-light areas, thus thermal optics would be unnecessary especially since there is a much lower SNR and dynamic range which would prove challenging for our datasets.

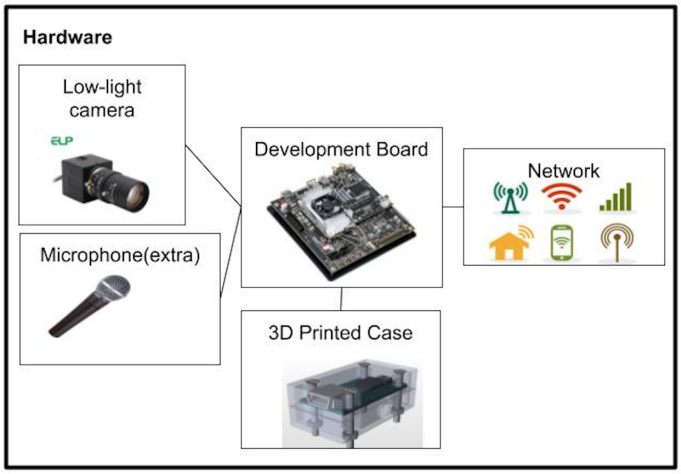

Figure 3: LENS hardware platform

Lastly, the requirements of our models are 25fps in MJPEG. USB was chosen as the interface to our board as the USB Video Class (UVC) and USB on-the-go (OTG) are incorporated into the devices firmware which allows for plug and play access to the camera and its controls/resources. Being able to control the camera's video acquisition is also a huge factor as remaining agile and being able to tweak parameters will yield higher accuracy.

Ultimately the ELP 2MP Sony IMX322 USB Camera With 5-50mm Varifocal Lens was chosen for its low cost, favorable low-light performance, efficient integration, and user control. The decision matrix and further camera compari-

son can been seen in Appendix C and D respectively. Figure 3 shows a high-level overview of the hardware platform, including possible extension peripherals that could be added to the system.

4.2 Network Architecture

The three main network components in the project were the edge computers, cloud, and the clients, all shown in Figure 4. The edge computers used Amazon FreeRTOS and AWS IoT Greengrass to process the images, perform inference using the computer vision models, and communicate with the cloud. The edge computers were WiFi enabled and communicate to a local router that contains the local network of edge computers. In the cloud, AWS IoT Core and Lambda functions process incoming messages and relay them to the clients. The clients were law enforcement using a mobile app that was be authenticated with the cloud to consume and send messages and requests. These messages were handled with REST API calls to populate the mobile app and notify law enforcement.

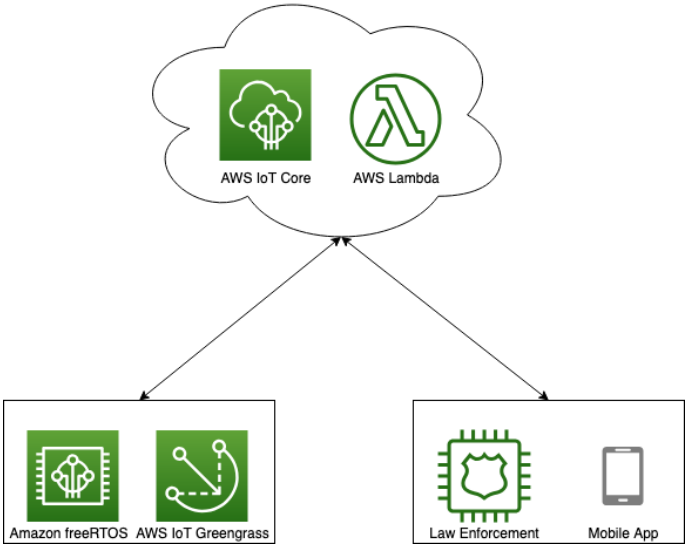

Figure 4: LENS IoT architecture

4.3 Cloud Services

In order for the cameras to communicate with an app or law enforcement, a cloud component was used to handle requests. The camera was be WiFi enabled in order to scale to multiple cameras in the same network. If the computer vision model detects a criminal activity, the corresponding camera ID, GPS location and clip of the incident were be sent to the cloud. From there, the message was relayed to law enforcement through the mobile app.

Ultimately, the AWS IoT suite was best cloud service for our needs. On the edge machine, AWS IoT Greengrass and Amazon FreeRTOS acted as the local OS and send messages to the cloud. In the cloud, AWS IoT Gateway was used to process requests from edge machines and relay them appropriately. In this model, AWS IoT Core could interpret messages from the camera and the mobile app. AWS IoT Core used AWS Lambda, AWS S3, and other services to handle incoming and outgoing messages. Much of the computation for the inference model and triggers were be left to the GPU or processor on the camera, so the cloud simply parsed the results and relayed messages.

4.4 User Interface

The user interface for this project is a mobile app that allows law enforcement officers to monitor and control the LENS system in real time. Notifications appear on a users computer that show what kind of crime is occurring and where the crime is taking place based off of when the cameras are located. An officer monitoring the app can then notify other officers in the area where the crime is taking place through a mobile notification. An additional option will allow the officer to notify civilians through a separate app where they can opt into notifications based on their chosen location. Figure 5 shows a mock-up of the app landing screen which incorporates this functionality.

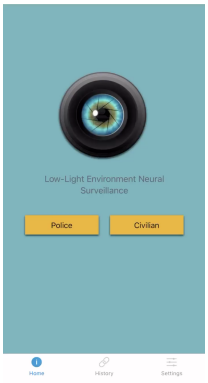

Figure 5: App landing screen

The app was created using React Native and node.js for easy deployment across iOS, Android, and the web as well as simple integration with AWS.

4.5 Computer Vision

The computer vision algorithm used consisted of two parts: dense optical flow calculation and action recognition. For the dense optical flow calculation we used FlowNet2.0, a deep learning approach developed at the University of Freiburg in Germany [8], and for the action recognition we used a two stream approach [1]. The overall network architecture can be seen in Figure 6.

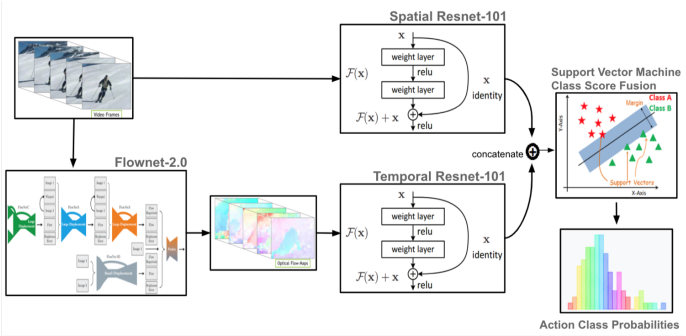

Figure 6: Computer vision network architecture

4.5.1 Data

Despite the many computer vision datasets available, less than 2% of them feature low-light data. On top of that, we require data specific to our domain, meaning videos of crimes occurring. Due to these factors, we needed to record our own data and employ transfer learning to fine-tune our models on this data.

To collect the data, we recorded actions continuously in one minute intervals and then split up the resulting videos into 4-7 second clips which were used for training. This allowed us to get approximately 10 times as much data without having to record each video individually.

While we recognize that this method of data collection caused our models to overfit, thus causing it to only perform well when the subjects resemble us, it still allowed us to validate our pipeline and show that further improvement primarily requires more diverse and robust data.

4.5.2 Optical Flow

In order to perform action recognition using a two-stream approach as detailed in [1], the optical flow must be calculated for the temporal stream. In the aforementioned papers, the optical flow calculation is determined to be the bottleneck operation. In order to prevent this, we have chosen FlowNet2.0, which on top of being state-of-the-art, can be configured to perform at a higher frame rate at the cost of reduced accuracy. This allowed us to find the appropriate balance between speed and performance.

FlowNet2.0 architecture, shown in Figure 7, consists of layered networks, each of which calculates or refines the optical flow. The first three layers calculate large displacement optical flow, which translates to large movements and requires less accuracy, while the last layer calculates small displacement. The small displacement layer was added in FlowNet2.0 to improve the optical flow related to fine movements and details.

This is very relevant to our project as the subjects may only be a few pixels, or the movements we are interested may not be very large. At the end of the

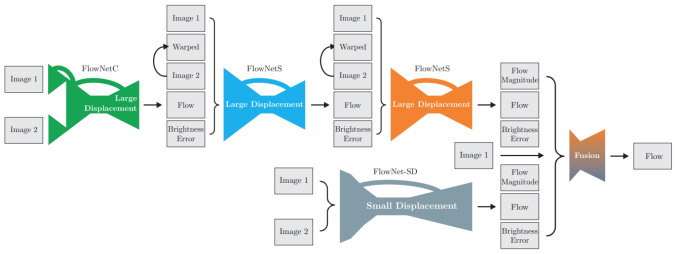

Figure 7: FlowNet2.0 network architecture[8]

network, the last large displacement layer and the single small displacement layer are fused together to produce the final result. An example of the results achieved from the FlowNet 2.0 architecture can be seen in Figure 8.

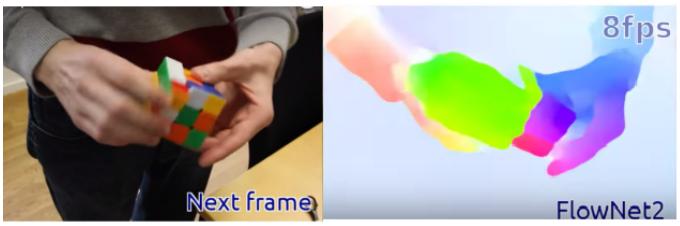

Figure 8: Example of FlowNet2.0 output achieved at 8 FPS

4.5.3 Action Recognition

The action recognition approach follows [1][2], where two CNN architectures are fused for final prediction results (Figure 14). A two-stream 2D CNN architecture was chosen over 3D CNN architectures because the two-stream CNN explicitly calculates the optical flow in the temporal network stream, whereas the 3D CNN implicitly calculates the optical flow which leads to poor motion estimates. State-of-the-art research methods were employed, as there are insufficient resources during the capstone period for trial and error, which would have left little room for error. The interpretation of motion in the models is what distinguishes the action recognition model from a object detection model. This is because object detection models only observe a spatial RGB image, whereas the action recognition model observes an RGB image and motion estimated from future RGB frames. The fusion for the final prediction was achieved using an SVM trained on the concatenated predictions from both the spatial and temporal networks. The spatial CNN parameters were optimized by stochastic gradient descent with momentum set to 0.9. The initial learning rate was set to $5e^{-4}$ and decayed over multiple epochs using a Plateau learning rate scheduler. The temporal CNN was optimized in a similar manner.

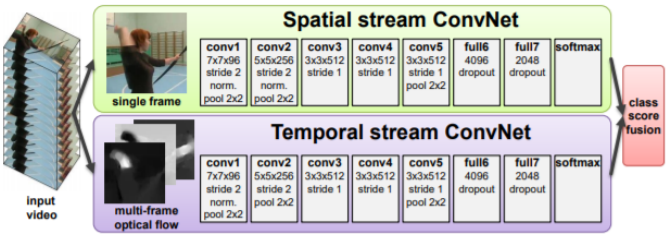

Figure 9: Example of two-stream CNN architecture [1]

The temporal and spatial streams were initially trained on the UCF101 dataset, then fine-tuned using transfer learning on our criminal action recognition dataset. The UCF101 dataset was divided into three train-test splits, and standard evaluation of the network on the UCF101 dictated that the model be trained on the first train-test split, and the accuracy averaged over all three test splits.

Several data augmentation tricks such as random cropping, resizing, channel normalization, and scale/aspect jittering were employed during training and testing to prevent overfitting. Other regularization techniques were also used such as batch normalization and dropout on the fully connected layers of the CNN architectures.

The original spatial and temporal stream network architectures were replaced by a novel deep neural network that alleviates vanishing/exploding gradients with residual connections called ResNet [9], where the last fully connected layer (FC) as replaced with a new FC layer that outputs a vector with dimension equal to the number of action classes represented in a dataset. The temporal and spatial stream base ResNet network was initialized with weights pre-trained on the ImageNet dataset. The first convolutional layer weights for the temporal stream was initialized with a special training trick called cross-modality weight initialization [2].

All networks were trained on Google Cloud, as large memory and computation capability are required to train deep neural networks. A virtual machines was deployed using the Deep Learning VM image in the Google Cloud Marketplace on the Google Cloud Compute Engine, where the PyTorch, FastAI, NVIDIA configuration settings were applied. Due to the training datasets taking over 100 Gigabytes (GB) in memory, a 500 GB zonal persistent disk was added and mounted to the VMs for dataset storage. The GPUs used was an NVIDIA Tesla P100.

5 Results

5.1 Action Recognition

The spatial stream and temporal stream were pre-trained on the UCF101 dataset, in an attempt to duplicate the experiments from Zisserman and Simonyan. After replicating/improving the author's accuracy on the train01-test01 split of the UCF101 dataset, the spatial and temporal streams were fine-tuned on the LENS dataset.

For the spatial stream of the action recognition network, we were able to achieve 71% accuracy after training it on our data. However, the spatial stream had a hard time differentiating between the assault and theft classes, most likely due to their similarities, especially without the context that motion provides. By removing one of these classes or using a binary classification of "crime" or "no crime" instead it might be possible to improve these results even further. The training curve for the spatial network is shown in Figure 10.

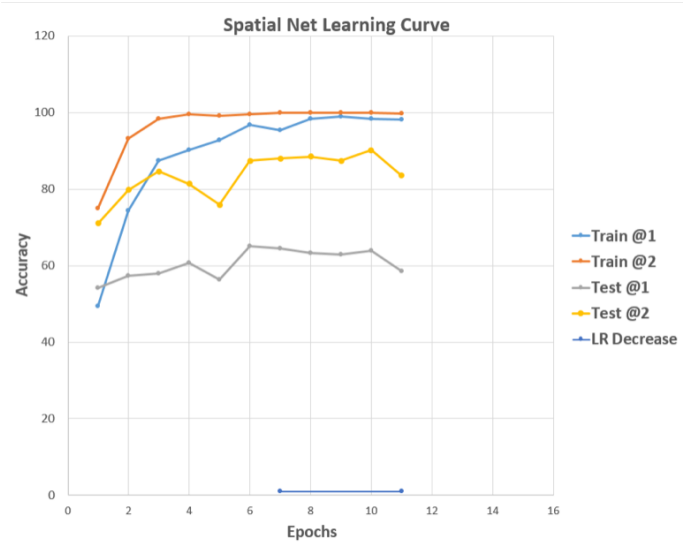

Figure 10: Training curve for spatial network

The motion stream on the other hand achieved around 87% accuracy, which is significantly higher than the accuracy in the original research paper. This shows that motion played a key part in action understanding and made it clear why it was needed in the pipeline. It did however occasionally mix up the assault and shooting classes, again possibly due to the similarities the actions can share and the fact that the temporal network does have perception of a gun. The training curve for the motion network is shown in Figure 11.

Finally, we used a support vector machine to combine the predictions from each of the two streams during the action recognition step. The purpose of this

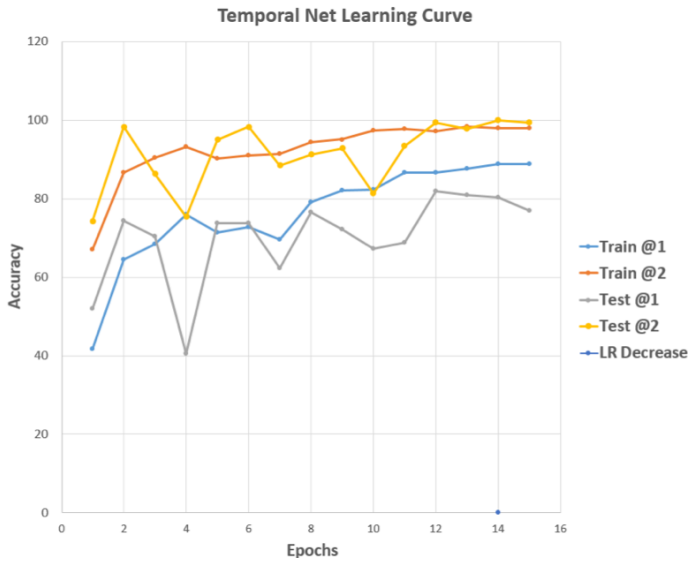

Figure 11: Training curve for the temporal network

step was to apply a nonlinear function to the predictions in the hope of making a better decision than just adding them together. In theory, this should help eliminate some of the deficiencies from each model and produce a better result in the end. There was no benchmark for this step, but we were able to achieve an accuracy of 71% for the classifier. While this does not seem any better than either stream individually, the SVM helped prevent the network from failing quite as badly on any one action. Figure 12 shows the confusion matrices for each step of the action recognition component.

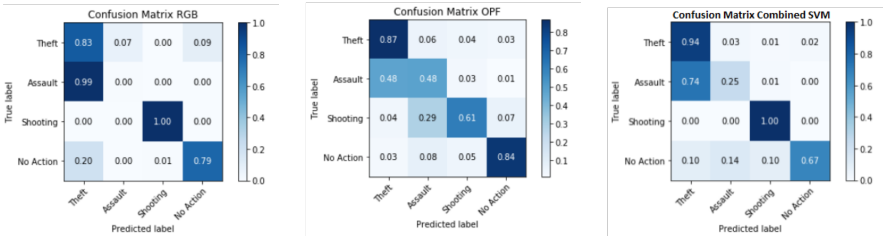

Figure 12: From left to right, confusion matrices for spatial stream, temporal stream, and combined SVM

5.2 Optical Flow

There were several issues when pre-computing FlowNet2.0-CSS optical flow on the LENS dataset for training, validation, and testing partitions. These issues

included image compression, normalizing optical flow, and image resolution of the frames.

To pre-compute and store optical flow for training neural networks, the optical flow images were saved as JPEG images, which is a lossy compression technique that inherently reduces the pixel intensity variance, by removing outliers and/or the tails of a Gaussian distribution. This phenomenon was observed when the histogram of pixel intensities of the pre-computed optical (which was saved in JPEG format) were compared to the histogram of pixel intensities of the live-time optical flow (before JPEG compression). For future analysis, the pre-computed optical flow should be saved as a NumPy file (.npz), which completely preserves the pixel intensities. However, a trade-off between memory storage and quality occurs when no compression technique is applied. In our case, memory storage issues were inconsequential because all computation was hosted on a Google Cloud Compute virtual machine.

For motion between frames to be accurately computed, a large image resolution was required due to the brightness constancy constraint, which assumes that small image patches share the same dynamics and do not change intensity. When the image resolution is small, the brightness constancy constraint fails and produces poor optical flow as the small image patches inherently occupy a higher percentage of the image, and fine-grained movement is not captured. Unfortunately, there was a trade-off between optical flow accuracy and latency. Higher image resolution meant higher accuracy optical flow, but also incurred higher latency. An image resolution of 426x640 allowed for sufficient optical flow accuracy, while retaining a reasonable latency. Ideally a higher image resolution is desired, because the lower resolution degrades the temporal network's performance. This was further made worse by the fact that we chose to use the FlowNet2-CSS network architecture instead of the full FlowNet2 architecture to increase the frame rate. This is a pared down version of the original that makes sacrifices in accuracy to improve the frame rate. Figure 13 shows an example of optical flow from each network architecture.

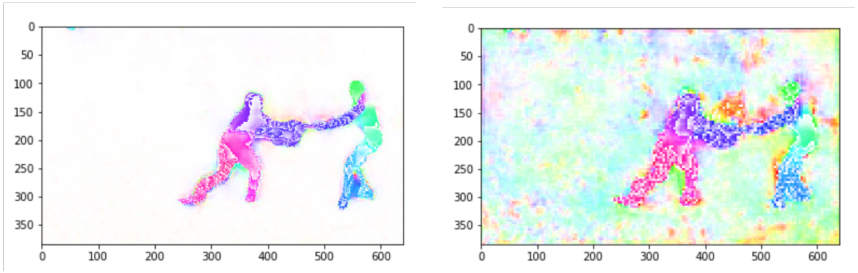

Figure 13: Optical flow output from FlowNet2 (left) and FlowNet2-CSS (right)

One issue we initially ran into when pre-computing the optical flow was not normalizing the pixel intensities in the range of 0-255. Compressing images with the JPEG format required all pixel intensities to be in the range of 0-255, and when this constraint was not satisfied the pixel values were clipped to satisfy

the constraint. Therefore, all leftward horizontal and downward vertical motion between frames was neglected, as leftward horizontal and downward vertical motion would have negative pixel values. Saving optical flow in JPEG format effectively removed half of the motion information computed in optical flow. After discovering this, we normalized the pixel values of optical flow to the range 0-255 by adding a bias of 128 to optical flow before saving as a JPEG, restoring the leftward horizontal and downward vertical between frames.

In terms of frame rate, we were able to achieve approximately 10 FPS running on the NVIDIA Tesla P100 in the cloud. This was further improved by skipping frames, especially since the motion captured between two sequential frames when recording at 30 FPS is largely redundant. As many as three or four frames could be skipped before performance started to suffer, which would amount to a 3-4x increase in speed.

On the NVIDIA Jetson TX2 we were able to achieve approximately 2 FPS due to the decreased processing power, however we were able to improve upon this in a few ways. The first was by skipping frames as mentioned previously. We also managed to stream from the Jetson to the cloud and perform the inference on the cloud. While this did not run as fast as native video on the cloud, it still improved the inference time and would be worth developing further. Finally, because the bottleneck was primarily the optical flow calculation, we were able to achieve a very high frame rate on both the Jetson and the cloud by only running the spatial stream of the action recognition network. This eliminated the optical flow calculation completely and reduced the inference to RGB frames only, which could be done very quickly. Unfortunately, this also meant that accuracy suffered since the inference did not incorporate motion.

Ultimately, our pipeline performed about as well as we had hoped. While we did have to make some trade-offs between speed and accuracy, access to better hardware could have allowed us avoid some of those sacrifices. The biggest bottleneck with regards to accuracy was the data, which was not robust enough to allow us to generalize as much as we wanted it to. Because the subjects of our data were limited to our group members and a few friends, the model overfit pretty heavily. However, we expected this from the beginning and we were still able to show that the pipeline works even with our limited data. An example of the pipeline inference on a crime, occurring is shown shown in Figure 11.

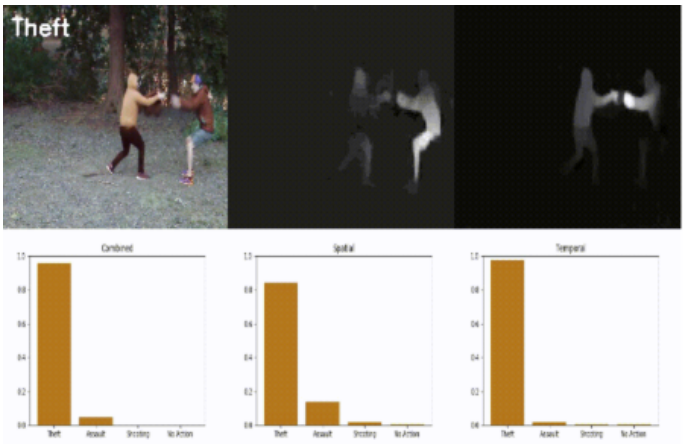

Figure 14: Clockwise from top left: RGB frames, optical flow (horizontal), optical flow (vertical), spatial predictions, temporal predictions, and combined predictions

6 Conclusion

This report details how our group designed a complete and modular system that could provide an effective tool for law enforcement to identify criminal activity in low-light areas. Not only does our solution allow police to be notified of potential criminal activity detected by our camera and action recognition system, it does so with a high degree of accuracy. Our system has the possibility to make many areas safer and allow peace of mind for people traveling through an unsafe areas after dark. Additionally, it provides the opportunity for law enforcement to direct resources where they will be most effective, saving time and money while still maintaining the most effective service possible.

Appendix A Compute Board Decision Matrix

Development Board	Cost (20%)	Performance (50%)	Peripherals (30%)	Total
NVIDIA Jetson TX2	1	3	3	2.6
Raspberry Pi 3	3	1.5	1.5	1.8
PICO-Pi IMX7	2	1.5	1.5	1.6

Appendix B Compute Board Comparison

Board	NVIDIA Jetson TX2	PICO-Pi
CPU	ARM Cortex-A57 (quad-core) @ 2GHz + NVIDIA Denver2 (dual-core) @ 2GHz CPU	ARM Cortex-A7 + M4 @ 1GHz NXP i.MX7 Dual
GPU	256-core Pascal @ 1300MHz GPU	SoC Integrated Graphics Engine
RAM	8GB 128-bit LPDDR4 @ 1866Mhz, 59.7 GB/s RAM	512MB DDR3L
Memory	32GB eMMC 5.1 Storage	4GB eMMC
Video	4Kp60, (3x) 4Kp30, (8x) 1080p30 Encoder, (2x) 4Kp60 Decoder - Video	-
Camera	12 lanes MIPI CSI-2, 2.5 Gb/sec per lane, 1400 megapixels/sec ISP Camera Port	2 lanes MIPI CSI
Display	2x HDMI 2.0 / DP 1.2 / eDP 1.2, 2x MIPI DSI	Custom external display (LCD)
Network	802.11a/b/g/n/ac 22 867Mbps, Bluetooth 4.1	1x Atheros AR8035 Gigabit LAN, 1x Broadcom BCM4339 BT 4.1 (BR+EDR+BLE)
Interfaces	10/100/1000 BASE-T Ethernet, USB3.0, USB2.0, Gen 2 PCI, CAN, UART, I2C, I2S, GPIO	1x Atheros AR8035 Gigabit LAN, USB2.0, USB2.0 C, I2C, I2S, PWM, SPI, UART, GPIO
SDK	Jetpack 4.2.1 SDK	PI SDK
Price	$299	$119

Appendix C Camera Decision Matrix

Camera	Cost (20%)	Low-light (40%)	Speed (10%)	Integration (30%)	Total
ELP 2MP Sony IMX322 Low Illumination 5-50mm	4	3	1.5	3.5	3.2
Hyperyon 2MP SONY STARVIS IMX290 Ultra Low-light USB camera	1	3	3.5	3.5	2.8
e-CAM21_CUTX2 2MP Sony Ultra-low light Jetson TX2/TX1 Camera	3	3	3.5	1.5	2.6
Logitech Brio 4K	2	1	1.5	1.5	1.4

Appendix D Camera Comparison

Camera	**ELP 2MP Sony IMX322 Low Illumination H.264 USB Camera With 5-50mm Varifocal Lens, Support Audio**	**Hyperyon 2MP SONY STARVIS IMX290 Ultra Low-light USB camera**
Resolution	1920x1080 @ 30fps & 1280x720 @ 60fps	1920x1080 @ 60fps (H264)
Sensor	1/2.9" SONY IMX322 CMOS SENSOR	1/2.8" SONY IMX290 CMOS Sensor
Shutter	Electronic rolling shutter/Frame exposure	Electronic rolling shutter/Frame exposure
Lens	5-50mm zoom lens (2.8-12 mm optional)	Fixed Focus (M12 lens)
SNR	42dB	40dB
Dynamic Range	86dB	WDR
Operating Environment	0.01 lux	0 lux
Codec	H264, MJPEG, YUV2	H264, MJPEG, YUV2
Parameter Control	Manual	Manual
Drivers	UVC	UVC
Operating System	Windows, Linux with UVC (linux-2.6.26 or later), MAC-OS X 10.4.8 or later, Wince with UVC, Android 4.0 or higher with UVC	Windows 8.1/10 and Linux (Ubuntu 14.04, 16.04 & 18.04)
Price	$71.50	$249

References

[1] Karen Simonyan and Andrew Zisserman. "Two-Stream Convolutional Networks for Action Recognition in Videos". In: *CoRR* abs/1406.2199 (2014). arXiv: 1406.2199. URL: http://arxiv.org/abs/1406.2199.

[2] Limin Wang et al. "Temporal Segment Networks: Towards Good Practices for Deep Action Recognition". In: *CoRR* abs/1608.00859 (2016). arXiv: 1608.00859. URL: http://arxiv.org/abs/1608.00859.

[3] Du Tran et al. "C3D: Generic Features for Video Analysis". In: *CoRR* abs/1412.0767 (2014). arXiv: 1412.0767. URL: http://arxiv.org/abs/1412.0767.

[4] Bowen Zhang et al. "Real-time Action Recognition with Enhanced Motion Vector CNNs". In: *CoRR* abs/1604.07669 (2016). arXiv: 1604.07669. URL: http://arxiv.org/abs/1604.07669.

[5] João Carreira and Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". In: *CoRR* abs/1705.07750 (2017). arXiv: 1705.07750. URL: http://arxiv.org/abs/1705.07750.

[6] Brendan Klare and Mark Burge. "Assessment of H.264 Video Compression on Automated Face Recognition Performance in Surveillance and Mobile Video Scenarios". In: *Proceedings of SPIE - The International Society for Optical Engineering* 7667 (Apr. 2010). DOI: 10.1117/12.851349.

[7] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. "Learning Spatio-Temporal Features with 3D Residual Networks for Action Recognition". In: *CoRR* abs/1708.07632 (2017). arXiv: 1708.07632. URL: http://arxiv.org/abs/1708.07632.

[8] Eddy Ilg et al. "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks". In: *CoRR* abs/1612.01925 (2016). arXiv: 1612.01925. URL: http://arxiv.org/abs/1612.01925.

[9] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.