Final Project Proposal: Filtering

Adam Beckwith, Aser Atawya, Jack Terwilliger, Julian Sanders

May 5, 2023

1 Introduction

1.1 The Problem

The applications for Machine Learning Computer vision is growing everyday. As such, many are pushing for real time (or close to it) video analysis for things like traffic cameras and people counting.

However, many real time advanced vision analytic algorithms do not maximize the output capable by the available compute resources in loud services resulting in sub-standard real time analysis. This problem stems from the algorithms themselves because "they require high computing resources and high-quality video input, but the (wireless) network connections between visual sensors (cameras) and the cloud/edge servers do not always provide sufficient and stable bandwidth to stream high-fidelity video data in real time." One solution to low gpu resources is to limit the number of frames we process. By filtering insignificant frames and not processing them, we can reduce work. Called *frame filtering*, this method sacrifices accuracy for computation time.

In this paper, we explore various frame filtering solutions which will identify and select key frames to send to the cloud for the analysis. We will analyze the results and determine the optimal composition of frame filtering techniques to optimize bandwidth and compute.

1.2 Background

The strength of Machine Learning methods is always a question of resources. Making progress on frame filtering increases resources and expands the functionality of these computer vision algorithms in the future. The biggest hurdle for Computer Vision will always be real time analysis. If we completely solved this Frame Filtering problem, a computer vision pipeline could then process frames on the cloud in real time without loss of significant information.

Frame filtering has applications in monitoring traffic footage, and security cameras (anywhere the camera is pointed at a single spit). Traffic footage analysis needs to be in (functionally) real-time, because traffic footage is always recording, therefore any delay leads to falling behind. While Security cameras

want to notify of intruders as quickly as possible. Therefore, any analysis time decrease is important to pursue.

1.3 Challenges

There are three main challenges we faced when working on this project. Firstly, we had to account for the speed of our Frame Filtering algorithm would have to function. We were limited to $O(n)$ (n is a number of pixels) since anything large could not be reasonably processed in real time. Even without machine learning, many solutions reached our computation limit. This is a problem for other solutions as well, you can read more about that in section 5.

Another problem is video size. Videos are getting larger and larger which only increases our computation rate. Even for traffic footage, the quality of cameras used is only increasing, so we have to adapt to high fidelity and large footage.

The last challenge is that each possible frame filtering algorithm can be optimized for different thresholds on different videos. We want to measure each one on their own merits, which means testing a variety of thresholds, but even then the result may be different across thresholds. It doesn't help that each threshold may be different orders of magnitude.

2 Design and Implementation

Our approach examined three different frame filtering techniques, structural similarity index, mean squared error, and absolute error, and for each of these methods we also implemented a Gaussian Blur, to create a total of 6 different techniques. After we implemented each frame filtering technique, we ran the model against two different intersection traffic camera footage. Next we will explain each frame filtering technique and how we implemented it.

2.1 Structural Similarity Index

The Structural Similarity Index (SSIM) is a widely used metric in image processing and computer vision that quantitatively measures the similarity between two image frames. It takes into account both the structural information and perceived changes in luminance, contrast, and texture. The SSIM index ranges from -1 to 1, where a value of 1 indicates a perfect match between the frames. To compare image frames using SSIM, you first compute the mean, standard deviation, and covariance of the pixel intensities in local patches of the images. Then, these values are used to calculate the SSIM index, which represents the degree of similarity between the frames. Higher SSIM values indicate greater similarity, while lower values indicate more dissimilarity. By analyzing the SSIM index, one can assess the quality or similarity of image frames in various applications such as video processing, image compression, and image enhancement. The SSIM index technique is primarily used for video processing, but because it

conceivably has the function to assert frame similarities it posed an interesting question if it could accurately filter frames.

2.2 Mean Squared Error

Mean Squared Error (MSE) is a commonly used metric for quantifying the similarity between image frames. It measures the average squared difference between corresponding pixel intensities in two images. The MSE is computed by taking the sum of the squared differences between each pair of corresponding pixels, and then dividing it by the total number of pixels in the images. A lower MSE value indicates a higher similarity between the frames, with a value of 0 representing a perfect match. However, MSE has limitations when it comes to human perception of image quality, as it tends to prioritize fine details over global structural similarities. Therefore, it is often used in conjunction with other metrics or in specific applications where pixel-level differences are more important, such as image de-noising or image restoration.

2.3 Absolute Error

Absolute Error (AE) is a metric commonly used to evaluate the similarity between image frames. It measures the absolute difference between corresponding pixel intensities in two images. The AE is computed by summing up the absolute differences between each pair of corresponding pixels. Unlike Mean Squared Error (MSE), which takes squared differences into account, AE focuses solely on the magnitude of the differences without considering their sign. A lower AE value indicates a higher similarity between the frames, with a value of 0 indicating a perfect match. AE is often used as a simple and intuitive metric for image comparison, especially when the absolute magnitude of the differences is more relevant than the specific details.

2.4 Gaussian Blur

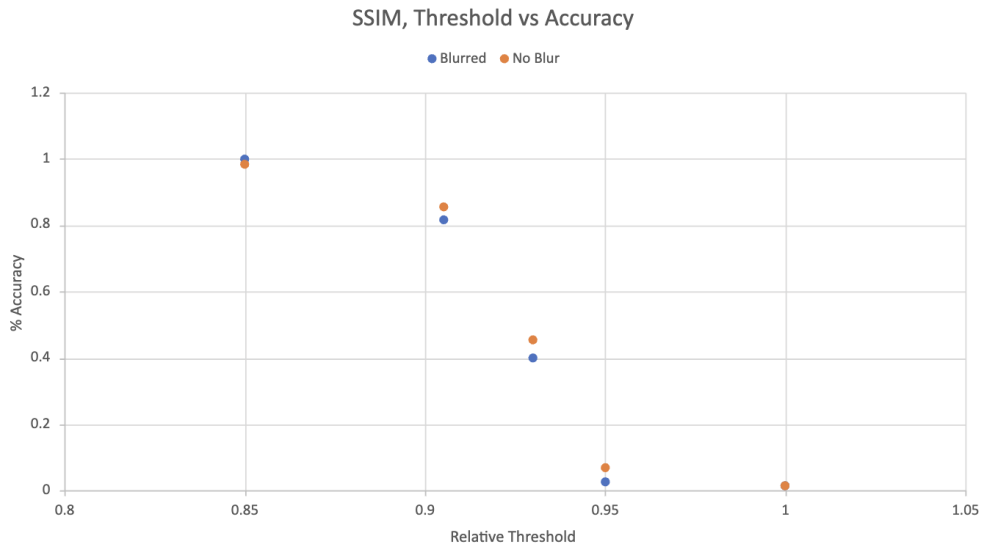
Gaussian blur is a commonly used image to reduce noise and smooth out the details of an image. This involves transforming the image with a two-dimensional bell shaped function. A Gaussian kernel assigns a weight to each pixel based on its proximity to the central pixel. The closer the pixel is to the center, the greater the weight. By spinning an image with a Gaussian kernel, the pixel values are fused, creating a blurry effect. Gaussian blurring can be implemented with different energy levels, controlled by the size of the Gaussian kernel. Gaussian blurring can be used to reduce repetitive noise and small changes in pixel values when determining image frame similarity, making images more robust to small contrasts. This enables configuration information and global similarity compares well and reliably, rather than focusing on fine detail or pixel level anomalies.

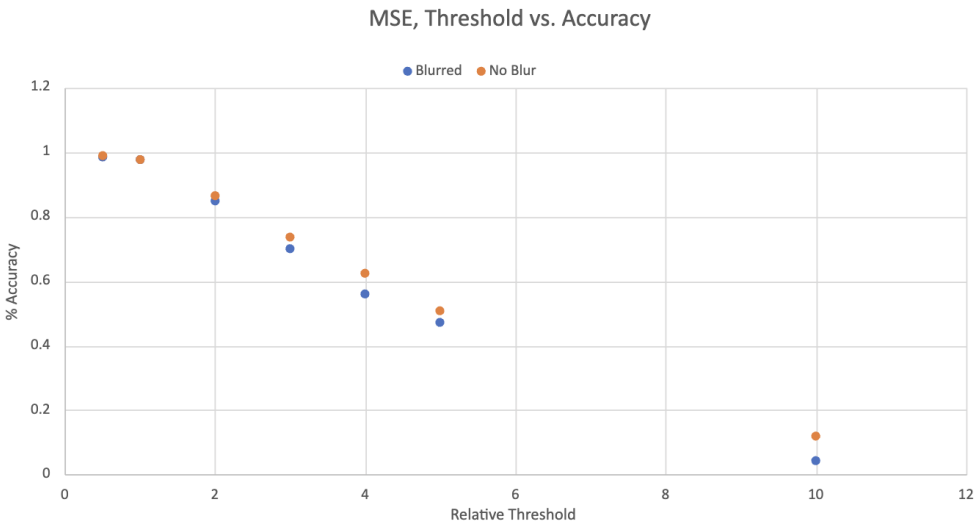
3 Evaluation

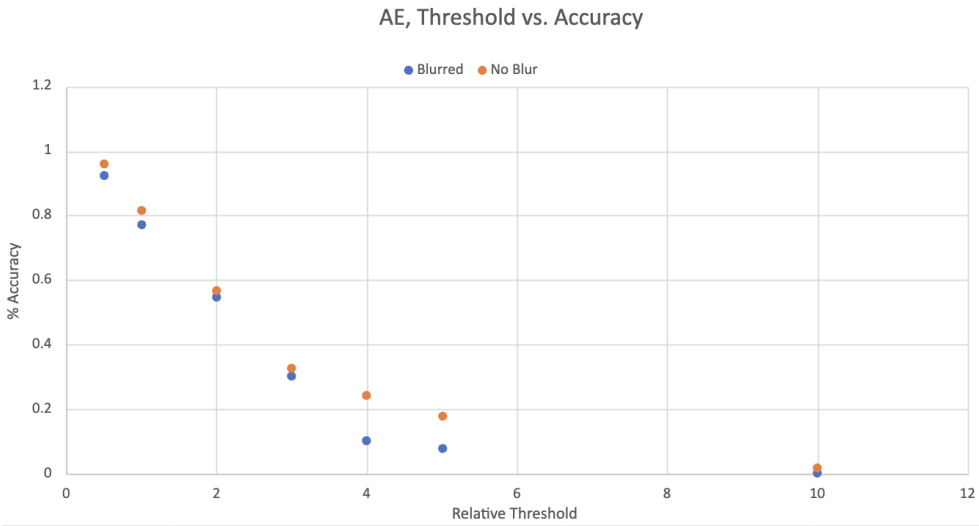
We performed two forms of evaluation: firstly, we chose a set of arbitrary thresholds for each algorithm (SSIM, AE, MSE). The values of the thresholds themselves was not important; rather, we chose a range of values across the spectrum so that our output would be well characterized (ranging from 0% filtered out and 100% accuracy to 99.994% - all but one frame - being filtered out yielding an accuracy of 1.625%). Choosing these thresholds proved surprisingly challenging for SSIM in particular, since the output varied widely with only marginal adjustments to the threshold. We then compared the accuracy of within these algorithms by comparing how they performed with and without a Gaussian blur at the same relative thresholds. The second evaluation we performed was by comparing the curve trajectory of the three algorithms by plotting the % filtered out vs % accuracy for each algorithm, irrespective of the algorithms' relative thresholds.

3.1 Blur vs. No Blur

The first evaluation we performed was by comparing the accuracy of each algorithm when applied on frames with a Gaussian blur and then on frames without a Gaussian blur. This comparison was done at the same relative threshold within each algorithm so as to get an "apples-to-apples" comparison.





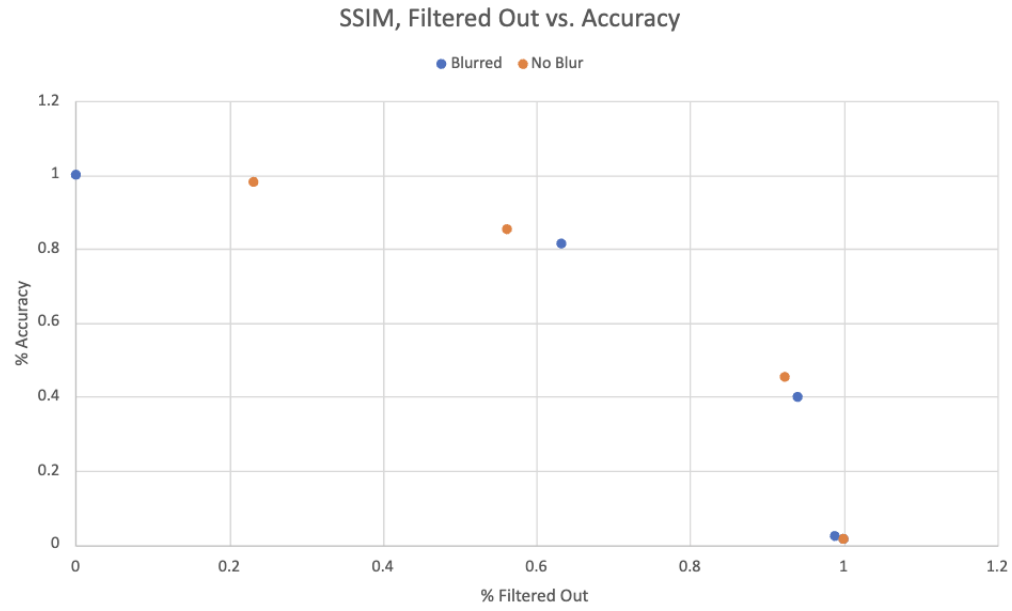


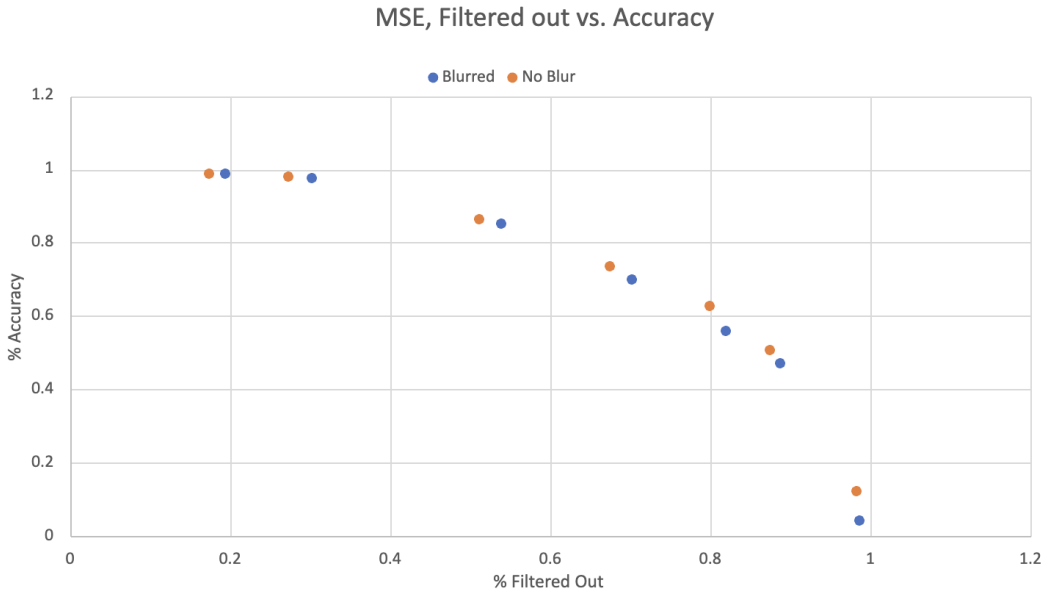
From this data, it's clear that applying a Gaussian blur decreases the model's accuracy when compared to the same filtering method applied to frames with no Gaussian blur. However, as the data in the next section indicates, applying a Gaussian blur also increased the percentage of frames filtered out (indicated by a rightward shift on the trajectory curve). At this point, it is unclear whether applying a Gaussian blur is a net-positive or net-negative.

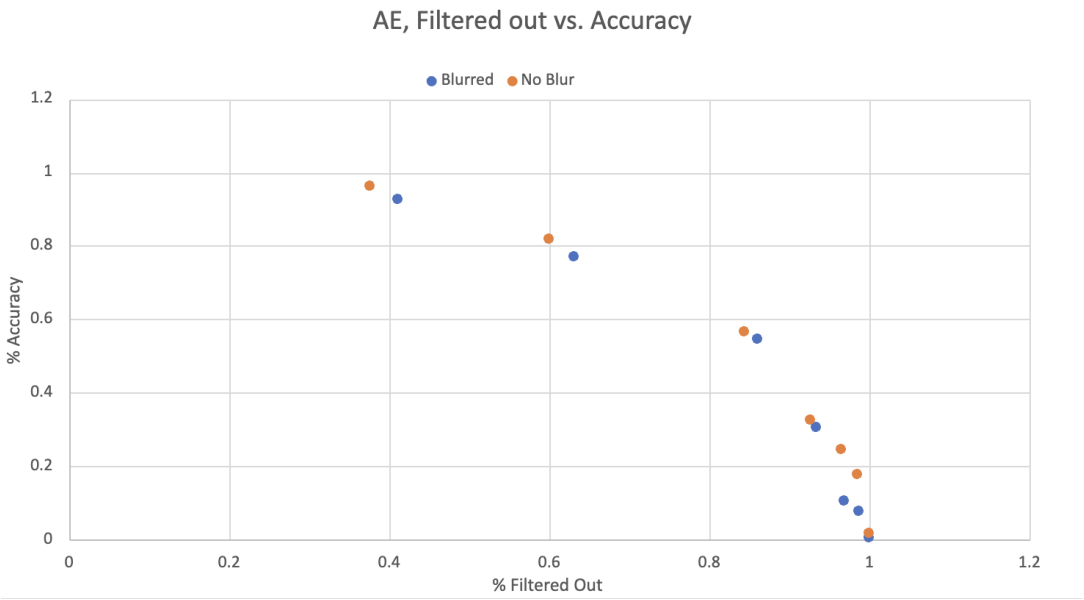
3.2 Comparing Efficacy Between Filtering Algorithms

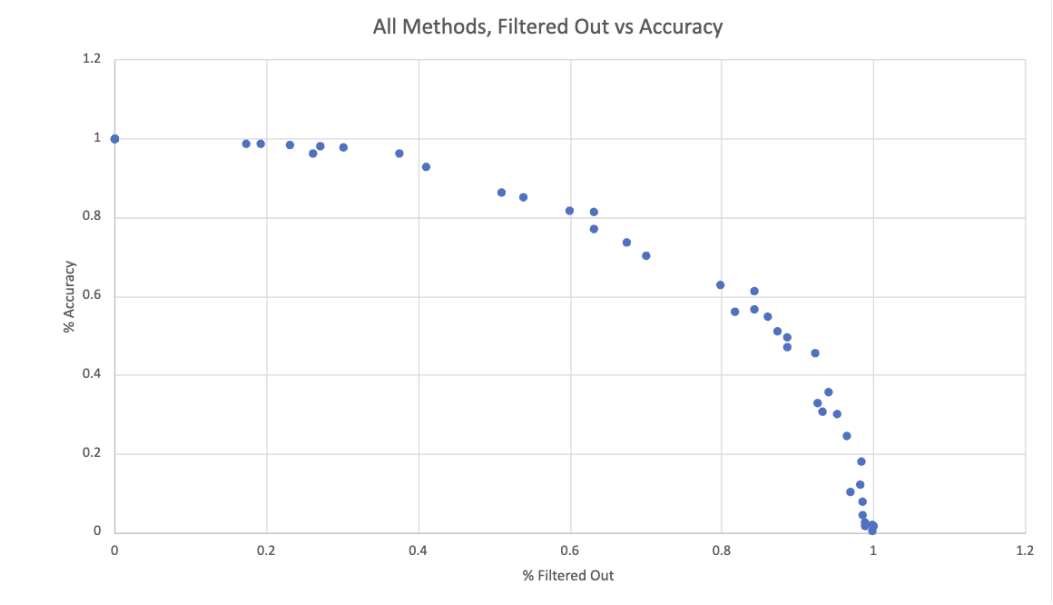
We then compared each algorithm against each other based on the percentage of frames they filtered out vs. the percentage accuracy, yielding three trajectory curves for each algorithm. Algorithms that perform better would be able to

filter out a higher percentage of frames while still preserving a high accuracy (in other words they drop down to the X-axis more gradually), whereas algorithms that perform less effectively would show a lower accuracy for the same percentage of frames filter out (meaning the drop to the X-axis more quickly).









When the curves for each of the three algorithms are plotted on the same graph for comparison, we expect them to lay on three different trajectories, indicating that each algorithm has a unique efficacy at filtering out frames while preserving accuracy; more effective algorithms drop to the X-axis more gradually and less effective algorithms more quickly. However, when we look at the results, this isn't what we see. Instead when we plot all three curves (including both

blurred and un-blurred data for each algorithm), all three lie on the same general trajectory. This yields two conclusions: first and most significantly, this data indicates that the accuracy of our model only depends on the percentage of frames filtered out, and how these frames are filtered out (whether using MSE, SSIM, or AE) has negligible or no impact on accuracy. Secondly, and proceeding from the first point, applying a Gaussian blur only had an effect insofar as it filtered out more frames (a rightward shift on the curve), but the accuracy that resulted from this still lies on the same general trajectory.

4 Related Work

Before analyzing previous works, it's important to understand frame filtering in the greater context of video analytics. Frame filtering in its essence is one of the methods used to tackle the challenges associated with the deployment of real-time video analytics data pipelines. While there exist accurate, robust models that can efficiently perform quite a few computer vision and video analytics operations, such as object detection and classification, and there exists extensive data continuously sourced from cameras everywhere, the challenge is achieving real-time inference using these models. And that is because these computer vision and video analytics models require extensive computational resources while cameras and end devices are computationally limited. To tackle this issue, many solutions were proposed, such as running the models on the cloud, where computational resources are abundant, but the challenge is getting data to the cloud, and that's where the need for frame filtering arises. To run models on the cloud, it's essential to provide stable, fast network connection between cameras and the cloud to transfer video data at real-time, which is logistically and economically expensive.

One method to tackle the network challenge is filtering irrelevant frames to reduce the size of data that needs to be transferred to the cloud and so reducing the bandwidth needed for data transfer. Reducto is an on-camera frame filtering system designed for this end goal. It filters frames dynamically based on a sequence of operations, starting with an offline server profiling to extract differencing features using traditional data analytics pipelines. Afterwards, the resulting features are processed by a diff-extractor to filter frames based on user-specified query and to pick a correct threshold for filtering, the model uses a K-means clustering model and a simple regression model based on the relationships between differencing values, filtering thresholds, and query result accuracy (Li et al. 2020). In essence, Reducto adjusts filtering choices based on the changing relationship between feature type, filtering threshold, query accuracy, and video content. The system's filtering decisions are made in real-time and are customized to each particular situation (Li et al. 2020). The system could achieve significant results; it could filter out 51–97% of frames, save 21–86% of bandwidth, and reduce 50-96% of backend computation overheads and 66–73% of query response times (Li et al. 2020).

Another method used to tackle this issue is what is known as the edge-cloud split. Instead of running frame filtering models on the end devices, we deploy binary classification models on edge devices to transfer only relevant data to the cloud. For instance, a model that reads license plates on the cloud should only process input images that have cars or license plates in the first place; thus, a quantized binary classification model will classify images depending on the presence of cars or license plates and only send those images to the cloud. One of the important models used for classification is YOLO (You Only Look Once); from the name, the data is processed through the model in a single pass. YOLO is a real-time object detection model that uses a Darknet neural network model architecture to detect objects in images and video streams. The model is based on a single neural network that predicts both the object locations and their class probabilities in one pass through the network. It divides the input image into a grid of cells and predicts a set of bounding boxes and class probabilities for each cell. Each bounding box contains information about the coordinates of the box, the confidence score for the box, and the class probabilities for the objects in the box.

There have been many upgrades to the original YOLO model. YOLO v2 used batch normalization to improve accuracy and Darknet-19 as a lighter neural network to reduce inference time (Rathi et al. 2020). YOLO v3 used logistic regression to allow for multi-label classification and used Darknet-53, a deeper model compared to Darknet-19, to improve accuracy (Rathi et al. 2020). YOLO v3-Tiny, the light-weight version of YOLO v3, is essentially a compressed YOLO v3 model with shallower neural network. It's 442% faster than previous YOLO models but has lower accuracy (Rathi et al. 2020). Despite being a lightweight model, YOLO tiny can't be deployed on end devices; it failed to achieve 1 frame per second inference on 1.5 GHZ-processor and 1 GB-RAM cameras (Li et al. 2020). However, it can be used on edge devices. From that, we can conclude that the existing light-weight binary classification models are not suitable for cameras and end devices computational power, and significant progress needs to be made to design lighter yet accurate models. Also, the only option for those binary classification models is to run on edge devices.

FilterForward is another model used to optimize edge-cloud split by detecting relevant frames. Unlike YOLO architecture that is built on the concept of single pass, FilterForward employs microclassifiers on the top of a base DNN to reuse computation and produce feature maps. FilterForward uses MobileNet architecture as the base DNN to balance between accuracy and computational needs (Canel et al. 2019). After the input image is processed using the base DNN, a set of microclassifiers, which are essentially binary classification models, run in parallel to evaluate the relevance of a particular frame to more than one query at the same time. And as the number of microclassifiers increase, the throughput measured in frame per second drops significantly. While FilterForward could process 15 fps with one microclassifier, it could hardly process 8 fps

with 2 microclassifiers and could only process 4 fps with four microclassifiers. Also, it's important to note that FilterForward was tested on a desktop with 32 GB of RAM and Intel® Core™ i7-6700K CPU. Cameras are not close to the desktop capabilities, so FilterForward is not expected to achieve real-time inference on cameras and end devices, but can achieve real-time on edge devices as long as the number of microclassifiers is reduced to minimum. Also, FilterForward is designed for scenarios, where relevant frames are expected to be rare (Canel et al. 2019). So, it's not deployable in all cases. Back to the example of the license plate reader, the majority of frames captured by camera installed on a congested highway will likely be relevant, so FilterForward wouldn't function efficiently unlike the case of a camera installed in a neighborhood street. Comparing Reducto and Filterword, Reducto filters out 93% more frames compared to FilterForward (Li et al. 2020).

Another area of research is quantizing the video analytics pipelines to suit the limited computational resources of cameras; however, this significantly jeopardizes the accuracy of those models. And since simple binary classification models, such as YOLO tiny, were not even close to achieve real-time inference on cameras, more complicated complete computer vision and video analytics data pipelines are not expected to achieve real-time inference.

5 Conclusion

This experimentation led to some interesting conclusions. Namely that regardless of the filtering method, the accuracy vs percentage filtered graph exhibited the same trend in every case. This would indicate that the filtering method chosen has no correlation to accuracy of the model, and the accuracy is rather simply a function of the number of frames filtered. Additionally, the Gaussian blur we applied to each frame performed worse in every case compared to the same image without the Gaussian blur. We are sure that this must be a by-product of a couple of factors.

Firstly, it's worth noting the obvious fact that videos can vary in many ways, including frame rate, shutter speed, bit rate, bit depth, encoding, CMOS/CCD sensor, and many other factors. For our purposes, it's likely the frame rate, shutter speed, and encoding of the two videos we chose had a significant effect on our results, arising from the simple fact that video that looks natural to the human eye intentionally blurs motion between frames, but video that works best for machine learning tasks should have as little blur between frames as possible (in other words, each frame should be a sharp, stop-motion picture). From this difference arises a tradeoff, that video optimized to be watched by a person (i.e. a movie or youtube video) should have motion blur between frames, increasing the frame-to-frame pixel difference and making it difficult for a machine learning algorithm to identify objects from the blurry image. Conversely, a video filmed to be used for machine learning should have a high shutter speed and little-to-

no motion blur, but doing so makes the video unpleasant to watch (see [4] for an example of this phenomena, particularly how unnatural the high frame rate video looks to your eye). Put simply, video optimized for human consumption is not ideal for machine learning applications and vice-versa.

Secondly, we noted that both videos which we ran the model against had significant pixel change frame to frame. This could lead to each filtering method making the same choices, not because the methods are identical but because there is so much change that every filtering method would opt to filter many frames. Another theory for why these methods may perform the same against these two videos is that the frame filtering method should be tailored specifically to the video. By tailoring the frame filtering technique to the video, it becomes possible to account for specific factors such as noise levels, motion patterns, lighting conditions, and the presence of artifacts or occlusions. This customization allows the model to learn and adapt to the specific features and patterns present in the video, improving its accuracy and performance.

Additionally, by tailoring the technique, it becomes more efficient in terms of computational resources and can be optimized to handle the specific requirements and constraints of the video analysis task at hand. So, to next steps in this research would be: (1) to increase the number of videos we run the model against and examine how the "one size fits all" approach continues to work, and (2) tailor the frame filtering technique to each video specifically to assess whether that increases accuracy or changes performance based on the filtering method.

6 Reference

1. Li, Y., Padmanabhan, A., Zhao, P., Wang, Y., Xu, G. H., Netravali, R. (2020). Reducto. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. SIGCOMM '20: Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. ACM. urlhttps://doi.org/10.1145/3387514.3405874

2. Adarsh, P., Rathi, P., Kumar, M. (2020). YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS). 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS). IEEE. https://doi.org/10.1109/icaccs48705.2020.9074315

3. Canel, Christopher Kim, Thomas Zhou, Giulio Li, Conglong Lim, Hyeontaek Andersen, David Kaminsky, Michael Dulloor, Subramanya. (2019).

4. Lisota, Kevin. Understanding Video Frame Rate and Shutter Speed. (2020). https://kevinlisota.photography/2020/04/understanding-video-frame-rate-and-shutter-speed