

Embedded implementation of a random feature detecting network for real time classification of time-of-flight SPAD array recordings

Joyce Mau¹, Saeed Afshar², Tara Julia Hamilton³, André van Schaik², Rudi Lussana⁴, Aaron Panella⁵, Jochen Trumpf⁶, Dennis Delic¹

¹Defence Science Technology Group, Edinburgh, Australia

²Western Sydney University, Sydney, Australia

³Macquarie University, Sydney, Australia

⁴Politecnico di Milano, Milan, Italy

⁵elmTEK Pty Ltd, Adelaide, Australia

⁶Australian National University, Canberra, Australia

ABSTRACT

A real time program is implemented to classify different model airplanes imaged using a 32x32 SPAD array camera in time-of-flight mode. The algorithm uses random feature extractors in series with a linear classifier and is implemented on the NVIDIA Jetson TX2 platform, a power efficient embedded computing device. The algorithm is trained by calculating the classification matrix using a simple pseudoinverse operation on collected image data with known corresponding object labels. The implementation in this work uses a combination of serial and parallel processes and is optimized for classifying airplane models imaged by the SPAD and laser system. The performance of different numbers of convolutional filters is tested in real time. The classification accuracy reaches up to 98.7% and the execution time on the TX2 varies between 34.30 and 73.55 ms depending on the number of convolutional filters used. Furthermore, image acquisition and classification use 5.1 W of power on the TX2 board. Along with its small size and low weight, the TX2 platform can be exploited for high-speed operation in applications that require classification of aerial targets where the SPAD imaging system and embedded device are mounted on a UAS.

Keywords: LiDAR, convolutional layer, embedded computing, SPAD, Single photon avalanche diode, UAS, time-of-flight, classification

1. INTRODUCTION

Autonomous systems are increasingly used for remote and dangerous operations in Defence. In particular, Unmanned Aerial Systems (UASs) are often used for remote surveillance. One major limitation of UASs is their short flight time due to on-board portable battery limitations. A larger battery size provides more power however also uses more UAS power to carry. Hence there is a requirement for any on-board sensors and processors to be small in size, and low in weight and power usage (SWaP) to prolong the flight time of UASs.

In this work, a real time classifier for live images from a 32x32 Single Photon Avalanche Diode (SPAD) camera is implemented in a mix of serial and parallel processes to classify different types of model airplanes. Identifying an airplane is a difficult problem because it is a fast-moving target and its different classes share common complex features.

SPADs detect single photons and can be paired with a pulsing laser to determine the distance to a target by measuring a laser pulse's time-of-flight (ToF) to the target; arrays of SPADs can be proficiently exploited as 3D imagers by measuring the ToF of each photon that is detected by every pixel¹. SPAD array cameras that are used in conjunction with a laser have a broad range of applications including military, meteorology, space, augmented reality, remote sensing, drone-based surveillance, advanced driver assistance systems (ADAS) and autonomous robotics.

The classifier and SPAD imaging system are run and tested on the NVIDIA Jetson TX2 board. This is to demonstrate the feasibility of having an on-board object classifier for UASs that can classify SPAD images in high speed and high accuracy. The Jetson TX2 is an embedded System-on-Module (SoM) board that contains GPUs where their parallel

processes can be called via the board's CPU using NVIDIA's parallel programming language Compute Unified Device Architecture (CUDA). Its small size and low power consumption make it suitable for UAS operation or other applications that require the embedded classifier to have low SWaP. The image first goes through a filter to determine if an object is in it, and if so, the image goes through a classifier. This is where the image is processed through a convolutional layer, then is mapped into a vector for matrix multiplication with the classification matrix to produce a vector that enables decision to be made on the most likely airplane type. This structure can also be viewed as a single layer Convolutional Neural Networks (CNNs), which is unlike many existing CNNs as they are usually multi-layered.

Currently in the literature, there are many algorithms that perform classification. However, in many cases the classification algorithm is used as one building block in an overall process that is aimed at the detection of multiple objects in a scene. In this work, detection and classification are defined to be separate processes. Detection is defined as determining whether an object is in the field of view, and classification is defined as the categorization of an object into one of the trained classes.

To the best of the authors' knowledge, while there exists algorithms that classify SPAD data² and LiDAR point clouds³, none of them have been implemented on an embedded system. For SPAD data, two CNNs have been proposed; one 2D and one 3D network². Both networks utilize a multi-layer CNN and they are used to classify 64x64 SPAD images, which are higher in resolution than the 32x32 images that are used in this paper. The networks are trained and tested separately to classify eight different objects into three different classes: airplane, chair and UAS. The average precision, recall and F1-score are used to measure the network's performance. The authors report that the 2D network achieves the same score of 0.95 for all these measures². Likewise, the 3D network is reported to achieve the same score of 0.97 for all these measures².

There is also another 3D CNN called VoxNet that is proposed for classifying LiDAR point clouds³. It utilizes the voxelization process which enables 3D point clouds to be divided into grids, which are called voxels. The voxels of the image are labelled as free, occupied or unknown depending on whether the points in the original data satisfy certain conditions. The multi-layer network is designed for general point clouds, and its creation and application on LiDAR data is demonstrated by Maturana et al.³. A subset of 14 classes from the Sydney Urban Objects dataset is used to perform training and testing of VoxNet. This data is collected using the Velodyne LiDAR. VoxNet achieves an average F1 score of 0.73³.

Despite the lack of work being conducted with SPAD images, there are many classification algorithms for standard images. Most of these are implemented on standard desktop machines. For this reason, the papers do not report on the power consumption of the overall classification system because it is not designed for energy efficiency. A classification performance analysis of the three most popular convolution neural networks is reported by Sharma et al.⁴ The CNNs investigated are AlexNet, GoogLeNet and ResNet50. The authors fine-tune these pre-trained networks using three common datasets: CIFAR-100, ImageNet and CIFAR-10. For testing, the authors look at the classification accuracy of 20 classes of objects, where 10 classes come from CIFAR-100 and CIFAR-10 each. The average accuracies are reported to be 40.0875% for AlexNet, 68.152% for GoogLeNet and 68.96% for ResNet50.

For classification algorithms that are implemented on embedded systems, there is little existing work that focuses on single object classification. Nonetheless, there is one research group that has implemented a real time classifier called ShuffleNet on a Qualcomm Snapdragon 820 processor⁵. The network optimizes the performance of 1x1 convolutions in multi-layer CNNs by using pointwise group convolutions and a novel channel shuffle operation⁵. The processor is designed for mobile devices so it is much smaller, lighter and power efficient than the Jetson TX2. In terms of GPU power, there has not been any formal comparisons made between them. The training and testing are conducted using the ImageNet 2012 classification dataset. The authors report ShuffleNet's performance to have a classification error as low as 26.3% with an execution time of 108.8ms.

In summary, to the best of the authors' knowledge, this paper proposes the first real-time embedded classifier for single object classification from SPAD time-of-flight images with a performance on par with the best existing algorithms.

This paper will start with the description of the SPAD technology in Section 2, then the algorithm will be described in Section 3. The collection of training data will be described in Section 4 which will then be followed by a discussion of the implementation in Section 5. Results and a brief discussion will be presented in Sections 6 and 7 respectively. Finally, conclusion will be provided in Section 8.

2. SINGLE PHOTON AVALANCHE DIODE (SPAD) TECHNOLOGY

The Single Photon Avalanche Diode (SPAD) is a type of solid-state photo-detector that is capable of detecting single photon, which makes it very attractive for high-sensitivity, low-illumination detection applications. Creating an array of SPADs is possible by miniaturizing SPAD devices so that they all can fit onto a single integrated circuit chip. By accompanying each SPAD device with digital circuits it is then possible to create an individual pixel or smart detector that has the ability to both count the number of photons impinging onto the detector and to time-stamp their arrival time.

This type of SPAD array photo-detector microchip allows the development of SPAD cameras which are suitable for capturing 2D images in low-light conditions and particularly attractive for Direct Time-of-Flight (DToF) 3D imaging applications using LiDAR (Light Detection And Ranging).

The DToF SPAD camera used in this work is based on a monolithic chip, fabricated in a standard silicon CMOS technology process, integrating one SPAD and one time-to-digital converter in each of the 32x32 pixels, as already presented by Villa et. al.⁶. Depth 3D-ranging information is obtained through the use of a pulsed laser operating at 660 nm, synchronized to the camera, illuminating distant target with pulses of light. The reflected photons are detected by the camera which by using a Direct Time-of-Flight (DToF) ranging method, accurately measures the time taken by photons to travel to and from the target. In order to acquire a 3D depth-resolved image of a scene rapidly, it is possible to measure the ToF information pixel-by-pixel for an entire array of pixels in the SPAD sensor and communicate all this information off camera during each pulse of the laser.

Using the images produced from the 3D DToF SPAD camera offers an ideal opportunity to develop novel classification algorithms that can deal with the classification of 3D objects in various LiDAR based imaging applications.

In this work, the SPAD camera is fitted with a Navitar NMW-12WA⁷ lens and a Thorlabs 660 nm filter⁸. The SPAD camera's field of view is set to 26.22 degrees. The laser used to obtain ToF data is a 100 mW 660 nm Coherent CUBE diode laser (name: CUBE 660-100C). It is fitted with Navitar 1-50486 12x zoom lens⁹ and is adjusted to illuminate the field of view of the SPAD camera. The laser's beam divergence is at 22.61 degrees and its focus is at 40 mm. As the laser's divergence is smaller, the laser is moved further away from the object until its beam covers the SPAD camera's field of view. The object is always placed at 400 mm away from the camera and the minimum distance of the laser is calculated to be 473 mm away from the object.

3. THE ALGORITHM

The algorithm that is used consists of three major steps; denoising, detection and linear classification. The detection step determines whether there is an object in the current image frame, that is then elaborated by the classification step which determines the type of the detected object. If no object is detected, the algorithm moves onto the next image frame.

3.1 Denoising

A moving average of 5 images is used for training, which is found to be the minimum number of images needed to average in order to remove noise without compromising the information in each image. For testing, a block average of 16 images is used instead. This will be explained in Section 5.2.

3.2 Detection

Detection is carried out by determining if there is a block of columns and rows whose sum is above a defined threshold value. The threshold values are determined empirically, by visually identifying if the calculated boundaries include most of the object. The boundary values are updated by the following:

$$E_t = (1 - k) * E_{t-1} + k * e; \quad (1)$$

where E_t is the current edge, E_{t-1} is the edge value from the previous image, and e is the intermittent edge value from determining the indices of the block of column or block of row that satisfies the threshold condition. The value k is set to 0.14 for this work. To ensure the boundary encompasses the object, the current edge value is rounded up for east and south edges while it is rounded down for west and north edges. The choice of rounding up or down depends on which side the indices start counting. Here the indices start on the west and north edges. The block size is checked for whether

it is within the required range, which is defined to be 10 to 30 for this work with model airplanes. If an object is detected, the region within the identified boundaries is used for classification. Otherwise, the algorithm waits for the next image.

3.3 Linear classification

A linear classifier is used in this work where the object is classified by determining the largest value of the product of the image vector X and the classification matrix W . The equation is written as follows:

$$XW = Y \quad (2)$$

The object that corresponds to the index of the largest value in the vector Y is the classified type determined by the algorithm. The image vector X is the histogram of the original image and its corresponding filtered images while the classification matrix W is calculated from images with known object labels. Details of them are given in the following sections.

3.4 Determining the image vector X

The image from the detector is transformed in multiple steps into the image vector X . Firstly, the denoised image passes through one convolutional layer, where it is convolved with a number of different filters. For this work, the classification performance of different numbers of filters are investigated. The filters' dimensions are 4 by 4 pixels, which are empirically determined to be the smallest possible filter size that is effective.

Secondly, the convolved images are split into positive and negative valued frames. This approach is found empirically to produce better classification accuracy than keeping the positive and negative values together. The original images are also normalized because this is found to produce better classification results. Thirdly, the original and convolved images' positive and negative matrices are cropped at the boundaries identified by the detection algorithm in Section 3.2.

For the fourth step, each of these cropped images are then summed along the row and column to form two vectors each, and then these vectors are resampled. The resampling algorithm used here is also optimized for fast processing, which will be discussed in more detail in Section 3.5. The vectors are then concatenated into the image vector X for classification.

3.5 Resampling

Zero-order hold is used in this work for upsampling where for an upsample factor of n , every element in the vector is replicated n times. For a downsampling factor of n , every $i+n$ -th element is kept where $i=\{1,2,3,4,\dots\}$. An additional step is added before resampling to decrease the processing time. It determines whether artificially extending or shortening the original vector will decrease the processing time of the resampling operation. This is because if the original vector's length is coprime with the output vector length, the upsample factor becomes very large, hence increasing the processing time. However, by changing the original vector's length by a few elements, the upsample factor can decrease significantly.

This extra step takes advantage of the noisy nature of the SPAD data to prioritize processing time over precision. Losing a few values or adding a few zeros at either end of the summed row and column vectors (from Section 3.4) will have a negligible effect on the overall information in the data. The implementation details are as follows.

The new length x for the original vector is determined by finding a minimum of the following function.

$$f(x) = \text{Penalty function}(x) * \text{LCM}(x, \text{output vector length}) \quad (3)$$

where

$$\begin{aligned} \text{Penalty function}(x) \\ = \text{remapping resistance factor} * (x - \text{original vector length})^2 + 1 \end{aligned} \quad (4)$$

The LCM in equation (3) is a measure of the relative computational cost of resampling. The penalty function from equation (4) measures the loss in information for removing elements or adding additional zeros to the original vector before resampling. The remapping resistance factor is used to determine how much difference in length can be tolerated. The higher the value, the lesser the tolerance. The remapping resistance factor in this work is set to 1.25.

In the implementation, a lookup table is calculated for all possible original vector lengths. These are between 1 to 31 because all the images are cropped before resampling. Only one lookup table is required since the output vector length is always 32 for this work.

Once the desired input vector length is determined, the length of the original vector is changed by adding zeros to either side or removing elements as required. The number of elements to remove or add on the left and right is determined by the following formulas:

$$\begin{aligned} & \text{Number of elements to add or remove to the left} & (5) \\ & = \begin{cases} \frac{\text{new length} - \text{original length}}{2} & \text{if length difference is even} \\ \frac{\text{new length} - \text{original length} - 1}{2} & \text{if length difference is odd} \end{cases} \end{aligned}$$

$$\begin{aligned} & \text{Number of elements to add or remove to the right} & (6) \\ & = \begin{cases} \frac{\text{new length} - \text{original length}}{2} & \text{if length difference is even} \\ \frac{\text{new length} - \text{original length} + 1}{2} & \text{if length difference is odd} \end{cases} \end{aligned}$$

Once the original vector has been changed to the desired length, then it is upsampled using zero order hold and down sampled by choosing every n-th element of the vector.

3.6 Determining the classification matrix W

The classification matrix in this work is determined using the following operation.

$$W = (X_{train}^T * Y_{ground\ truth})^T (X_{train}^T * X_{train} + b)^{-1} \quad (7)$$

where b is the regularization factor added to the diagonal of the matrix product. X_{train} is a matrix that contains the image vectors of the recorded frames of the airplanes in each row. In this work where four different airplanes are classified, the regularization factor is set to 100. This value is found through trial and error on what value provides the best classification matrix. Equation (7) is derived by using the pseudoinverse of X_{train} and applying the Ridge Regression (also called Tikhonov regularization).

4. MODEL AIRPLANE DATASET

The model airplane dataset is collected from dropping each of the four types of model airplanes in front of the SPAD camera and laser system. This simulates the airplanes moving across the camera's field of view at a high speed. The model airplanes are shown in Figure 1 and are Tu-128, Su-35, Su-24 and MQ-9. Their sizes vary but are not larger than 17 cm by 12 cm by 2 cm. This allows them to be within the camera's field of view when imaged at a closer distance.



Figure 1: Model airplanes used for imaging.

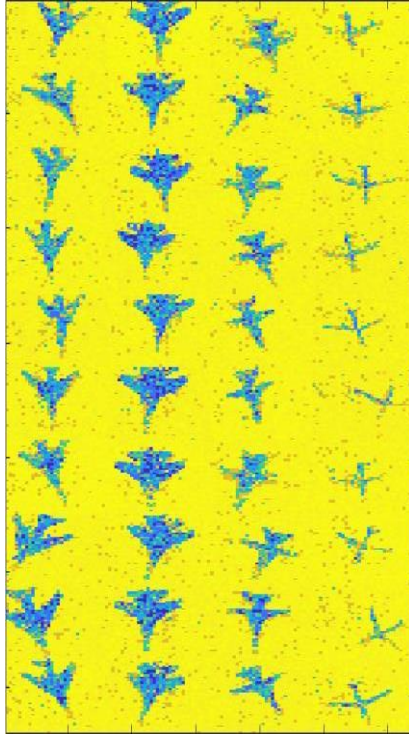


Figure 2: A selection of images from the training dataset. The airplane types are (from left to right) Tu-128, Su-35, Su-24 and MQ-9.

A separate recording is made for each airplane drop and each airplane is dropped 100 times. The model airplanes are painted white to standardize the recordings. The recordings collected are also flipped left to right to double the number of images available for training. This gives a training set of 800 recordings in total where each recording on average has 427 images. Figure 2 shows a sample of the collected data after denoising. The airplanes are dropped in different orientations 40 cm from the front of the camera to simulate a large airplane flying past the camera's field of view. The laser pulse width is 17 ns and the camera has a timing resolution of 390 ps, hence making the model airplanes correspond to pixel values ranging between 44 different time bins.

5. IMPLEMENTATION

As this work is focussed on the real time classification of SPAD images on the Jetson TX2 board, only the inference part of the algorithm is implemented on the TX2. The algorithm is trained and tested offline with different numbers of filters before the classification matrix and filter values are transferred to the TX2 board. The 200 recordings for each airplane are split randomly in half for training and testing. This allows for fast training and testing to be executed before the optimal parameters are transferred to the TX2 board. Furthermore, the size of the training datasets will not be limited in the future by the memory capacity of the TX2 board.

5.1 Software implementation

The implementation of the classifier is designed to adhere to three principles: multithread where appropriate to increase performance, make use of the L1 caches when possible, and offload specific functions to the GPU only when they can benefit from concurrent operations. Image acquisition and classification are split into two separate threads in the implementation. This design choice is made because image acquisition requires less time than classification, so the implementation can acquire the next image before the classifier finishes processing the current image. Hence, the classifier only idles during data transfer between threads. The TX2 board contains an ARM Cortex A57 with 80 KiB L1 cache which is sufficient for accommodating 1024 numerical values for each image frame. Therefore, the execution time of data transfers between threads is minimal. Furthermore, cache hits are able to be utilized to perform basic arithmetic functions serially. This is much faster overall than transferring data from the CPU to the GPU, performing the same arithmetic functions concurrently, and then transferring the data back to the CPU.

Implementing parts of the classifier in parallel processes is another way to improve the classification time. Operations are implemented in parallel using CUDA when the reduction in time resulting from concurrent execution in the GPU outweighs the extra time spent transferring data to the GPU. This is true for convolution, summation of matrices to vectors and matrix multiplication. In particular, the summation of matrix rows and columns is implemented using the CUDA warp shuffle functions. This accesses the image at the register level, allowing very fast summation operations. Furthermore, two design steps are implemented to reduce the amount of time spent on transferring data between CPU and GPU. Firstly, the functions that are parallelized are only called once. For example, this is achieved in convolution by concatenating multiple filters into one array and multiple output images into another. Secondly, the GPU's shared memory is used to store values.

5.2 Denoising of real time images

A block average of 16 images is implemented instead of a moving average of 5 images for real time classification. This is because a burst of 16 raw images is transferred to the TX2 at each camera acquisition and extra steps will be required if a moving average of 5 images is to be implemented. Since camera acquisition and classification are implemented in two separate threads as explained in Section 5.1, calculating an average of 16 frames does not risk losing any real time information whilst keeping the processing time short.

5.3 Additional check for real time detection

An additional detection check is added to stop any background noise to be detected as an object for classification. The detected object is only used if its distance is within the distance range which the airplanes are placed. This is determined by checking whether the 5th percentile's value of the frame is less than a certain threshold value. The threshold value is determined by calculating the 5th percentile value of frames which contain the airplane being held at the furthest possible point.

6. RESULTS

6.1 Denoising and detection

Denoising and detection are implemented as described in Sections 3.1 and 3.2. Example outputs are shown in the following figures:

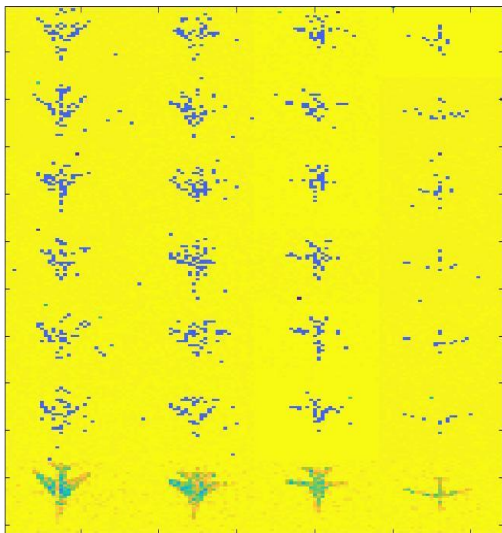


Figure 3: Effect of averaging 16 frames on the TX2 board, here the first 6 rows contain the raw images and the last row contains the averaged images.

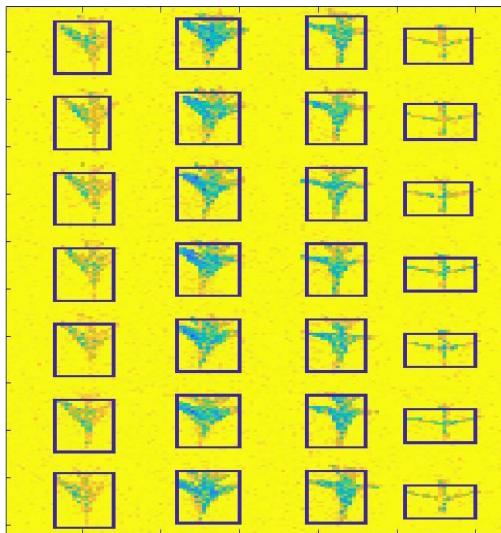


Figure 4: Detection identifying the bounding box of each airplane.

The last images in each column in Figure 3 are averaged images of the 16 raw images, only 6 of the 16 raw images are shown to illustrate the nature of the raw images. Figure 4 shows the model airplane being detected by the bounding box.

6.2 Filters selection

In this work, the classification accuracy of the implementation on the TX2 board is tested for different numbers of random filters in the convolution layer. 10 sets of random filters are generated offline for each number of filters to decide which set should be used on the TX2 based on the resulting classification performance. 200 recordings of each airplane are split in half randomly 20 times to train and test the algorithm for each set of random filter values. This means the training and testing sets contain 107800 images each. Recordings are split instead of images to prevent the training or testing dataset from containing too many images of the airplanes at a certain position in the drop.

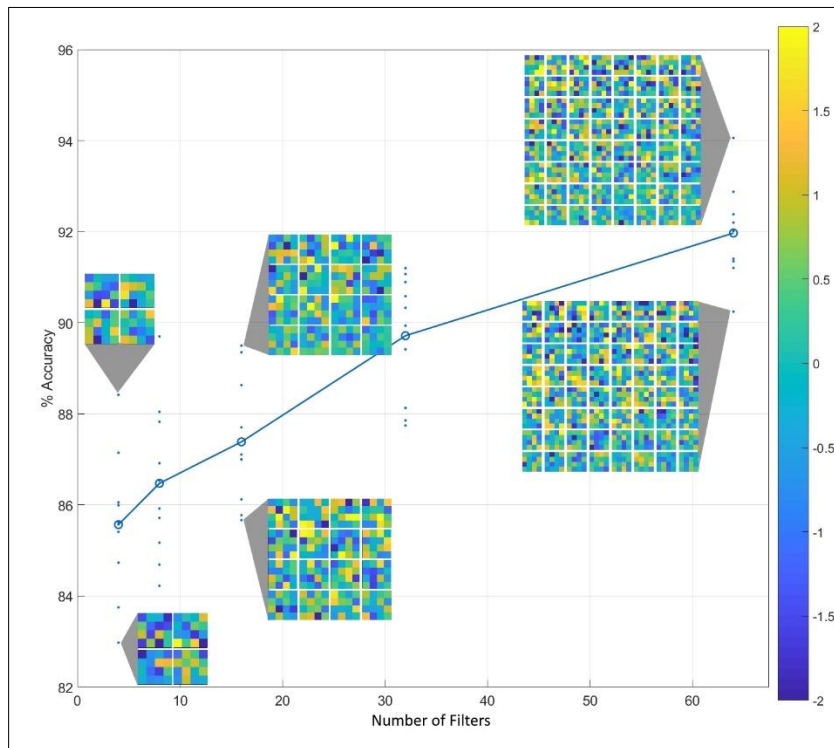


Figure 5: Classification Accuracies per Frame of Different Filter Values for Different Numbers of Filters

Their classification accuracies are shown in Figure 5. Each '+' data point in the figure represents the mean accuracy of one set of random filters over 20 random airplane data splits. Each 'o' data point represents the mean accuracy of all the sets of filters for that number of filters.

6.3 Accuracy results

The classifier is tested with different numbers of filters and with the airplane being held in three different ways: downwards vertically, angled downwards and dynamically. The third way is done by moving the airplane in a small circular motion to simulate the camera experiencing vibrations whilst on the UAS. For each number of filters and each way of holding the model airplane, 32 classification experiments are conducted for each airplane type. As an example, Figure 6 shows some of the live image feed used to test the classifier for each way of holding the airplane. The number of filters that is used in the classifier during this testing is 16. The image feeds for the dynamic case look similar to the image feed for the vertical case because for all three cases, the airplanes are placed 40 cm away from the camera, and the dynamic case only has the airplanes moving within a radius of 1.5 cm from that position.

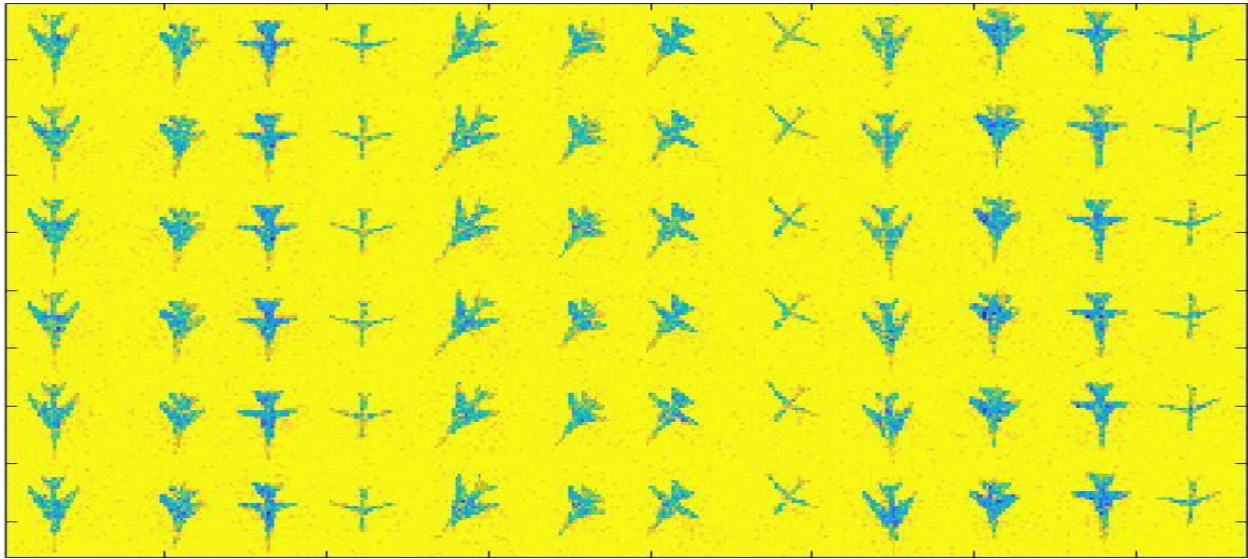


Figure 6: Images saved from testing the classifier with 16 filters using a live image feed, each group of four columns are chosen from each way of holding the airplane (starting from the left): a) vertical, b) angled, c) dynamic (where the airplane is moving in a circle).

Confusion matrices are used to analyse the performance of the classifier with different numbers of filters and in different ways of holding the model airplanes. The predictions from the set of 32 classifications are recorded and used for calculating the performance measures in Table 1. The predictions are recorded as percentages for each airplane type. Recall and overall accuracy share the same row because they have the same values since the number of testings for each airplane are the same.

Table 1: Performance measures for different number of filters.

Measures	Way of holding the airplane	Number of Filters				
		4	8	16	32	64
Overall Accuracy (%) and Macro Recall (%)	Vertical	90.63	96.88	97.66	98.44	100.00
	Angled	87.50	90.63	94.53	96.88	99.22
	Dynamic	82.81	89.06	93.75	95.31	96.88
	Average	86.98	92.19	95.31	96.88	98.70
Macro Precision (%)	Vertical	92.37	97.22	97.77	98.48	100.00
	Angled	88.81	92.45	94.91	97.01	99.24
	Dynamic	85.99	91.05	94.64	95.86	97.06
	Average	89.06	93.57	95.77	97.12	98.77
Macro F1 measure (scale of 0-1)	Vertical	0.91	0.97	0.98	0.98	1.00
	Angled	0.88	0.92	0.95	0.97	0.99
	Dynamic	0.84	0.90	0.94	0.96	0.97
	Average	0.88	0.93	0.96	0.97	0.99
Average Error Rate (%)	Vertical	4.69	1.56	1.17	0.78	0.00
	Angled	6.25	4.69	2.73	1.56	0.39
	Dynamic	8.59	5.47	3.13	2.34	1.56
	Average	6.51	3.91	2.34	1.56	0.65

In further test runs we simulate an airplane flying past the camera. The camera is fixed onto a gimbal and moved at a constant speed to scan a static vertical model airplane from bottom to top. The classifier is only stopped after the airplane exits the field of view. The number of detections during each scan is recorded and the classifier’s confidence rate for each airplane type during the midpoint of the scan is shown in Table 2 below. 64 filters are used in the classifier for these test runs because Table 1 shows this number of filters has the best performance.

Table 2: Classification performance of classifier when model airplane is moving at a constant speed in field of view.

Airplanes scanned	Confidence Rate of the scan’s middle frame (%)				Number of detections
	Tu-128	Su-35	Su-24	MQ-9	
Tu-128	91.30	0	8.70	0	44
Su-35	0	86.96	13.04	0	45
Su-24	0	0	100	0	43
MQ-9	0	0	4.76	95.24	40

6.4 Execution time

The average execution time for the TX2 board to classify an image is presented in Table 3. These are the average execution times for the dynamic case. The overall execution time is only the classification time because image acquisition and classification are implemented in two separate threads as discussed in Section 5.1. Also, the amount of time it takes for the SPAD camera to see the object will depend on the laser strength and the distance to the target hence this is not included in the execution time.

Table 3: Average execution time of classifier using different number of filters (ms).

	Number of Filters				
	4	8	16	32	64
Tu-128	31.99	30.00	40.61	46.22	64.79
Su-35	28.63	32.39	39.76	48.43	66.45
Su-24	29.06	31.85	35.98	46.79	62.99
MQ-9	31.04	31.38	38.07	49.80	66.04
Average	30.18	31.40	38.61	47.81	65.07

The power consumption of the TX2 board when executing the classifier in real time is found to be 5.1 W.

7. DISCUSSION

From the classification results in Table 1, it is evident that the classifier performs best with 64 filters. In this setting, its overall classification accuracy reaches 98.7% and its error rate is at its lowest of 0.65. Its F1 measure is 0.99 given its recall and precision scores are at 98.7 and 98.77 respectively. Overall, the classification accuracy improves with increasing number of filters. The average execution time for this classifier with 64 filters is around 73.55 ms. The power consumption for the classifier to execute in real time is between 5.1 W. As there is no reported power consumption in other related work, the power consumption cannot be compared.

The performance of the classifier is comparable to other existing CNNs presented in the literature. Using 16 or more filters result in the classifier having a higher precision, recall and F1 score than the 2D network proposed by Ruvalcaba-Cardenas et. al.². However, there are a few differences between the 2D network and this paper’s proposed network. First of all, the 2D network uses a VGG-16 model that is pre-trained on 14 million standard images, and adds four layers to fine-tune the network to classify SPAD images². This makes a total of 18 layers which is much larger than this paper’s random feature detecting network. Secondly, the training data is different. An average of 4270 SPAD images per class are used for training in this paper while 641 SPAD images per class are used to fine-tune a pre-trained model for the 2D network². Thirdly, this paper’s images are 32x32 in size and Ruvalcaba-Cardenas et. al.’s original images are 64x64 which are then scaled up to 224x224 to be compatible with the VGG-16 model’s input layer². Furthermore, Ruvalcaba-Cardenas et. al. are classifying eight different objects into three classes (airplane, chair, UAS)² while this paper is

classifying four different airplanes into four different classes (Tu-128, Su-35, Su-24, MQ-9), where there is a one-to-one correspondence between each object to a class.

For the 3D network proposed for SPAD images², this paper’s classifier has equal performance when using 32 filters and performs better at 64 filters. Point clouds are used as input data for the 3D network. This is different to this paper’s work where the SPAD images are interpreted as 2D images with the values giving distance information. Furthermore, the object’s pixels are projected over the z-plane to create a 3D point cloud for the 3D CNN². This is needed because the objects are being timed at the same bin by the SPAD and laser system, hence appearing flat in a 3D map². The 3D network also has a different structure to this paper’s network, where the 3D network has 11 layers² and this paper’s network can be viewed as a single layer CNN.

The network in this paper also performs better than VoxNet³ for all numbers of filters. The average F1 score of the Multi Resolution VoxNet is 0.73³ while this paper’s network has a lowest F1 score of 0.88 at 4 filters. However, this work is only classifying 4 classes of objects while VoxNet is tested to classify 14 classes of objects³. VoxNet’s training data size is unknown so it is difficult to determine whether we are using a larger dataset for training. It is known that the dataset used (Sydney Urban Objects dataset) for VoxNet only has 631 images¹⁰ for each class of object, and this is much smaller than this paper’s training dataset of 4270 images per class. VoxNet also uses many more layers than this paper’s network that can be viewed as a single layer CNN. Its CNN consists of two identical 5-layer networks working in parallel and then one final classification layer. The two parallel networks are used to process the data that is voxelized by two different sized grids, 0.1m and 0.2m. This is to ensure the classifier is rotationally invariant.

Table 4: Classification accuracy of different CNNs

	AlexNet	GoogLeNet	ResNet50
Average accuracies	40.0875	68.152	68.96
Max accuracy	95.9 (Truck from CIFAR-10)	97.1 (Truck from CIFAR-10)	99.2 (Motorcycle from CIFAR-10)
Min accuracy	0 (Bed from CIFAR-100)	0.84 (Streetcar from CIFAR-100)	33.4 (Table from CIFAR-100)

The network in this paper has an overall accuracy that is comparable to the accuracies reported on AlexNet, GoogLeNet and ResNet50 by Sharma et. al.⁴. Table 4 shows the average, minimum and maximum accuracies of the three CNNs at classifying 20 different classes of objects. Classes from CIFAR-100 and CIFAR-10 each make up half of the 20 testing classes. The accuracies for each of the classes vary a lot and are much lower than this paper’s network accuracy for any number of filters. Even though the number of testing classes are greater than the classes used in this paper, all three of these CNNs have far more layers than the proposed random feature detecting network proposed in this paper. AlexNet has 8 layers¹¹, GoogLeNet has 22 layers¹², and ResNet50 has 50 layers¹³. Also, the authors take the pre-trained models of these CNNs and further fine-tune these models with three datasets: CIFAR-100 with 32x32 RGB images, ImageNet with varying sizes of RGB images, and CIFAR-10 with 32x32 RGB images. Further fine-tuning is possible because these models allow transfer learning, which is also the case for the 2D network proposed by Ruvalcaba-Cardenas et. al.² that stemmed from VGG-16 to classify SPAD images. This is different to this paper’s network as it is unable to store memory from previous learning.

Furthermore, the minimum accuracies of these three networks are always from one of CIFAR-100’s classes and the maximum accuracies always come from one of CIFAR-10’s classes. One possibility is that the difference between CIFAR-10 and CIFAR-100 is that CIFAR-10 only has 10 classes while CIFAR-100 has 100 classes with 20 superclasses. Superclasses are a broader category that encompasses more than one of the classes such that each image had two labels, one representing its class and the other representing its superclass. Hence, the training for CIFAR-100 is performed with each of its images containing two labels, which is different to the training dataset used to test this paper’s random feature detecting network. We only use one label for each image.

Finally, this paper’s random feature detecting network for all numbers of filters has a much smaller error rate than ShuffleNet⁵. The network’s highest error rate is 6.51% when using 4 filters whereas ShuffleNet’s lowest error rate is 26.3%⁵. ShuffleNet is tested against the ImageNet 2012 classification dataset⁵, which is different to the dataset used in this paper. There are multiple error rates because ShuffleNet is tested with different numbers of filters at each layer and

different numbers of pointwise convolutions to be grouped at a time. However, it is not clear how the authors of ShuffleNet calculated the classification error. For this paper, the error rate is calculated by averaging the percentage of false negatives and false positives for each model airplane. When comparing the execution times, the real time classifier in this paper has an execution time that varies between 34.30 and 73.55 ms for different numbers of filters whereas ShuffleNet's times are between 15.2 to 108.8 ms for 224x224 images⁵. The values vary because the authors measure the time of the three highest performing configurations of ShuffleNet. Further testing is required before arriving at any conclusions regarding execution time.

8. CONCLUSION

A real time classifier for time-of-flight SPAD images is implemented on the NVIDIA Jetson TX2 board. It processes the images with a convolutional layer and uses matrix multiplication to determine the highest likelihood object class. It has been tested to classify four different types of model airplanes (Tu-128, Su-35, Su-24 and MQ-9) when using five different numbers of filters (4, 8, 16, 32 and 64). The classifier achieves an overall classification accuracy of 98.7% with an F1-score of 0.99 when using 64 filters. Its execution time varies between 34.30 and 73.55ms depending on how many filters are used and its power consumption is between 5.1 W. Overall, the results show that the proposed classifier can be executed on a TX2 board that is mounted on a UAS as well as for low SWaP applications.

9. FUTURE WORK

There are many potential branches in the future for this work. Firstly, further speedups can be investigated to decrease the execution time of the classifier. Secondly, more classes of planes or other objects can be introduced for classification. Thirdly, the system's robustness can be improved further to allow eventual implementation as a neuromorphic system-on-chip.

10. ACKNOWLEDGEMENTS

The authors of this paper would like to thank Geoff Day from DST Group for helping with the testing of the Polimi SPAD camera, Dr Vladimiro Devrelis from Ballistic Systems Pty Ltd, Maurizio Gencarelli from DST Group and Lindsey Paul from Queensland University of Technology (QUT) for their assistance in testing the classifier's performance. The work is also co-funded by NATO SPS project 984840.

REFERENCES

- [1] Lussana, R., Villa, F., Mora, A.D., Contini, D., Tosi, A., and Zappa, F., "Enhanced single-photon time-of-flight 3D ranging," *Optics Express* 23(19), 24962 (2015).
- [2] Ruvalcaba-Cardenas, A.D., Scoleri, T., and Day, G., "Object Classification using Deep Learning on Extremely Low-Resolution Time-of-Flight Data," in *2018 Digital Image Computing: Techniques and Applications (DICTA)*, 1–7 (2018).
- [3] Maturana, D., and Scherer, S., "VoxNet: A 3D Convolutional Neural Network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 922–928 (2015).
- [4] Sharma, N., Jain, V., and Mishra, A., "An Analysis Of Convolutional Neural Networks For Image Classification," *Procedia Computer Science* 132, 377–384 (2018).
- [5] Zhang, X., Zhou, X., Lin, M., and Sun, J., "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6848–6856 (2018).
- [6] Villa, F., Lussana, R., Bronzi, D., Tisa, S., Tosi, A., Zappa, F., Dalla Mora, A., Contini, D., Durini, D., et al., "CMOS Imager With 1024 SPADs and TDCs for Single-Photon Timing and 3-D Time-of-Flight," *IEEE Journal of Selected Topics in Quantum Electronics* 20(6), 364–373 (2014).
- [7] Navitar, "Navitar Machine Vision - 1/2" Format Lenses.", <<https://www.ivsimaging.com/Products/Lenses/Navitar-Lenses/Navitar-Machine-Vision-Lenses/Navitar-NMV-12WA>> (19 March 2019)
- [8] Thorlabs, "FB600-10 - Ø1" Bandpass Filter, CWL = 600 ± 2 nm, FWHM = 10 ± 2 nm.", <<https://www.thorlabs.com/thorproduct.cfm?partnumber=FB600-10>> (19 March 2019)

- [9] Navitar, "Navitar 1-50486 Navitar 12X zoom lens with 2mm fine focus Description : Navitar 12X zoom lens with 2mm fine focus," <<http://www.inspection.ie/navitar-1-50486-navitar-12x-zoom-lens-with-2mm-fine-focus-description-navitar-12x-zoom-lens-with-2mm-fine-focus.html>> (18 March 2019)
- [10] Quadros, A., Underwood, J., and Douillard, B., "Sydney Urban Object Dataset," 2013, <<http://www.acfr.usyd.edu.au/papers/SydneyUrbanObjectsDataset.shtml>> (18 March 2019)
- [11] Krizhevsky, A., Sutskever, I., and Hinton, G.E., "ImageNet classification with deep convolutional neural networks," *Communications of the ACM* 60(6), 84–90 (2017).
- [12] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A., "Going Deeper with Convolutions," arXiv:1409.4842 [cs] (2014).
- [13] He, K., Zhang, X., Ren, S., and Sun, J., "Deep Residual Learning for Image Recognition," arXiv:1512.03385 [cs] (2015).