Convolutional Neural Network Designed for On-orbit Binary Image Classification on a 1U CubeSat

Convolutional Neural Network Designed for On-orbit Binary Image Classification on a 1U CubeSat

By

Abhas Maskey

17595903

Summited to the

Department of Applied Science for Integrated System Engineering

Graduate School of Engineering

Kyushu Institute of Technology, Kitakyushu, Japan

In partial fulfillment of the requirement for the degree of

Doctor of Philosophy

In

Applied Science for Integrated System Engineering

25.09.2020

ABSTRACT

As of 2020, more than a thousand CubeSats have been launched into space. The nanosatellite standard allowed launch providers to utilize empty spaces in their rockets while giving educational institutions, research facilities and commercial start-up companies the chance to build, test and operate satellites in orbit. This exponential rise in the number of CubeSats has led to an increasing number of diverse missions. Missions on astrobiology, state-of-art technology demonstration, high revisit-time earth observation and space weather have been implemented. In 2018, NASA's JPL demonstrated CubeSat's first use in deep space by launching MarCO A and MarCO B. The CubeSats successfully relayed information received from InSight Mars Lander in Mars to Earth.

Increasing complexity in missions, however, require increased access to data. Most CubeSats still rely on extremely low data rates for data transfer. Size, Weight and Power (SWaP) requirements for 1U are stringent and rely on VHF/UHF bands for data transmission. Kyushu Institute of Technology's BIRDS-3 Project has downlink rate of 4800bps and takes about 2-3 days to reconstruct a 640x480 (VGA) image on the ground. Not only is this process extremely time consuming and manual but it also does not guarantee that the image downlinked is usable. There is a need for automatic selection of quality data and improve the work process.

The purpose of this research is to design a state-of-art, novel Convolutional Neural Network (CNN) for automated onboard image classification on CubeSats. The CNN is extremely small, efficient, accurate, and versatile. The CNN is trained on a completely new CubeSat image dataset. The CNN is designed to fulfill SWaP requirements of 1U CubeSat so that it can be scaled to fit in bigger satellites in the future. The CNN is tested on never-before-seen BIRDS-3 CubeSat test dataset and is benchmarked against SVM, AE and DBN. The CNN automatizes images selection on-orbit, prioritizes quality data, and cuts down operation time significantly.

Keyword; Image Processing, Convolutional Neural Network, 1U CubeSat, Automatization

TABLE OF CONTENTS

+

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 CubeSat Technology and Access to Space

Nanosatellites are defined as satellites that weigh between 1-10kg. CubeSat is a nanosatellite standard first designed and proposed in 1999 by Prof. Jordi Puig-Suari, then professor at California Polytechnic State University (CalPoly), and Prof. Bob Twiggs, then professor at Stanford Univeristy's Space Systems Development Laboratory (SSDL) [1]. The standard defines a base unit (1U) as weighing up to 1.33kg with a volume of 10x10x10 cm3. 1U can be stacked to allocate more size, weight, and power (SWaP). The standardization allowed both launch providers to utilize unallocated space inside the rocket while giving educational, research and commercial startups the opportunity to build and launch low-cost space systems. To keep costs low and make space affordable, CubeSat utilizes Commercial-off-the-Shelf (COTS) technology and amateur band for communication. As of 2020, specialized products aiming at CubeSat developers are available from an array of vendors [2] and communications have shifted to other bands as technology has matured [3][4].

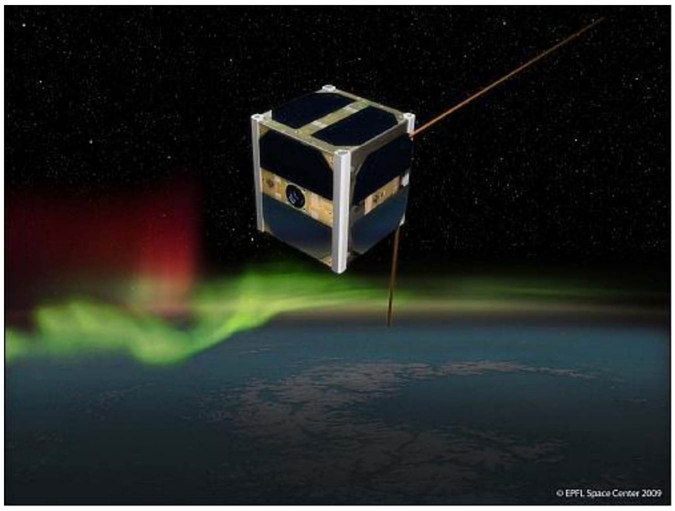


Figure 1. Switzerland's first satellite, SwissCube is a 1U CubeSat [5]

With over a thousand CubeSats launched [6], CubeSats have democratized access to space. Not only has the platform provided the opportunity for educational institutions to work on space systems, but it has also allowed nations to put their first satellites into orbit. Since launch of SwissCube, Switzerland's first 1U satellite in 2009 [5] shown in Fig. 1, several other countries have taken the same initiative too. Hungary, Poland, and Romina had all built their first satellites by 2012 [7]. The Joint Global Multination BIRDS Satellite Project (BIRDS) of Kyushu institute of Technology (Kyutech) led by Prof. Mengu Cho took a step further by empowering nations with limited adequate resources to build, test and launch their first satellites. As of 2020, Mongolia, Bangladesh, Ghana, Bhutan, Sri Lanka, Nepal, and Paraguay have built their first satellites in collaboration with Kyutech. All satellites are 1U.

1.2 CubeSat Missions

Democratization of space has led to diversifying space missions as well. Planet Inc. has so far deployed major chunk of CubeSats that have been deployed in orbit -351 as of April 2020. At any given time, the earth imaging company has over 100 active 3U CubeSats collecting 250 million square km of imagery every day [8]. For context, the earth's surface area is roughly 500 million square km. Spire Global, Inc. has launched over 100 3U CubeSats [9] for maritime tracking and safety and uses GPS occulation to collect data which can be used for weather forecasting and atmospheric research [10]. Besides these, CubeSats have been used for astrobiology, technology demonstration and space weather [11]. Until 2018, all satellites had been placed on LEO. National Aeronautics and Space Administration (NASA)'s Jet Propulsion Laboratory (JPL) demonstrated CubeSats deep space capability by launching twin 6U CubeSat, Mars Cube One A and B (MarCO). MarCO successfully relayed information from NASA's InSight Mars lander to Earth using high gain X-band antenna [12]. The CubeSats conducted a flyby, managing to take images of Mars for the first time [13]. Artist's rendering of the two satellites in space is shown in Fig. 2.

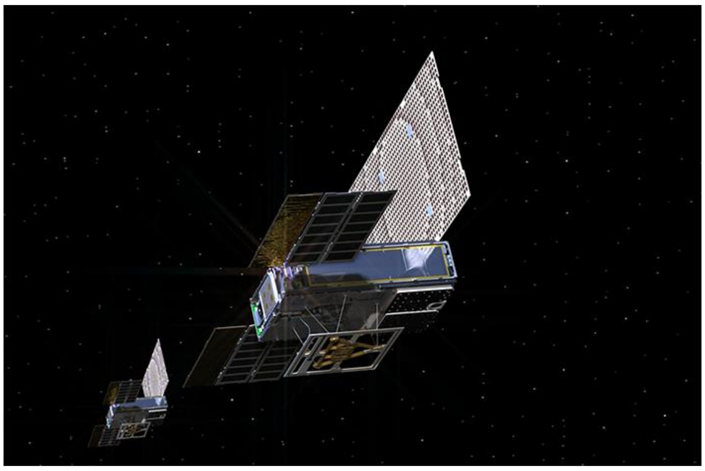


Figure 2. Artist's rendering of NASA's deep space CubeSat MarCo A and B [14]

Increasing complexity in missions require increased access to data. While comparatively "few efforts have been made to offer communication solutions using CubeSats" [4], NASA's 2018 report on State of the Art of Small Spacecraft Technology shows mature use of Very High Frequency (VHF), Ultra High Frequency (UHF) and S-band for data transmission and communication. Bigger data require faster data rate which in-turn require higher frequency bands. Planet Inc. uses proprietary X-band tuned to achieve an average of 160 Mbps [15] to downlink large volume of earth imaging data. Astro Digital's 16U remote sensing Landmapper-HD downlinks at 300 Mbps in Ka-band. CubeSats, however, normally operate in the amateur VHF and UHF [4] . DICE mission has achieved a downlink rate of 3 Mbps using UHF. 3U CSSWE and RAX-2, which used the 9600bps amateur UHF band, downloaded 60 and 242MB of data respectively [16]. Unfortunately, "CubeSats do not get more than a few MB over the course of a mission" [16]. This was true in 2012, and is true in 2020 as Kyutech's BIRDS-3 1U satellite case will illustrate.

1.3 Problem Statement: 1U Data Rate Bottleneck

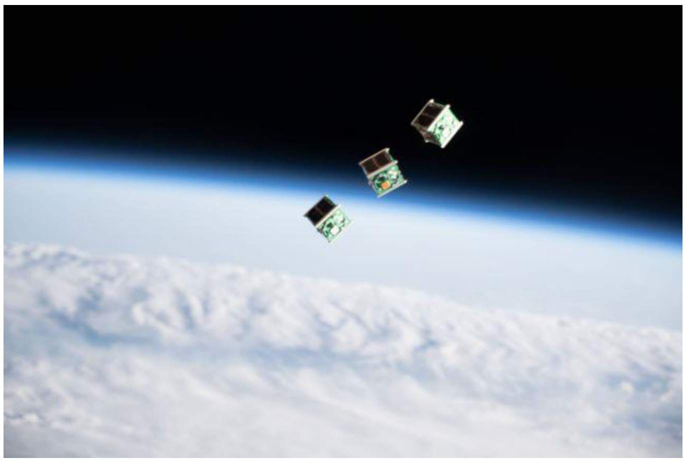


Figure 3. BIRDS-3 CubeSats just after deployment from ISS on June 17, 2019 [17]

This thesis uses BIRDS-3 1U CubeSats to study limitations imposed by CubeSat's smallest standard, a 1U CubeSat, on conducting data intensive missions such as taking images from space. The project has a constellation of three 1U CubeSats; NepaliSat-1 from Nepal, Raavana-1 from Sri Lanka and UGUISU from Japan. The satellites were launched on April 17, 2019 and deployed from ISS on June 17, 2019 as shown in Fig 3. All satellites have since been operational and missions have been successfully conducted. Operations are conducted daily. Given that each satellite has been operational for a year, the project has clocked in three years of cumulative operation time. This experience provides a good understanding on the difference between theorical and practical estimates to downlink data rates. This thesis assumes that the information taken from BIRDS-3 can be applied generally for 1U CubeSats.

Each of the three satellites is equipped with an RGB camera. The camera is placed primarily for outreach and media purposes. The COM is limited to 4800bps in the amateur UHF band. There is a 35 to 45 minute communication window per satellite per GS per day. The camera has a hardware capability to take 5MP images. However, the software design has limited the camera to take maximum VGA (640x480) resolution. The resolution has the optimum size for social media distribution while being small enough to downlink in short amount of time. The Ground Sampling Distance (GSD) is 1.2km for International Space Station (ISS) orbit of 400km.

The camera is placed primarily for outreach and media purposes. Since it is the first satellite for Sri Lanka and Nepal, taking images of their home countries provides proof to the public that the satellites exist and are functional. This generates awareness and excitement. Not all images, however, are deemed fit for public consumption. BIRDS-3 has active stabilization but no pointing. The images are therefore categorized as either "good" or "bad." The "bad" images include saturated, space-faced, very cloudy and sunburnt images. The "good" are images that face towards the earth. The "good" images are processed and released every Friday through the official website and social media channels. The selection is done by the team.

The challenge has been to obtain “good” images. In the first three months of operations, 30 images were downlinked. Of that, 12 images were “good” shown at the left of Fig. 4 and the rest were classified as “bad” as shown at the right. Majority of data that were being downlinked was not being used. Improvements must be made to increase the number of “good” data. For that, an understanding of bottlenecks for the system must be made.

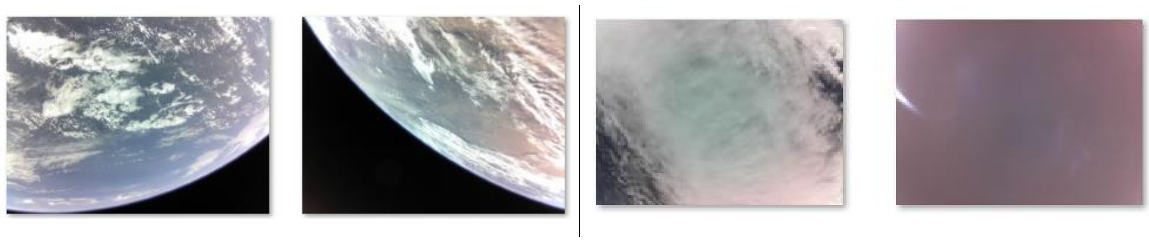


Figure 4. Two images each that are classified "good" (left) and classified bad (right)

BIRDS-3 satellites have downlinked about 150 images using UHF band in the first year of operation from three different satellites. That is roughly twelve images per month. Ideal calculations show that much more data could have been downlinked. Factors such as bad weather, sharing of resources in ground station (GS) between projects and repairs must be taken into consideration. This reduces the amount of data that can be accessed from the theoretical value. Additionally, there is a loss of 10%-15% of data for every set 50 packets of data downlinked. Each set must be downlinked 2 to 3 times to complete the set. Completing the packets is important when downlinking image data as missing data does not allow the images to be fully constructed. In combination of all these factors, BIRDS-3 completes one image every 2-3 days.

The downlink rate must improve to increase the amount of downlinked data. Higher downlink rate consumes larger power. BIRDS-3 uses amateur UHF with an uplink/downlink bandwidth of 4800bps. During downlink, the communication subsystem (COM) transmits at 0.8W. Given that COM has transmitting efficiency of 17.5%, the COM consumes 4.56W. This calculates to 0.95mJ (equivalent to 26x10^{-5}mWh) per bit for 4800bps downlink. Table 1 summarizes the COM parameters for all BIRDS-3 satellites.

Table 1. BIRDS-3 power (PWR) and COM characteristics

Parameters	**Details**
Energy Generation per orbit	1480mWh
Transmission PWR	0.8W
Supply Voltage	3.8V
Transmission Efficiency	17.5%
Over Current Protection	4A

BIRDS-3 has NiMH batteries placed at three parallel, two series (3P2S) and has a total capacity of 3800mAh rated at 3.8V. The solar panel can generate an average of 1480mWh per orbit. Since the capacity is more, the limitation is on the power generation. Not all generated power is utilized as loses can be expected while supplying. With 26x10^{-5}mWh per bit, 1480mWh can downlink 5.6Mbit or 700kB per orbit at maximum. In a day, the theoretical limit of the amount of data downlink is equivalent to 10MB assuming continuous communication with GS, full usage of the packet (i.e. no header and footer) and 100% power usage for communication. The maximum amount, therefore, is probably 1MB or less. Practically though, BIRDS-3 downlinks 25-30kB of real data per day per satellite using the GSN. This is far less than the theoretical limit.

1.4 Exploring Solutions and their Challenges

There are two ways to tackle the bottleneck for 1U CubeSats. The quantity or the quality of the data can be increased. To increase quantity of data, one straight forward solution is to increase the bitrate. Table 2 shows ideal power calculations and total communication time for an orbit if BIRDS-3 satellites were to increase the bitrate using commercially available COM. The calculations are based on EnduroSat's UHF [18], S-band [19] and X-band [20] COM modules. EnduroSat is used for reference. Table 2 shows that, in theory, it is still possible to increase bitrate up to 20 Mbps if S-band is implemented. Goliat [21] has shown that it is feasible for a 1U CubeSat to transmit in S-band.

Table 2. Power consumption and ideal data downlink per orbit on BIRDS-3 bus

Frequency	**Bitrate**	**PWR consumption (W)**	**Energy/bit (mWh)**	**Ideal data/orbit**
UHF (BIRDS-3)	4.8kbps	4.56	0.00026	700kB
UHF	19.2kbps	11.4	0.00017	1.12MB
S-band	20 Mbps	11.4	1.6x10^{-7}	1.17GB
X-band	120 Mbps	68.6	1.6x10^{-7}	1.17GB

While placing a higher bitrate and frequency COM board into the bus system is an option, it is not as straight forward. The BIRDS satellites should have better attitude control for S-band patch antenna pointing. License needs to be applied where necessary and may extend the project time. Rigorous testing on both anechoic chamber, space environment and long-range testing (LRT) need to be additional done to accommodate BUS design changes. These could elongate project time and subsequent costs.

Instead of changing the BUS system, the focus can shift to GS. Increase in GS increases ground visibility time for satellites and therefore, increases data that can be accessed. The BIRDS Ground Station Network (BGSN) has GS in fourteen countries spread around Asia, Africa and Americas. Through the Cross-Boarder Collaborative Satellite Operation Demonstration (CCSOD), image data has been downlinked from different parts to support the operations done from a single GS at Kyutech.

CCSSOD is ongoing and the process for data downlink collaboration is improving. Fig. 5 shows some of the results from CCSSOD. However, there are significant challenges that remain. Not all GS are operational as some have technical issue and some have lack resources. Systems and processes still need to be improved to communicate which packets need to be downlinked, when and where that data should be stored. At the time of writing, Kyutech GS is the only GS that operates daily.

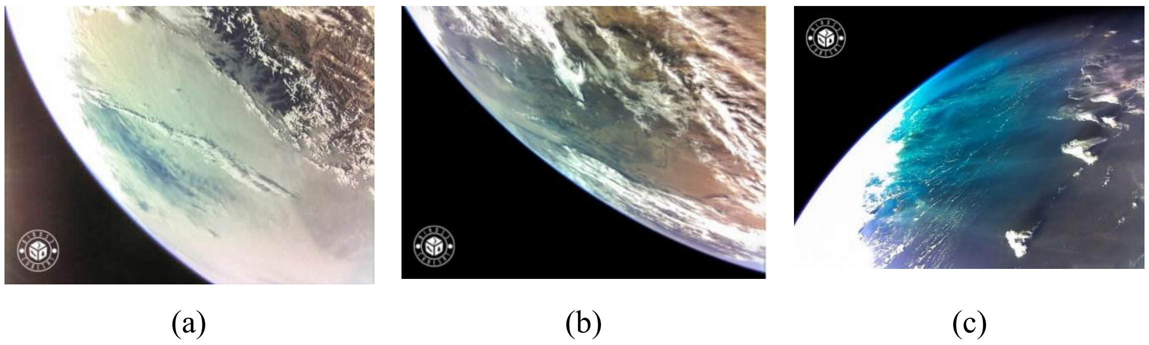


Figure 5. BIRDS-3 image data support from a) Bhutan b) Mongolia and c) Ghana GS

Image processing by data compression is a proven tool that reduces the size of the onboard data. BIRDS-3 uses JPEG compression algorithm to convert an eight-bit RAW RGB image. A 640x480 RAW RGB has 307.2kB of data. Depending on what the scene the image has, JPEG reduces to 7kB to 74kB. That is 2.3% to 24.1% of the original image which is a significant reduction in size.

Instead of increasing the data quantity, a better approach is to focus on the data quality. BIRDS-3 satellite has active attitude stabilization but no pointing. Currently, the team is unaware of which direction the satellite is pointing at when image command is sent. Taking image of the earth is left to chance. Observing this, BIRDS-4 team have implemented active pointing on their 1U system. The satellites are scheduled to launch in 2020.

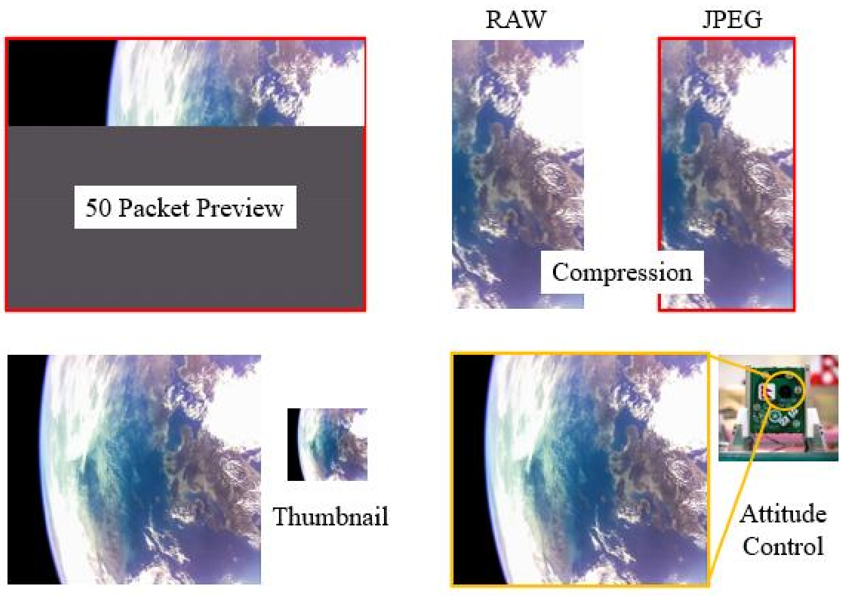


Figure 6. Processes to decreases data size and increase quality of downlinked image

Data can be classified “good” or “bad” even before the image is completed. BIRDS-3 downlinks data in set of 50 packets. The GS operator observes the first set and makes an educated guess as to what direction image is facing towards. If the part-reconstructed image is earth-facing, the downlink is proceeded to the next set. If not, the operator begins with the next image. This technique has shown to be effective when combined with burst or semi-burst imaging mode. Command is sent to take multiple images in short period of time. The GS operator then downlinks the first set and checks. Empirically, at least one image is shown to be “good” out of five burst image commands.
Thumbnails provide an alternative to first-set downlink approach. A thumbnail image is generated from each image. The thumbnail is easier to downlink and provides complete information on the image. The GS operator can then decide whether the image is “good” and proceed. Otherwise the operator can skip to the next image. The methods are illustrated in Fig. 6.
While these methods improve quality of image, they are also incredibly manual. The images need to downlinked first and observed, before proceeding to downlink further. Intervention from the GS operator reduces eventual data waste, however, consumes time and effort. A better approach would be to automatically select what is “good” and what is “bad” onboard the CubeSat. That way, the operator can simply downlink the data without having to make any decisions from ground.

1.5 Machine Learning (ML) Solution

The solution thus shifts towards automation through software implementation. Classification of an image takes place in two steps. Firstly, features must be identified. Each object in the image has a unique feature. The feature is extracted through a feature extractor. The algorithm then understands what specific features to look for. After that, the image is classified. The portion of the algorithm that classifies the image is called the image classifier. While this thesis is limited to classifying the whole image, objects in an image can be independently classified as well. This is called segmentation. Each object in the image is classified with a boundary. The portion of the algorithm that classifies the image is called the classifier.
In classical image processing, a programmer would hard-code both the feature and extraction algorithm. This strongly depended on the general understanding of the image, depth in knowledge of the programmer and the quality of the code to create a general enough model that could classify images. However, with the advancement of Machine Learning (ML) techniques, the hard-coded feature learning and extraction is modelled by the learning model instead. This creates a more general model and has shown to improve classifications.
ML is a type of Artificial Intelligence (AI) where computers are trained on input data to create models that represent an output as accurately as possible through statistical analysis. As stated before, this is different from traditional fields of computer science where models are programmed explicitly through a set of instructions written by a programmer. In ML, the models are trained instead.

Training is a process in which a range of input data is shown. The model learns features and classification information allowing it to statistically predict on new set of data. Image Classification (IC) is one of the most common prediction tasks in ML.

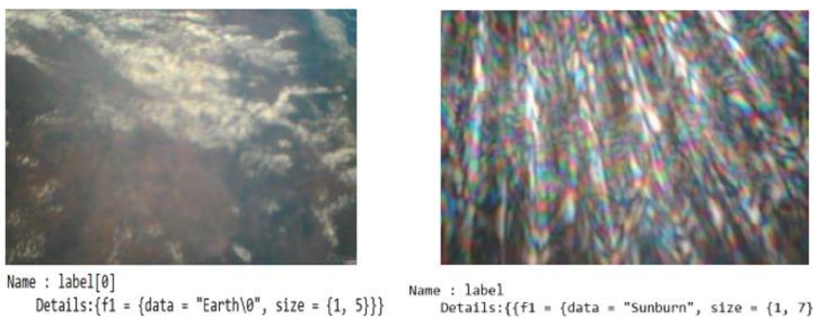
A ML model can carry out IC by unsupervised and supervised learning. Unsupervised learning is a learning method where a model extracts information from unlabeled input data during training. The advantage is that training takes less time and is simpler to deploy but takes time to create appropriate dataset. Supervised learning is a learning method where the model extracts information from labelled input data. Labels assign the images into categories. The trained model predicts new images as labelled output.

1.5.1 ML Solution for Image Classification (IC)

There are a couple of ML techniques for IC that can be effective; Random Forest (RF), Support Vector Machines (SVM), and a special type of Neural Network (NN) called Convolutional Neural Network (CNN). RF and SVM have the advantage of using less computational power, smaller size, and training dataset. RF is used while doing multiclass classification and is even less computationally complex than SVM. RF and SVM are ideal for constraints imposed by a 1U CubeSat.

On-orbit demonstration of RF along with SVM has been conducted on-board IPEX 1U CubeSat [22]. IPEX's RF classifier ran on AT91SAM9 and could segment a 3MP image into cloudy, clear, planetary limb or outer space [23]. The paper states that it is the first case of RF implementation in space. Such on-board image analysis in space would prioritize quality data to be downlinked. Quality can be controlled by stating predefined rules for downlink. For instance, an image which has the greatest percentage classified as clear can be downlinked first.

Kyushu Institute of Technology's BIRDS-4 has completed their Flight Model (FM) and is awaiting launch. Many missions have been implemented including a 5MP OV5642 ArduCam imaging module to take images from space. One of the improvements from previous BIRDS project is that it has Image Classification Unit (ICU) mission in place. The mission uses a trained SVM model to classify images taken onboard into earth, space, and sunburn. The SVM model runs on a STM32F29VI MCU and classifies images of 320x240 (QQVGA). The model is 75% accurate. Fig. 7 shows ICU classifying earth and sunburn correctly.



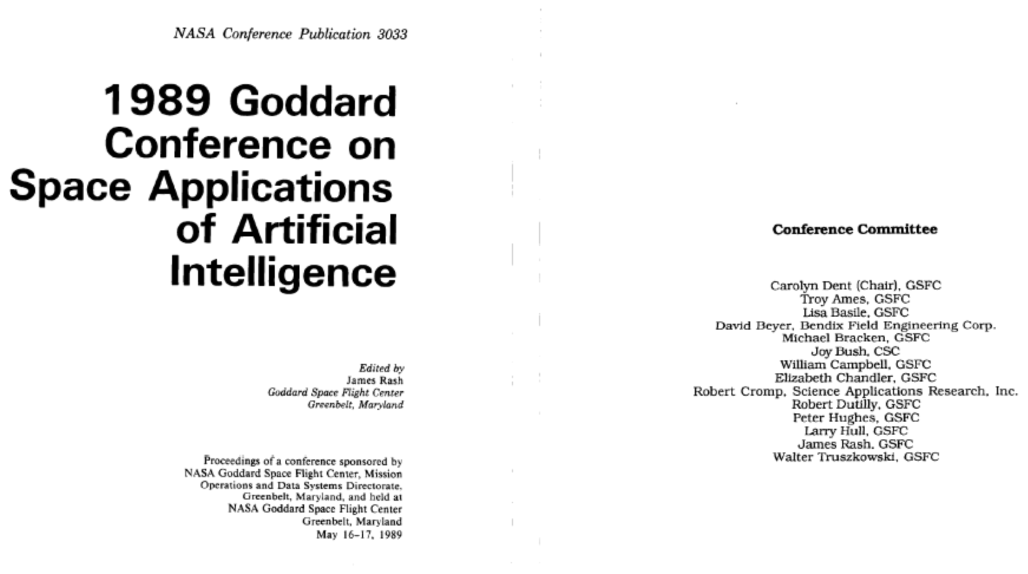
```
Name : label[0]
    Details:{f1 = {data = "Earth\0", size = {1, 5}}}
```

```
Name : label
    Details:{{f1 = {data = "Sunburn", size = {1, 7}
```

Figure 7. BIRDS-4's Image Classification Unit classifies satellite images

1.5.2 Neural Networks (NN) and Space

NN are built upon building blocks of neurons which takes in input and produces an output based on the mathematical function that defines the neuron. A NN is created through the interconnection between layers of neurons where the input is the input layer, inner layers are the hidden layer and the desired prediction exits from the output layer. The network is trained by a training dataset that is fed into the network and based on the output, the function in the neurons are recalibrated to minimize the loss. NN has applications in broad range of fields including robotics, social media, medicine, finance, marketing and now space.



NASA Conference Publication 3033

1989 Goddard Conference on Space Applications of Artificial Intelligence

Edited by
James Rash
Goddard Space Flight Center
Greenbelt, Maryland

Proceedings of a conference sponsored by NASA Goddard Space Flight Center, Mission Operations and Data Systems Directorate, Greenbelt, Maryland, and held at NASA Goddard Space Flight Center Greenbelt, Maryland May 16–17, 1989

Conference Committee

Carolyn Dent (Chair), GSFC
Troy Ames, GSFC
Lisa Basile, GSFC
David Beyer, Bendix Field Engineering Corp.
Michael Bracken, GSFC
Joy Bush, CSC
William Campbell, GSFC
Elizabeth Chandler, GSFC
Robert Cromp, Science Applications Research, Inc.
Robert Dutilly, GSFC
Peter Hughes, GSFC
Larry Hull, GSFC
James Rash, GSFC
Walter Truszkowski, GSFC

Figure 8. Proceedings published in 1989 by NASA's Goddard Space Flight Center [24]

The idea of using NN for space applications is not new. The proceedings from 1989 Goddard Conference on Space Application of Artificial Intelligence, shown in Fig. 8 has papers where use of NN has been conceptualized and in some cases implemented [24]. In the proceedings, Gaspin outlines the use of NN for a hybrid approach to mission scheduling. The paper presents OSCAR, a hybrid automated intelligent reasoning system, built for tackling such problems. In the proceedings, Bouret and Reggia present a method for satellite failure diagnosis by using two layers of NN. The paper states that the model will become better when the NN size becomes larger. In the proceedings, Campbell et al. explores the use of NN to categorize undefined objects and generate satellite imagery database to high-level data objects. The paper stresses the importance of NN as the data rate of Earth Observation (EO) missions increase over time.

While NN showed promise, researchers had difficulty working with NN as "the time and effort required to develop neural network architectures and training is very high" [25]. That has changed with the recent advances in parallel computing, high-level open source ML libraries and big data through plethora of interconnected devices. NN is now being applied for nanosatellite interplanetary autonomy [26] and attitude control [27].

1.5.3 State of Art: Convolutional Neural Network (CNN)

CNN is a subclass of NN where one or more convolutional layers are present besides the fully connected layer (FCL). CNN was first presented in 1988 by Fukushima [28] and later improved by LeCunn et al. [29] in the late 90s by training a CNN model on the MNIST handwritten dataset shown in Fig. 9. The way CNN interprets 2D and 3D images is thought to be like how our brain processes images and is therefore, optimized for problems regarding feature recognition and image processing. CNN's performance improved over the years, but it was only after the 2012's annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) that researchers began to have a strong renewed interest in CNN.

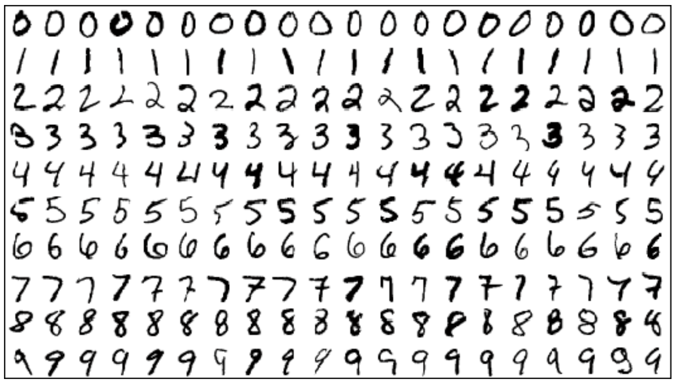


Figure 9. MNIST handwritten dataset designed for letter recognition [29]

Papers comparing SVM to CNN on classification problems in different fields show that CNN has performed better in the field of leaf identification [30] shown in Fig. 10, lung cancer classification [31], facial recognition [32] and ship detection in satellite imagery [33]. While traditional ML algorithms such as SVM rely on hand crafted feature extraction, CNN's convolution layers (CL) automatically extracts features during training and creates an optimized feature map for that problem.

Figure 10. Hedjazi et al. [30] showed that CNN had better results to SVM in leaf identification.

In the field of remote sensing, feature extraction from hyperspectral images shows "supervised techniques provide better accuracy than their unsupervised techniques" [34]. CNN is the most popular algorithm in deep learning for remote sensing [35][36]. Other algorithms include supervised and unsupervised methods such as Autoecoders (AE), Deep Belief Network (DBN), Recursive

Neural Network (RNN) and Generative Adversarial Networks (GAN) [35]. CNN has higher overall accuracy and kappa values for the University of Pavia dataset as compared to Deep Belief Network (DBN), Autoencoder (AE) and Residual Net (ResNET) [37]. Example of classification and segmentation is shown in Fig 11. The downside of CNN is that the training requires a very large dataset, is computationally expensive and has larger memory footprint.

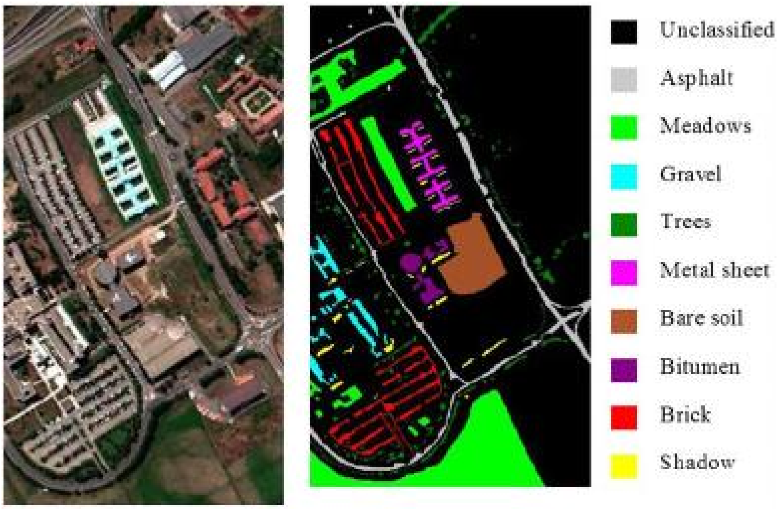


Figure 11. Classification example on University of Pavia remote sensed dataset [38]

1.5.4 CNN Image Application for Small Satellites

These challenges have not prevented researchers to explore the possibility of using CNN on small satellites. Buonaiuto et al. [39] presented a paper to create an intelligent CubeSat system that uses Nvidia Tegra X1 (TX1) for on-orbit CubeSat recognition, classification and segmentation as shown in Fig. 12. The paper demonstrated the potential to use GPU to process images by showing ground-based test results on an original dataset. The paper aimed to explore the use of TX1 as a primary instrument aboard a CubeSat. The challenges described on the paper are 1) CubeSat's fast movement makes it difficult to track 2) larger database needed to improve accuracy.

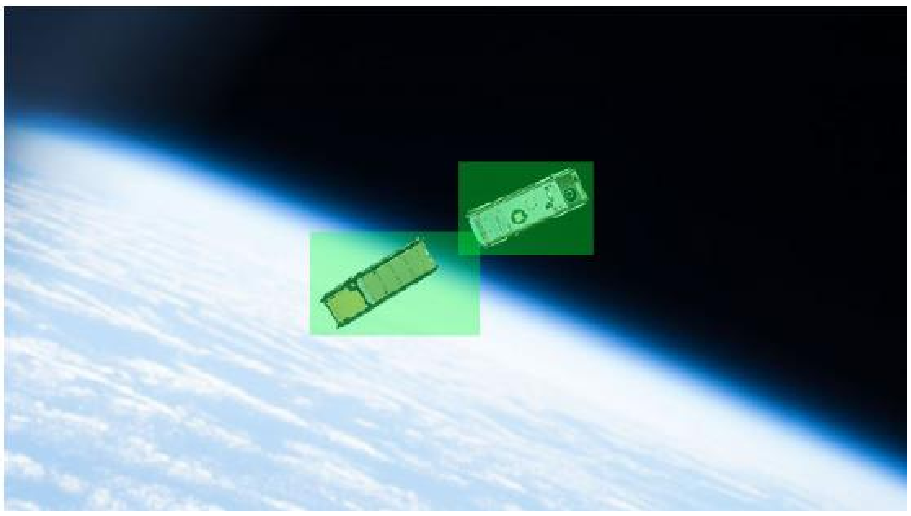


Figure 12. Buonaiuto el al.[39] tested CNN to identify CubeSats that are in orbit

Arechiga et al. in [40] used a mix of traditional machine vision techniques and CNN to identify ships in an image on Planet Inc.'s open California RGB dataset of 3m Ground Sampling Distance (GSD). The results are shown in Fig. 13. The CNN architecture is implemented on Nvidia Jetson TX2. The model is trained using high level Keras API to access Tensorflow library. The CNN is inspired by the design of VGGNet. The paper aimed at demonstrating the use of TX2 as the authors believe the System on Chip (SoC) fulfils the Size, Weight and Power (SWAP) requirements of a CubeSat.

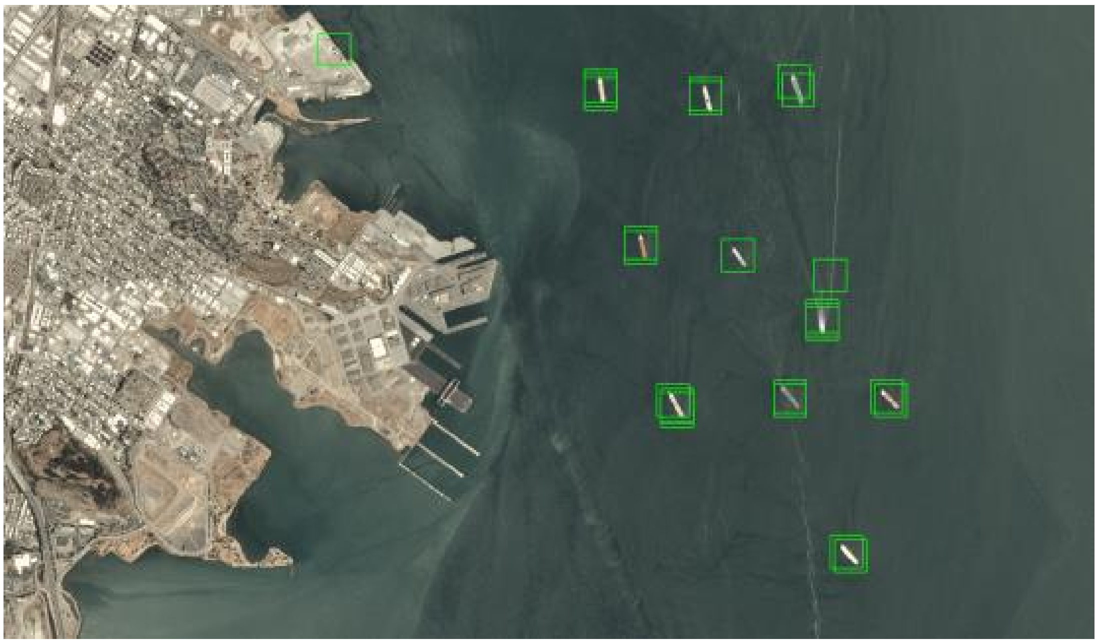


Figure 13. Arechiga et al.[40] used CNN to identify ships on Planet data.

Manning et al. [41] transfer trained four CNN models and used it on Xilinix Zynq7020, a FPGA with flight heritage. In the paper, MobileNet V1, MobileNet V2, Inception-ResNet V2 and NASNet Mobile CNN are selected for testing. Accuracy, execution time and run time are used for benchmarking. Original dataset of 8000 images, each of 489x400 resolution, are acquired from STP-HS/CSP aboard the International Space Station (ISS). Some examples are shown in Fig 14. The dataset is used to train the models. The paper selects MobileNet V1 and states that Tensorflow Lite is an appropriate tool for deploying CNN models on existing hardware. The paper further claims such hardware could be deployed on small satellite platform such as CubeSats.

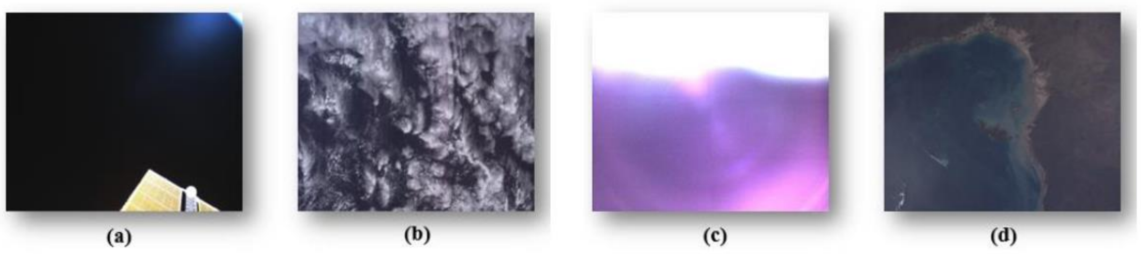


Figure 14. Manning et al. [41] created an original database of 8000 images taken from ISS
The images were classified as a) black b) cloud/water c) distorted d) land e) white (not shown).

Greenland et al. [42] used the same Zynq7020 FPGA to classify clouds for small satellite on-board applications as shown in Fig 15. In the paper, CNN model is created by transfer learning of VGG19 architecture. Machine learning libraries of Keras and Caffe are utilized through Python to train. LandSat-7 images are used for the purpose of demonstration. Proof of concept is shown by system in a loop simulation. A python based modular system acts as the simulator where images are fed into the FPGA and results are obtained. The paper states an accuracy of 98% for cloud detection.

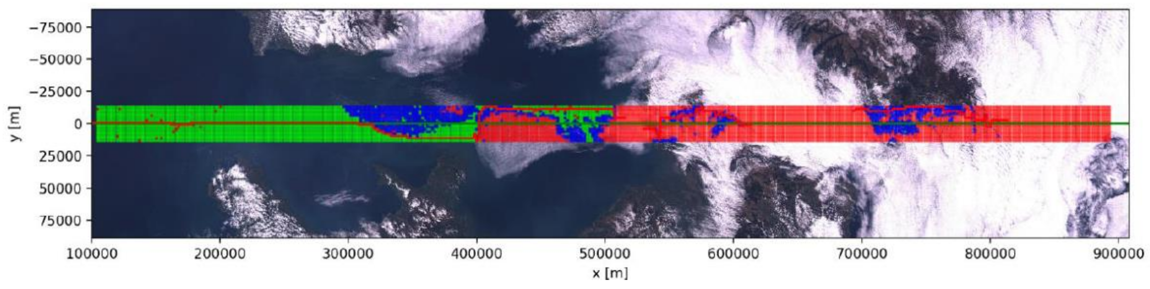


Figure 15. Greenland et al.[42] implementation on LandSat-7 data

Monirul Islam et al. [43] proposed to use "onboard deep neural computing and machine learning model to analyse and process multispectral images" for a 3U CubeSat. The authors do not explicitly state what type of CNN had been implemented but write that "Intel Movidius Neural Compute Stick seems like a good choice for early trials" because "it features Myriad 2 Visual Processing Unit (VPU)." The paper designed a payload with two cameras; a hyperspectral (VNIR) camera and LWIR thermal camera for capturing spectral images. The paper planned on deploying the CNN on the payload.

Braun [44], in his thesis, investigated ML hardware that can run CNN models for CubeSats. The thesis states that Nvidia Jetson TX2 and Movidius Neural Compute Stick are being used for ground based AI applications. Since CubeSats use Commercial off-the-shelf (COTS) for space and follow embedded system development that happens on the ground, his thesis argues that deploying Deep NN such as CNN can be possible in aforementioned Embedded AI hardware.

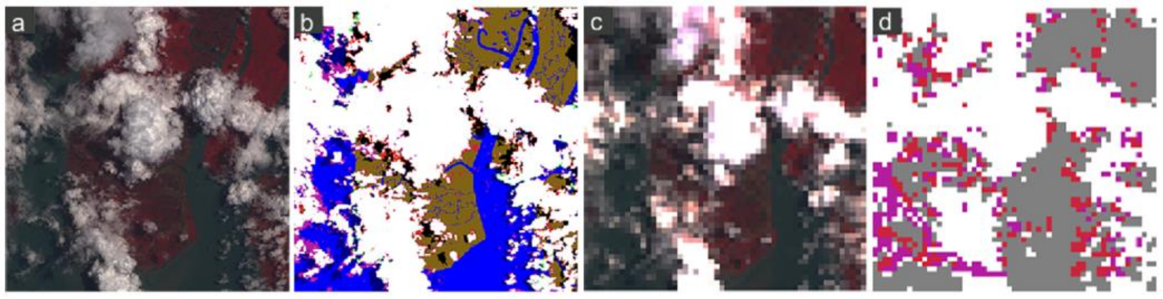


Figure 16. Zhang et al. [45] presented about CNN that could segment cloud as shown in d. Red and pink are misclassifications. a) is the image taken by LandSat-8 b) ground truth c) compressed image d) CNN segmented image.

Zhang et al. [45] used CNN model deployed on ARM9 MCU to detect and segment cloud for CubeSat applications. In the paper, U-Net, MobU-Net, Deconv-Net and MobDecov-Net are used to create four different transfer learned CNN. Publicly available LandSat-8 data is used to both train and test the model. The paper recommends using MobU-Net as the CNN showed highest accuracy. Cloud classification by the CNN is shown in Fig. 16. Similarly, Giuffrida et al. [46] proposed CloudScout CNN deployed on a Myriad 2 Visual VPU for on-board cloud detection on hyperspectral images. The payload is called HyperScout-2 and is designed for Φ-SAT-2 challenge from European Space Agency (ESA).

1.5.5 Limitations of Current Research

In reviewing the papers that has experimented and implemented CNN on hardware that can be placed on CubeSats, several limitations can be observed. Most papers have either implemented their model on FPGA, Nvidia's TX1/TX2 or dedicated VPU hardware. Only Zhang et al. have implemented their model on an MCU. MCUs consume a fraction of power compared to FPGA, TX1/TX2 or VPUs and are ideal for 1U CubeSats. The argument is that if a model can be deployed on a 1U CubeSat, it can be implemented on any bigger satellite.

Only Manning et al. had a dedicated image database of 8000 images to train CNN. Other papers relied on data from Planet or LandSat. CNN's accuracy scales with larger dataset, therefore a bigger dataset is required to achieve higher accuracy. As stated in the paper, Manning et al. have accepted that 8000 images are not enough for training a good CNN model. The database is also not open which prohibits other researchers to use and expand the database.

All papers have resorted in using transfer learning instead of building a lean CNN architecture that ensures small size and better performance. CNN designs are small such as MobileNet are popular. The CNN is customized to run on mobile platforms. However, it is not small enough to be able to fit inside an MCU.

Furthermore, the CubeSat community has yet to explore deep learning models used frequently for IC in remote sensed data. While CNN is still widely popular technique, supervised and unsupervised techniques such as AE, DBN, RNN, GAN and ResNET are gaining traction. Therefore, comparative results between different deep learning models have not been explored yet.

1.5.6 Dedicated CNN for CubeSat

This thesis is intended to address these limitations. Firstly, since no dedicated dataset for CubeSat is available. A large enough dataset is created for ML training. There is no dedicated CNN designed for CubeSats. SWAP requirements impose severe restrictions on the size and power consumption. A CNN small enough to fit in an MCU but high enough accuracy and F1 score to classify images is designed from scratch and documented. CNN is then compared to SVM, AE and DBN. CNN shows highest accuracy and F1 score used on dedicated test dataset created from images from BIRDS-3.

2 Convolutional Neural Networks

CNN dominant rise in the past decade as the one the most explored, implemented, and practical algorithms in the field of ML is a testament to years of thoughtful design, iterative improvements, and eventual explosion of big data. This chapter explores CNN's history and rise, dissects the design process, and explores how models are trained and optimized.

2.1 History in Brief: ANN to DNN to CNN

McCulloch and Pitts paper [47] in 1943 is considered as one of the first recorded document that attempted to model neural networks in brain. The paper had major influence when in 1945, John von Neuman used the neural model in discussions to design future of computers [48]. In 1958, psychologist Rosenblatt's paper [49] put forth the concept of perceptron. Perceptron is a mathematical model that replicates how a neuron in brain behaves. Perceptron is the basis for modern ANN. The concept was inspired by McCulloch-Water's paper. The limitations of the model, however, were highlighted by Minsky and Papert in 1969 in their book [50]. The book outlined that perceptron model could only solve linear separable problems but struggled in XOR problem [51].

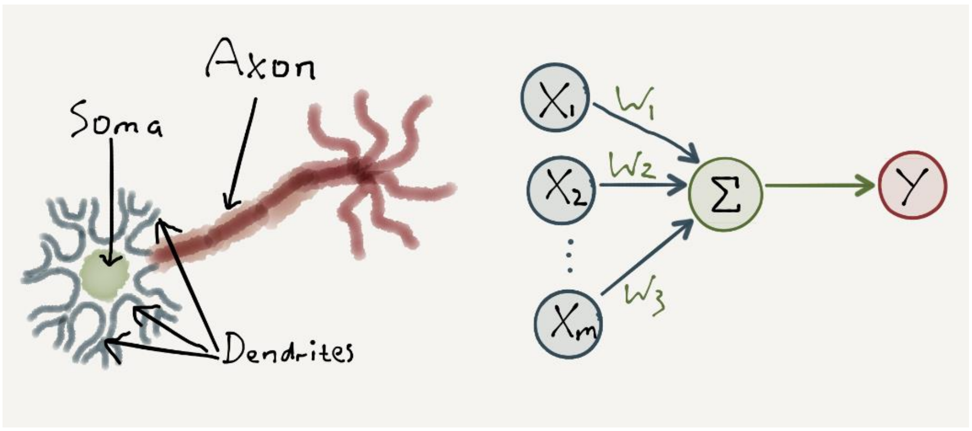


Figure 17. A perceptron is based on biological neurons. Structuring is similar [52]

By 1980s, the limitations posed by a single positron (shown in Fig. 17) were solved by combining two or more perceptrons. Placing two or more perceptrons developed the concept of Multi-later Perceptron (MLP) where layers of perceptron would be stacked on top of another to develop a model. Information is fed into the model and an outcome is predicted in the output layer of perceptron. This is known as the feed-forward technique. To improve predictions, Rumelhart [53] published a paper in Nature in 1986 introduced and derived backpropagation. Some literature state that Monro and Sutton[54] first introduced backpropagation as stochastic approximation method in 1951. Werbos specified backpropagation for NN in his dissertation in 1974 [55]. However, Rumelhart is credited to implement it on NN. Yann LeCun, who currently leads Facebook's AI division [56], proposed an alternative derivation to backpropagation in 1986 [57]. Feedforward and backpropagation methods allowed MLP models to calibrate internal parameters and improve the model. The process of calibration is called training. In 1988, Fukushima [28] took a step further and introduced hierarchical network called Neocognitron. The hierarchical network is based on work done by Hubel and Wiesel's [58] findings on visual cortex. Neocognitron later became the basis for modern Deep Neural Network (DNN), more specifically CNN.

The 1990s saw ANN take a back seat with SVM proposed by Boser et al.[59] driving ML technology forward. However, the emergence of DNN through CNN proposed by Yan LeCun et al. in their paper [29] in 1998 revived interest in ANN as an active ML research field. The paper brought forth the idea of scaling the hidden layers. In 2000s, researchers rebranded ANN to DNN. Hinton et al. published a breakthrough paper on Deep Belief Network (DBN) in 2006 [60] which showed tangible evidence that training NN is not as expensive as previously assumed. The paper created enough excitement for researchers to focus on DNN. This paved the way for rise of CNN in the next decade. In hindsight, the development of DNN was ahead of its time. The exponential growth through innovation in silicon technology put millions of handheld devices on people's hands by late 2000s. Access to internet, sharing of data and processing power on CPUs and GPUs became more affordable due to economies of scale. DNN, most notably CNN, leveraged sudden influx of data by labelling them and using GPUs to train deeper networks. Fig. 18 shows a Google Trends search on CNN. Y-axis is normalized axis while X is the time. A steady rise in search is seen after 2012.

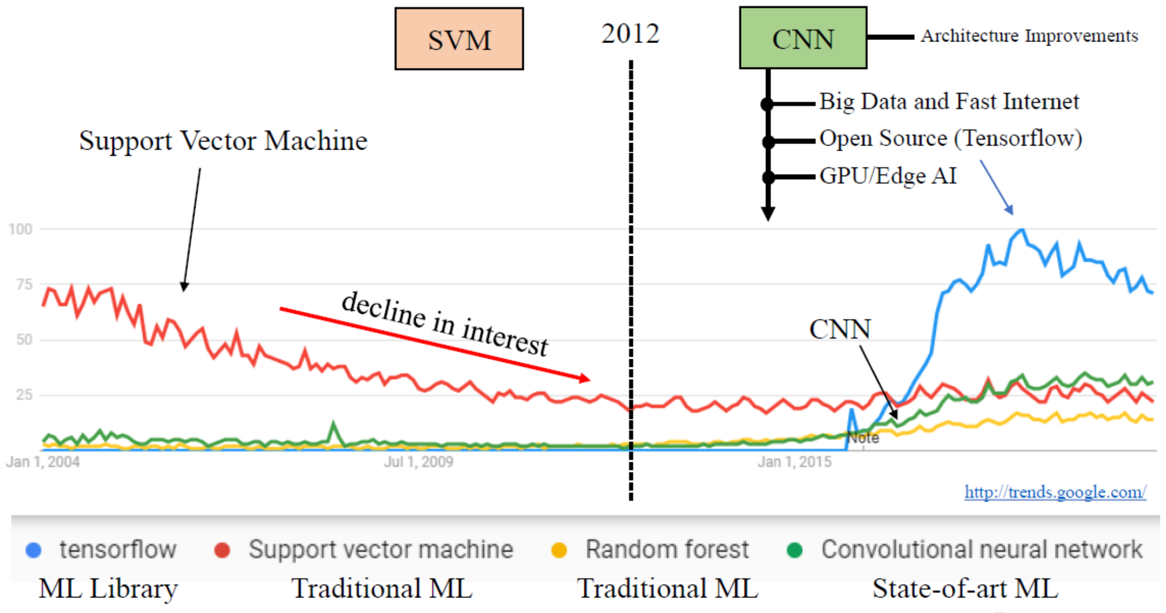


Figure 18. Google Trends [61] shows a steady rise in interest in CNN (green) after 2012

At ImageNet Large Scale Visual Recognition Competition (ILSVRC) 2012, Alex Krizhevsky et al. proposed a CNN model called AlexNet that achieved state-of-art results in object recognition and classification tasks. AlexNet received a top 5 error of 15.3% while the second received 26.2% error [62]. With such huge difference in performance, AlexNet stole the limelight which led to rapid interest and subsequent improvements in CNN. CNN has won every ILSVRC since. ILSVRC ended in 2017. After 2017, the competition moved on to classifying 3D objects using natural language [63].

2.2 CNN Structure

There are no specified rules that govern the way CNN is designed. However, modern CNN have a common structure. These networks are composed of convolutional layers together with pooling layers (CL) first and fully connected layers (FCL) after. More recently, global average pooling layers (GAPL) are used to reduce the number of parameters instead of FCL. In a 2D classification problem that the thesis is attempting to solve, 2D image feature extraction takes place in the CL and PL while the classification takes place at the FCL or GAPL. The basis for DNN is a perceptron. Stacking and interconnecting perceptrons make up multi-layer perceptron (MLP). As more layers are stacked, the design becomes deeper, increasing the depth. So forth, the name "**D**eep" is given in **D**NN. FCL is type of feedforward MLP. CL is a special type of FCL. Details are provided in this subsection.

2.2.1 Multi-Layered Perceptron

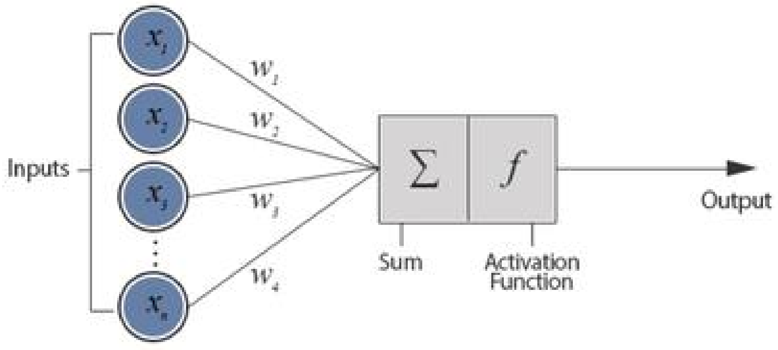


Figure 19. Model of a perceptron which forms the basis for DNN [64]

A perceptron is the most basic form of DNN. A perceptron is a mathematical representation of a neuron. In ML, perceptron, neuron or nodes are used interchangeably. Fig. 19 breaks down a perceptron into functional mathematical blocks. A perceptron maps a set of inputs $\boldsymbol{x}^l$ to $\boldsymbol{y}^L$. This means that a perceptron is a simple function approximator that provides some desired output based on inputs. The process firstly involves input x_j multiplied to some corresponding weight w_j. Weights w are a set of parameters that changes the behavior of the function. The resulting weighted input is summed and bias b is added for affine transformation. Until this stage, the function is linear. The sum is placed in an activation function $f(x)$ such as ReLU or sigmoid (details in Section 2.3.1.1). $f(x)$ brings non-linearity to the equation and is responsible to activate the perceptron. The output $y(w, x)$ can be represented mathematically as Eq. 2.1 and simplified to Eq. 2.2 taking $b = w_0$.

$$y(w,x) = f\left(\sum_{j=1}^{n} w_j\, x_j + b\right) \qquad (2.1)$$

$$y(w,x) = f\left(\sum_{j=0}^{n} w_j\, x_j\right) \qquad (2.2)$$

Eq. 2.2 shows a basic example of a process called feedforward. Inputs move from left to right to provide an output. The inputs are “fed” and “forwarded” through the perceptron. A perceptron helps to estimate simple functions, however, complex function require combination of perceptrons. A multi-layered implementation of a perceptron with input, hidden and output layer is called a multi-layered perceptron (MLP).

In MLP, inputs go through the input layer. Each node (perceptron) of a layer is connected to all the nodes of the next layer. Each node is staked to form multi-layered structure where the output of one layer is the input to the other. A web of connections is thus formed. The connection repeats until the final layer where desired outputs exit.

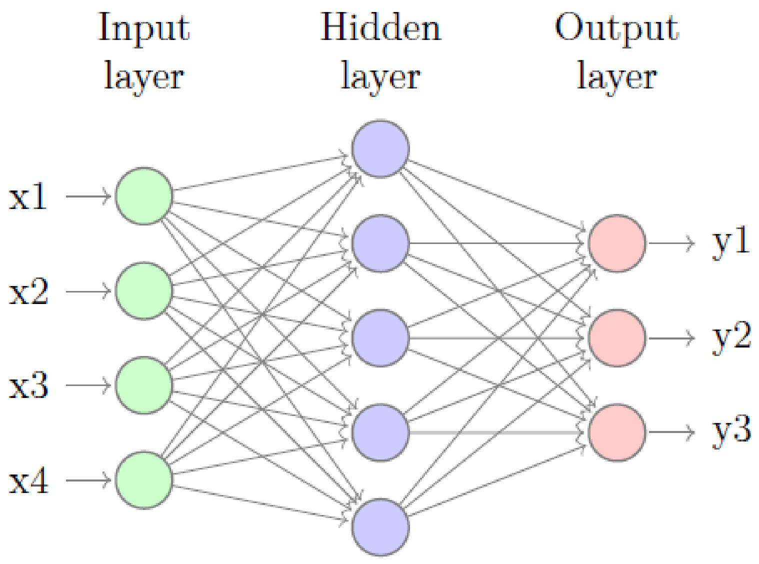


Figure 20 Nodes (perceptrons) form layers to create a multi-layered perceptron (MLP) [65]

Fig. 20 shows a simple four-input ($x_1, \dots, x_4$) feedforward-MLP with one hidden layer and three outputs ($y_1, \dots, y_3$). In l_i^2 the superscript 2 represents the layer number. l_i^1 is the input layer and l_i^2 is the hidden layer. The output layer is simply L in the superscript. The subscript i represents the node in layer l. Using the concept from Eq. 2.2, the equation can be expanded for MLP shown in Fig. 20. The equation is given in Eq. 2.3 and Eq. 2.4.

Input layer $$x_j$$

Hidden layer $$l_i^2 = f^2\left(\sum_{j=1}^{4} w_{ij}^2 x_j + b_i^l\right) \tag{2.3}$$

Output layer $$y_i = f^L\left(\sum_{j=1}^{5} w_{ij}^L l_j^2 + b_i^L\right) \tag{2.4}$$

In Eq. 2.4, f^L is an output function (details in Section 2.3.1.1 along with activation functions) instead of an activation function. f^L can be a softmax which will give probability for each y_i output. This way, MLP maps each input x_j to an output y_i. The quality of the mapping depends on each layer l's weights w. The challenge now will be to find the weights as accurately as possible so that approximated function maps input x_j as accurately as possible to output y_i. This process of finding appropriate weights w is called training (details in Section 2.3).

2.2.2 Convolution Layer

Input of the first CL is an RGB image. An RGB image has three channels thus making the input a 3D tensor. Given the height of the image is H, width of the image as W and depth or channel as D, the 3D tensor is represented by input $\boldsymbol{x}^1 \in \mathbb{R}^{H^1 \times W^1 \times D^1}$ where the superscript for $\boldsymbol{x}$ represents first CL $l = 1$. The input of one layer is the output of the other. Generalizing this, inputs for each l CL will be $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$. Correspondingly, location of a specific input in $\boldsymbol{x}^l$ will be shown by indexed set by i^l, j^l, d^l.

In one CL, three operations occur. Firstly, the general input $\boldsymbol{x}^l$ undergoes feature extraction through kernels $\boldsymbol{k}^l$. The kernel $\boldsymbol{k}^l$ has a predetermined height H^k and own width W^k. For a 3D tensor input $\boldsymbol{x}^l$, kernel $\boldsymbol{k}^l$ is a 4D tensor as the operation also takes place on D^l and multiple features D are extracted in a single layer. Therefore, the kernel is $\boldsymbol{k}^l \in \mathbb{R}^{H^k \times W^k \times D^l \times D}$ for any given layer l.

A kernel $\boldsymbol{k}^l$ convolves part of the $\boldsymbol{x}^l$ and shifts until all spatial location is covered. The amount kernel $\boldsymbol{k}^l$ shifts is called stride s. For $\boldsymbol{k}^l$ to go through every spatial location without skipping, the stride s is set at 1. In such circumstance, the convolution result will have the dimension given by Eq. 2.6.

$$(H^l - H^k + 1) \times (W^l - W^k + 1) \times D \tag{2.6}$$

If the stride s =1, Eq. 2.6 shows that the convolution operation reduces the dimension. Each $\boldsymbol{x}^l$ can be padded if spatial dimension is to be maintained. The padding increases H^l and W^l in such a way that $H^l = H^{l+1}$ and $W^l = W^{l+1}$. In the framework that this thesis uses, such padding is called

"SAME" padding (zero-padding) as the input and output spatial dimensions are the same. The other option is "VALID" padding (no padding) if dimension reduction is intended.
In a simple case where stride s =1 and no padding are used, the convolution can be represented as shown in Eq. 2.7 [66].

$$y_{i^{l+1},j^{l+1},d} = \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \sum_{d^l=0}^{D^{l-1}} k_{i,j,d^l,d} \times x^l_{i^{l+1}+i,j^{l+1}+j,d^l} \tag{2.7}$$

The second operation is pooling. The calculated output of Eq. 2.7 is now the input to the pooling layer (PL). To simplify and keep notations the same, let $\boldsymbol{x}^l$ be the input to the PL. The pooling operator p is a 2D tensor with $H^p \times W^p$ and moves with stride s. Operator p operates in each channel by channel independently. Operator p maps the overlapping region into a single number. In such circumstance, the output after application of pooling will have the dimension given by Eq. 2.8.

$$H^{l+1} = \frac{H^l}{H^p}, \qquad W^{l+1} = \frac{W^l}{W^p}, \qquad D^{l+1} = D^l \tag{2.8}$$

pooling can either be average pooling or maximum (max) pooling. In average pooling, the values in the overlapping region are averaged into a single number. In max pooling, the maximum value in the overlapping region is selected as the single number. A common PL operator p is max pooling and has the following properties as shown in Eq. 2.9 and Eq. 2.10 [66].

$$H^p = 2, \qquad W^p = 2, \qquad s = 2 \qquad (2 \times 2\,, 2) \tag{2.9}$$

$$y_{i^{l+1},j^{l+1},d} = \max x^l_{i^{l+1} \times H^p+i,j^{l+1} \times W^p+j,d} \tag{2.10}$$

The final operation in a CL is the application of activation function $f(x)$ that increases non-linearity. ReLU, softmax or sigmoid can be applied as $f(x)$ (details in Section 3.2.1.1). The result of the operation is the actual output from the CL and will be the input for the next CL, FCL or GAPL.

2.3 Training

Training of a CNN is done in two phases: forward propagation in the first phase, error calculation and backward propagation in the second. Forward propagation in CNN works the same way as described in previous chapter (Section 2.2.1). The input data $\boldsymbol{x}^l$ is fed into the network where $\boldsymbol{y}^L$ in the output layer provided probabilistic outcome. However, during training, the outcome is fed back from the output layer towards the input in a process called backpropagation. In a supervised training that this thesis is conducting, error is calculated based on labelled targets $\boldsymbol{t}$**.** This error is used to optimize weights based on advanced stochastic gradient descent (SDG) techniques (details in Section 2.3.2).

2.3.1 Hyperparameters

The result of the training depends on the selection of "built-in" hyperparameters such as cost function and optimizers. Better hyperparameter selection and regularization leads to better weight tuning which leads to better function approximation. Detailed description of the second phase is provided in this section.

2.3.1.1 Activation Function $f(x)$

One of the most critical hyper-parameters is the activation function $f(x)$. The boundaries of objects in an image is not linear and requires non-linear equations to generalize data better. Activation function $f(x)$ adds non-linearity to the model. Activation function $f(x)$ takes in a weighted sum of input and biases, computes and decides whether a neuron should be fired on or not. CL also have activation function after pooling operation. During training, selection of activation functions $f(x)$ play an important role. Common problems in gradients documented in literature are vanishing and exploding gradients [67]. To tackle the problem, activation function must be selected properly. Common activation functions $f(x)$ have been discussed in this subsection.

1) Sigmoid Function

The Sigmoid Function is mathematically represented by Eq 2.11.

$$f(x) = \left(\frac{1}{1 + exp^{-x}}\right) \tag{2.11}$$

Sigmoid is differentiable real function, defined for real input values, with positive derivatives everywhere [67]. Sigmoid is mostly used in the output layer of CNN and is useful in producing probabilistic outputs for binary classification tasks. Sigmoid is also used in the inner layers, however, they suffer from sharp damp gradients, gradient saturation, slow convergence and non-zero centered output which does not provide correct gradients for parameter tuning during backpropagation [67]. There are variants to Sigmoid. Hard Sigmoid Function offers lesser computation cost to standard Sigmoid. Sigmoid-Weighted Linear Units is only used for reinforcement learning. Derivative of Sigmoid-Weighted Linear Units improves standard sigmoid significantly in performance.

2) Hyperbolic Tangent Function (*Tanh*)

The Hyperbolic Tangent Function, also known as the *Tanh* function, is mathematically represented in Eq 2.12.

$$f(x) = (\frac{e^x - e^{-x}}{e^x + e^{-x}}) \tag{2.12}$$

Tanh output lies between -1 and 1 and is zero-centered. *Tanh* provides better performance for multi-layer neural networks to sigmoid. However, the function still suffers from vanishing gradient problem and produces dead neurons during computation. *Tanh* is primarily used for recurrent neural networks for natural image processing and speech recognition. Hard Hyperbolic Function is variant that is computationally cheaper to standard *Tanh*. [67]

3) Softmax Function

Softmax function is mathematically represented in Eq. 2.13. Subscript *j* represents number of class.

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.13}$$

Softmax takes in a real number vector input and produces a probability distribution ranging from 0 to 1. The sum of all the probabilities will equal to 1. Classification is done by looking at the highest probability for a class. Softmax is normally placed on the output layer.

This thesis uses Softmax function in CNN's output layer.

4) Softsign Function

Softsign function is mathematically represented in Eq. 2.14.

$$f(x) = (\frac{x}{|x| + 1}) \tag{2.14}$$

Softsign is primarily used on regression and speech systems.

5) Rectified Linear Unit (ReLU)

Rectified Linear Unit (ReLU) is mathematically represented in the Eq. 2.15.

$$f(x) = \begin{cases} x_i, & x_i > 0 \\ 0, & x_i \leq 0 \end{cases} \tag{2.15}$$

ReLU is the most widely applied activation function and is the current state-of-art [67]. ReLU performs better than Sigmoid and *tanh*. ReLU is also less computationally intensive as linear elements are present. ReLU operates in a way that any value above zero is identity while any value below is equaled to zero. ReLU is applied in both output and inner layers of CNN.
This thesis uses ReLu as activation function in CL of CNN.

2.3.1.2 Cost Function $C(x)$

Keeping notations consistent, $\boldsymbol{x}^l$ is the input data, $\boldsymbol{t}$ is ground truth (labeled) information and $\boldsymbol{y}^{\boldsymbol{L}}$ is the output from the final layer (output layer). To update the weights such that the function approximation is accurate as possible, error must be calculated. In supervised learning, $\boldsymbol{t}$ provides labeled information of the training dataset. Since $\boldsymbol{y}^{\boldsymbol{L}}$ is the predicted outcome, a function has to be in place such that error can be calculated between what the model *thinks* $\boldsymbol{x}^l$ is which is $\boldsymbol{y}^{\boldsymbol{L}}$ and what the model *should actually think* as $\boldsymbol{t}$. Cost function (also called error or loss function) $C(x)$ calculates how far off the model is from approximating a given task properly. Cost function $C(x)$ is only used in training and is only applied after the activation function $f(x)$ of the final or output later.
There are two popular Cost functions $C(x)$. First is the mean squared error (MSE). Given that there are N number of training samples, the MSE $C^{MSE}(x)$ is mathematically expressed as in Eq. 2.16.

$$C^{MSE}(x_i) = \frac{1}{N}\sum_{i=1}^{N}(t_i - f(x_i))^2 \tag{2.16}$$

Second is the cross-entropy function $C^{CE}(x)$ and is mathematically expressed as

$$C^{CE}(x_i) = -\sum_{i=1}^{N} t_i \log\left(f^{smax}(x_i)\right) \tag{2.17}$$

This thesis uses Cross-Entropy Loss function $C^{CE}(x)$ after applying softmax $f^{smax}(x)$ activation in the output layer.

2.3.2 Optimizer

Cost function C finds the error between input ground truth $\boldsymbol{t}$ and final output $\boldsymbol{y}^{\boldsymbol{L}}$. C must be minimized to create a more accurate model. Optimizers are responsible to change the parameters to minimize C. Parameters are weights $\boldsymbol{w}^{\boldsymbol{l}}$ in case of FCL or GAPL and kernel $\boldsymbol{k}^{\boldsymbol{l}}$ in case of CL for any layer $\boldsymbol{l}$ (for simplicity, the parameters will all be represented by $\boldsymbol{w}^{\boldsymbol{l}}$). Optimizers help to determine the direction and amount of change $\boldsymbol{w}^{\boldsymbol{l}}$ that need undergo to find an optimized solution to a model.

Modern optimizers are an extended form of gradient descent (GD) method. In GD, the parameters $\boldsymbol{w^l}$ are updated iteratively towards the opposite direction of the gradient of C. The process continues until $\boldsymbol{w^l}$ converges to a point where C is minimized. Eq. 2.18 repeats based on the number of iterations. The speed at which GD converges is the learning rate η (details in Section 2.3.5).

$$w'^l_i = w^l_i + \eta . \left(- \frac{\partial C}{\partial w^l_i} \right) \tag{2.18}$$

A widely applied technique is the SGD where a sample is randomly chosen among total sample N. The gradient of that random sample is used to update rather than using the complete batch. This accelerates calculation and has shown to achieve optimal convergence speed [68]. SGD has a variation where a mini-batch is used. Total sample N is a single batch. N is divided and fed in mini-batches. A random sample from that mini-batch is used to calculate the gradient. Mini-batch SGD has been the standard practice when selecting SGD.

SGD has issues in learning rate η selection and with saddle points where optimization reaches local minima instead of global. Nestrov Accelerated Gradient Descent (NAG) uses speed v and momentum mtm to escape local minima. The parameters $\boldsymbol{w^l}$ are updated as shown in Eq. 2.19 [68].

$$\begin{cases} \widetilde{w} = w + v^{old} . mtm \\ v = v^{old} . mtm + \eta . \left(- \frac{\partial C}{\partial w} \right) \\ w' = \widetilde{w} + v \end{cases} \tag{2.19}$$

The challenge of learning rate η selection persisted. AdaGrad, an adaptive learning rate method, adjusts learning rate η automatically, converges faster and achieves better results to SGD. The parameters $\boldsymbol{w^l}$ are updated by AdaGrad using historical gradient V_{stp} at each iterative step stp . Eq. 2.20 shows how learning rate η and parameters $\boldsymbol{w^l}$ updates.

$$\begin{cases} V_{stp} = \sqrt{\sum_{i=1}^{stp} \left(\frac{\partial C}{\partial w} \right)^2 + \epsilon} \\ w_{stp+1} = w_{stp} - \frac{\eta}{V_{stp}} . \left(- \frac{\partial C}{\partial w} \right) \end{cases} \tag{2.20}$$

Eq. 2.20 shows that increasing training time would increase iterative step stp which would then increase historical gradient V_{stp}. If V_{stp} is large, AdaGrad can no longer update $\boldsymbol{w^l}$ because of vanishing learning rates η . RMSProp and AdaDelta improved AdaGrad. However, Adaptive

Moment Estimation (Adam) has become popular. Adam uses adaptive learning and momentum. Adam improves on RMSProp and AdaDelta by introducing exponential decaying average of past gradients m_{stp}. Eq 2.21-2.24 shows Adam's step by step process.

$$Supposing\colon\ \ g_{stp} = \frac{\partial C}{\partial w} \tag{2.21}$$

$$m_{stp} = \beta_1 m_{stp-1} + (1 - \beta_1) g_{stp} \tag{2.22}$$

$$V_{stp} = \sqrt{\beta_2 V_{stp-1} + (1 - \beta_2) g_{stp}^2} \tag{2.23}$$

$$w_{stp+1} = m_{stp} - \eta \, \frac{\sqrt{(1 - \beta_2)} m_{stp}}{(1 - \beta_1)(V_{stp} + \epsilon)} \tag{2.24}$$

Adam's original paper [69] recommends using default values of decays β_1, β_2 and ϵ as 0.9, 0.999 and 10^{-8} respectively. Adam works well in practice and compares favorably to other adaptive learning rate algorithms [68].
This thesis uses Adam as its optimizer with default values.

2.3.3 Backpropagation

Cost function C calculates the error based on the output from the final layer L. The optimizer calculates gradients $\frac{\partial C}{\partial w}$ for layer L and tunes parameters $\boldsymbol{w}$ (and bias $\boldsymbol{b}$) . The error is then *propagated backwards* towards the inner layers by calculating gradients for each parameter $\boldsymbol{w}$ in subsequent layers. This process of feeding back the error and using the optimizer to tune the parameters $\boldsymbol{w}$ is called backpropagation.
To explain backpropagation, Eq. 2.3 is represented in vectorized form as shown in Eq. 2.25.

$$\boldsymbol{y}^l = f\,(\boldsymbol{w}^l \boldsymbol{x}^l + \boldsymbol{b}^l) \tag{2.25}$$

The expression for weighted output $\boldsymbol{w}^l \boldsymbol{x}^l + \boldsymbol{b}^l$ can be represented by $\boldsymbol{z}^l$ as shown in equation 2.26.

$$\boldsymbol{y}^l = f\,(\boldsymbol{z}^l) \tag{2.26}$$

Initially the gradient is calculated w.r.t $\boldsymbol{x}$ i.e. $\frac{\partial C}{\partial x}$. Let $\frac{\partial C}{\partial x}$ be represented by $\nabla_x C$. To tune parameters $\boldsymbol{w}$ based on the output error calculation, the gradient must be calculated w.r.t $\boldsymbol{w}$ i.e. $\frac{\partial C}{\partial w}$. Through the chain rule, backpropagation allows gradient calculation for each weight. There are four fundamental equations for backpropagation [70]. Given that $\delta = \frac{\partial C}{\partial z}$ and operator $\odot$ is the Hadamard product, $\boldsymbol{\delta^L}$ can be calculated using Eq. 2.27.

$$\boldsymbol{\delta^L} = \nabla_x C \odot f'(\boldsymbol{z^L}) \tag{2.27}$$

Backpropagation moves from the output layer L to the inner layers. For any layer l, the $\boldsymbol{\delta^l}$ is calculated based on the calculation from $l + 1$. The mathematical expression is given in Eq. 2.28.

$$\boldsymbol{\delta^l} = ((\boldsymbol{w^{l+1}})^T \boldsymbol{\delta^{l+1}} \odot f'(\boldsymbol{z^l}) \tag{2.28}$$

Thus calculating $\frac{\partial C}{\partial b}$ for bias $\boldsymbol{b}$ and $\frac{\partial C}{\partial w}$ for weight $\boldsymbol{w}$ for each neuron in layer l using Eq. 2.29-2.30.

$$\frac{\partial C}{\partial b_i^l} = \delta_i^l \tag{2.29}$$

$$\frac{\partial C}{\partial w_{ij}^l} = x^{l-1}\delta_i^l \tag{2.30}$$

In section 2.2.2, a CL with input $\boldsymbol{x^l} \in \mathbb{R}^{H^l \times W^l \times D^l}$ convolved by $\boldsymbol{k^l} \in \mathbb{R}^{H^k \times W^k \times D^l \times D}$ is described. To simplify notations to explain backpropagation in CL, the calculations are only included for a single channel input, i.e. $D = 1$ (furthermore, $\boldsymbol{k^l}$ will be referred to as $\boldsymbol{w^l}$ to keep notations consistent with backpropagation equations 2.27-2.30). Thus, the dimension of the convolutional result will be $(H^l - H^k + 1) \times (W^l - W^k + 1)$. Gradients for each individual weight can be calculated using Eq. 2.31-2.32 [70]. Operator $*$ is the convolutional operation.

$$\frac{\partial C}{\partial w_{m,n}^l} = \{\nabla_x C_{i,j}^l\}_{rot^{180°}} * y_{m,n}^{l-1} \tag{2.31}$$

$$where: \nabla_x C_{i,j}^l = \nabla_x C_{i,j}^{l+1} * \{w_{m,n}^{l+1}\}_{rot^{180°}} f'(x_{i,j}^l) \tag{2.32}$$

2.3.4 Regularization

A designed CNN model is a function approximator. The function is approximated based on the test dataset and the structure of the CNN. The function provides information on a never-seen-before test dataset. The ability of the function to provide accurate information on a general problem like a test dataset is called generalization. There are two cases where generalization is not achieved. A model does not perform well on both the training and test dataset. This type of model is known as an underfit model. A model performs extremely well on training but fails to emulate the results on a test dataset. This type of model is known as an overfit model. The aim of model training is to perform "just-right" where performance in test is high and can generalize a test dataset well.

The technique applied to regularize the process of generalization is called regularization. Regularization is used either to change the structure or value of the weights. When a model is underfit, a simple solution is to increase the size of the model. This increases the number of parameters and the model learns better. However, the challenge is not when underfit but when overfit. This is where regularization plays an important role in improving a model.

Data augmentation is one of the regularization techniques that decreases invariance and helps a model to generalize better. Data augmentation artificially increases dataset by introducing noise such as skewing, distorting and rotating. DNN models such as CNN has shown to improve accuracy when dataset is increased. The model performs better when real-world "noise" is applied to the training dataset.

Dropout regularization is a technique where a portion of the neurons in FCL or GAPL temporary shut off. This creates a transient structure that is different from the original architecture. In any given layer l, depending on the global dropout setting, 10%-50% of the neurons are deactivated randomly. The remaining active neurons are forced to learn during a mini-batch training. The process repeats in the next batch with 10%-50% of neurons in any given layer l deactivated again randomly. The technique was first demonstrated by Krizhevsky et al. in 2012 [62].

Early stopping is another common technique. The test dataset is divided into test and validation. A learning curve is simultaneously plotted against epoch. An epoch is completed when the complete dataset (or all mini-batch) is used for training once. Normally, as epoch increases, test and validation accuracies increase. At some point, the learning curve diverges. After that, any more training leads to the model overfitting. Early stopping terminates the training at the divergent point.

This thesis uses data augmentation, dropout and early stopping regularization (details in Section 3.1.4) to improve model performance.

2.3.5 Note on Searchable Hyperparameters

Section 2.3.1. Hyperparameters detailed "built-in" hyperparameters of a CNN. However, there are parameters $inside$ hyperparameters. This thesis calls these hyperparameters as "searchable" hyperparameters. Optimizers such as Adam, an advanced mini-batch SGD, require user to set the initial learning rate $\eta^{initial}$ and size of the mini-batch called $batch\_size$. A learning rate $\eta^{initial}$ set too high or too low will affect training time and accuracy. So does the size of the $batch\_size$. For regularization techniques, searchable hyperparameters such as dropout percentage $dropout$ and the number of epoch $epochs$ must be set by user. The details of the search are given in Section 3.2.4.

3 Methodology

This chapter outlines the methodology used in designing and training a dedicated, ultralight CubeSat CNN called CubeSatNet. The components for CubeSatNet design and development is illustrated in Fig. 21. CubeSatNet is designed to classify images into “good” or “bad.” Section 3.1 shows how a CubeSat imagery is sourced, conditioned, and augmented to create a dedicated dataset from scratch. Section 3.1 outlines training platform (framework) and library used. Section 3.2.2 derives the size requirement for model architecture through the selection of MCU. Section 3.2.3 shows the steps towards building an ultralight CNN while Section 3.2.4 shows how the network is optimized.

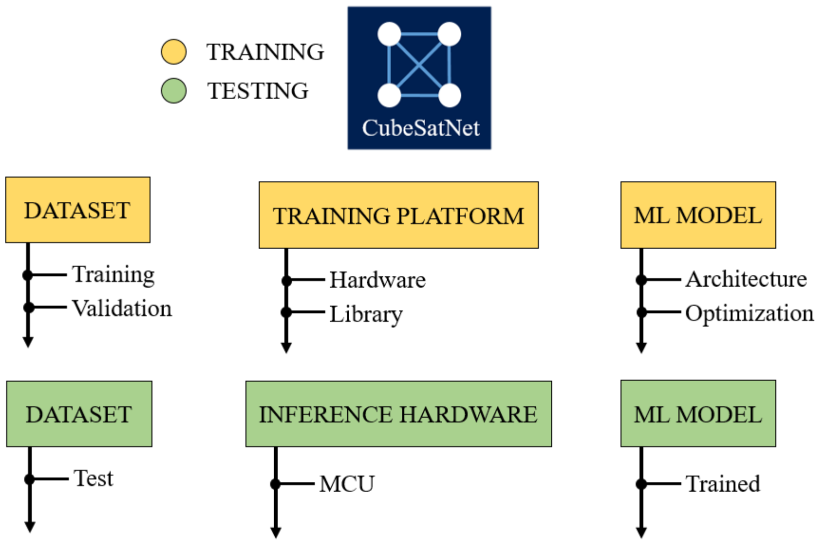


Figure 21. Breakdown of research components for building a CNN from scratch

3.1 Dataset Creation

Dataset is divided into training and test dataset. The training dataset is further divided into training and validation dataset. Test dataset must be created independent to the training dataset and should represent image taken from space as close to reality as possible. For this reason, all the images taken by BIRDS-3 CubeSat are used for test. All the other data is used for training. This section describes the end to end process steps to create a CubeSat database from scratch.

3.1.1 Dataset Mining

3.1.1.1 Real CubeSat Imagery

Since the end goal is to deploy a trained model on orbit, using real CubeSat imagery to train the CNN model is essential. An extensive search on CubeSats with imaging missions is conducted through the internet. The website nanosat.eu has created a list of nanosatellites with cameras. The first phase of data mining was done by going through each link and extracting all images from their official website. The second phase was focused on tracking images that have been posted on social media such as Facebook and Twitter and downloading them. Educational CubeSats which take images from space tend to reach out to a wider audience by using such platforms. After exhausting the source, the third phase used a direct contact approach. Emails were sent out to projects and teams requesting the data. The three phases of data collection supplemented internal image database of Kyutech. The internal image database has images from HORYU-4 and BIRDS-3 (only used for test dataset) project. Fig. 22 shows examples of some of images collected.

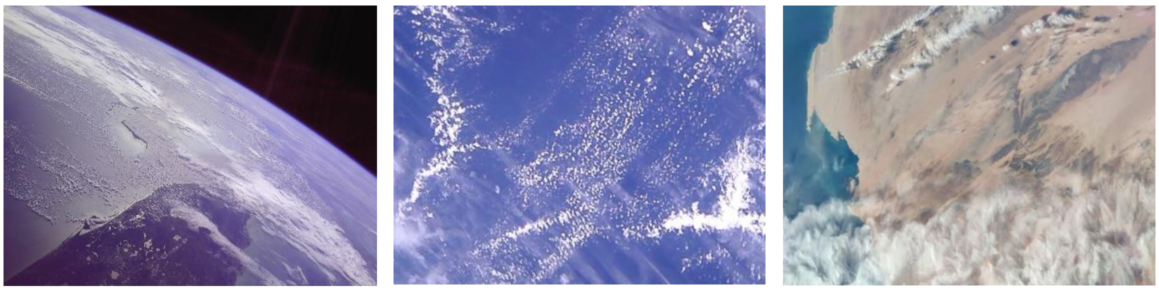


Figure 22. Images from CP9 [71], MySat-1 [72] and Xiaoxiang 1-08 [73]

3.1.1.2 ISS Imagery

CubeSat have limited downlink capacity. Teams downlink data but not all are published online. The image collected from on-orbit data is not enough to train CNN. To tackle the bottleneck, images from International Space Station (ISS) are used to supplement the dataset. ISS has high resolution cameras onboard that take images from Low Earth Orbit (LEO). LEO images taken aboard ISS are like CubeSats taking horizon images. NASA's website [74] uploads them on a regular basis.
Users on the video platform YouTube have created time-lapse videos by stitching together ISS images. Videos are useful because they have at least 30 frames per each second that can be extracted. Section 3.1.2.1 shows how videos are downloaded and images are extracted. Initially the process involved capturing the screen manually. A code written in python automated the entire process and helped to build a large database of ISS earth horizon imagery. Fig. 23 shows examples of some of images collected.

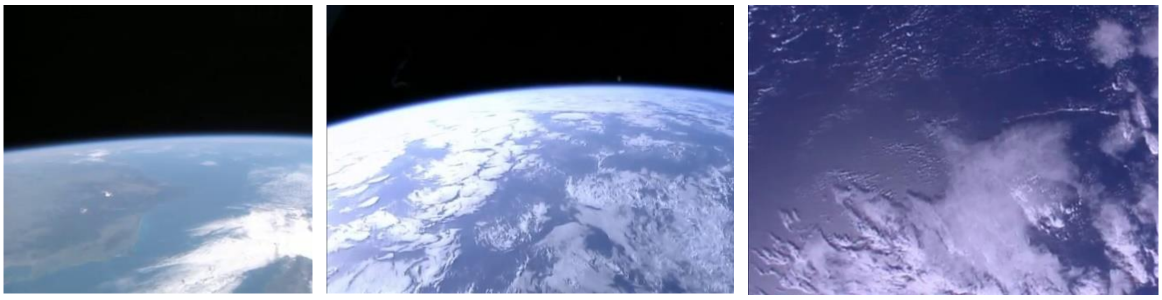


Figure 23. Shows images extracted from videos taken aboard the ISS

3.1.1.3 Sentinel 3A Imagery

Images generated from ISS are horizon images. The images must be balanced using nadir pointed images. Sentinel 3A is a LEO earth observation satellite by European Space Agency (ESA). The images taken are in R,G,B and are about 300m in GSD. The images look like CubeSat nadir pointed images. A JavaScript code explained in Section 3.1.2.3 shows how 150x150 thumbnails are generated using the browser-based Google Earth Engine (GEE). The same section also documents how browser-based Sentinel-hub Graphical User Interface (GUI) can also be used to download useful data. Fig. 24 shows examples of some of images collected.

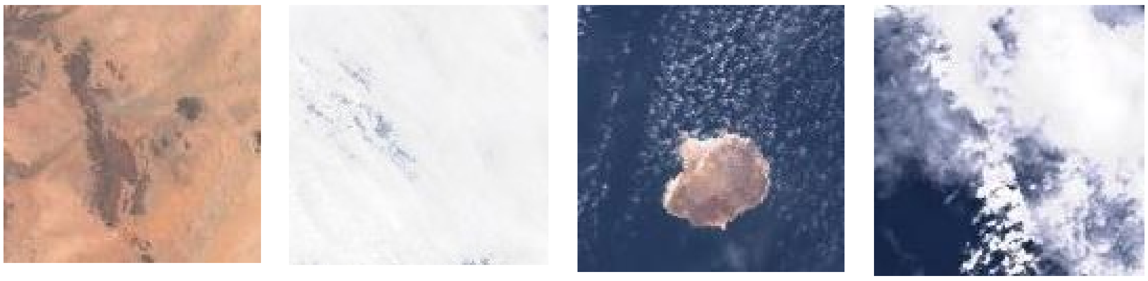


Figure 24. Images extracted from Sentinel 3A satellite

3.1.1.4 High Altitude Balloon (HAB) Images

Images generated from CubeSat are usually "good" images. CubeSat teams usually release good images. Images collected from ISS look "good" as well. A selective process in Sentinel 3A can generate bad images but since the process is still manual, generating large data takes time. Additionally, images that are classified as "bad"; sunburn, space, moon, or sun cannot be generated from previous sources.

A large number of High Altitude Balloon (HAB) projects have are documented in video on YouTube. These videos are good source for bad images as 1) the images are stored on SD card and have extremely long runtime 2) some cameras are pointed up to document the balloon bursting and so forth, frequently have footage of the sun, space and sunburn and 3) near space images look similar to images taken aboard CubeSats. By going through image extraction process as explained in Section 3.1.2.1. Fig. 25 shows examples of some of images collected.

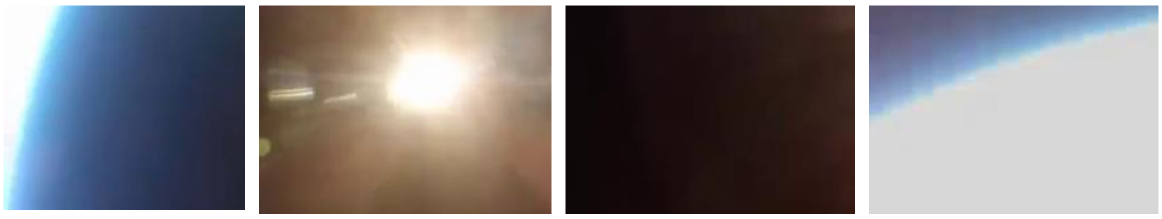


Figure 25. HAB images extracted from videos uploaded on YouTube using python

3.1.1.5 Amateur Rocket Images

Like HAB, action cameras like GoPro attached to amateur rockets provide additional source of data. The rockets reach altitudes above the Karmen line. Images extracted from videos taken aboard are like CubeSat images. A similar process of finding videos on YouTube, downlinking them, and extracting frames is conducted. Explanation is provided in Section 3.1.2.1. Fig. 26 shows examples of some of images collected.

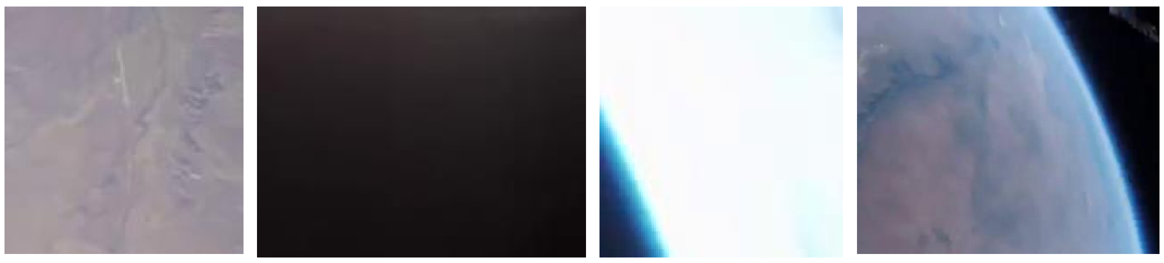


Figure 26. Images taken aboard amatuer rockets can also be useful for creating dataset

3.1.2 Extraction Process

3.1.2.1 Extracting Images from YouTube

Depending on the target, the keywords are placed on YouTube's search engine. Keywords such as *ISS images*, *ISS timelapse*, *high altitude balloon burst* or *amateur rocket footage* results in an array of videos. Pytube3 [75] is a python library that is used to download the target video first. Google Collaboratory (Colab) [76] is based on Jupiter notebook and is used to run the code. Colab runs on any internet browser and removes the need to install python on the computer. Colab also allows users to link personal Google Drive (GD) to access cloud storage. The platform provides over 100GB of storage during runtime. By using target video's YouTube address, Pytube3 directly downlinks the video on to Colab's cloud storage. If necessary, the videos can also be stored on the GD. For this thesis, mp4 videos at 320p with 30fps are downlinked.
FrameExtractor Class based on [77] using OpenCV library is then applied to extract frames. Frame extraction depends on how many images the user wants to extract per certain number of frames. For example, if the video is short, all the frames are extracted. If the video is long, one frame from every three are downlinked. The data is directly placed on the pre-determined folder on the GD. GD has a windows application that allows users to sync GD on the cloud directly to the hard drive. This way, as the images are being extracted on Colab and stored on GD, a real time transfer of the data to the hard drive is also taking place for data conditioning and augmentation. This saves a lot of time as there is no need to manually downlink the images from the cloud.

3.1.2.2 Bulk Download from Websites

CubeSat team place their images on their personal websites, social media pages and third-party image sharing portals. Firstly, a list of nanosatellites with imaging missions are collected. Free internet satellite database like nansats.eu and space.skyrocket.de are used in the process. Most of the images are shared on 1) twitter 2) project website and 3) Facebook. A chrome extension called "Download All Images" (DAI) is used after accessing each page. Clicking the activation button that comes with it, a zip file is created with all the images. The file is unzipped, unwanted images deleted, and necessary data extracted.

DAI is also useful when downloading image from search engine like Google Image Search (GIS). Placing keywords like *ISS* and *image from cubesat,* GIS shows a host of images that could be used for creating dataset. Clicking one image at a time and downloading it takes time and is a manual process. Using DAI, all images shown in the page are zipped and downloaded automatically. The file is then unzipped and images that are useful are kept.

3.1.2.3 Accessing Sentinel-3A Imagery

Google Earth Engine (GEE) is an open online platform where users can access large datasets of publicly released remote sensing data including Landsat, Sentinel and MODIS data through their Application Programming Interface (API). An online Integrated Development Environment (IDE) allows users to rapidly select, visualize and analyze data using a Javascript API. Furthermore, the IDE's console can generate thumbnails of 150x150 based on the satellite, time, location, and region of interest (ROI). This feature helps to generate a wide variety of nadir-pointed imagery data for ML. For this thesis, 300m Ground Sampling Distance (GSD) images from Sentinel 3A's Ocean and Land Color Instrument (OLCI) is used. BIRDS-3 1U CubeSats have a 5MP RGB camera on board with a maximum theoretical GSD of 300m calculated from 400km ISS orbit. Sentinel 3A's GSD is selected because 1) matches the resolution 2) bands 8,6,4 represent R,G,B 3) volume of data (available since 18 November, 2016.) Fig. 27 shows some of the examples of data generated from GEE.

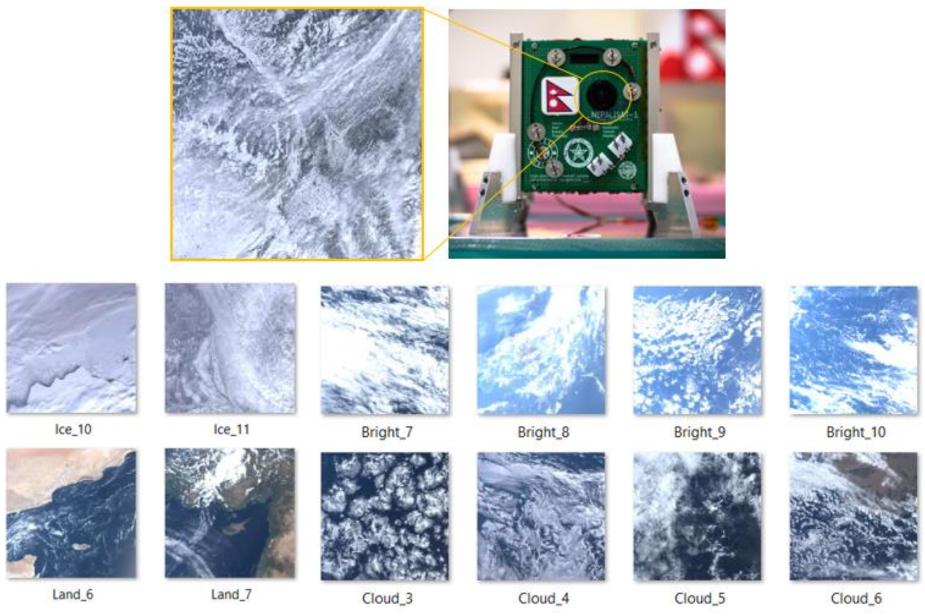


Figure 27. 150x150 thumbnails Sentinel 3A images extracted from Google Earth Engine

By changing ROI, data for clouds, land, ice, and water bodies can be created. By changing the saturation, bright images can be created. This is useful when ML model must be trained to identify overly light exposed pictures and classify them. All the images are nadir pointed.

Another way to access Sentinel 3A data is through Sentinel Hub Earth Observation (EO) Brower [78]. By searching data for a specific time and selecting true color images, 1468x644 Sentinel 3A images can be downloaded for a location. One generated image is large enough to split and create additional images. Splitting can be done in two ways. The first method is browser based GUI software called ImageSplitter by Postcron [79]. The software is originally designed to split images for Instagram. Each 1468x644 satellite image can generate 55 cropped images each of 128x129 pixels as shown in Fig. 28. The second method uses Image Slicer [80] python library. Using Image Slicer, each 1468x644 satellite image can generate 42 cropped images each of 209x107 pixels. The first method provides more control over how the image is cropped.

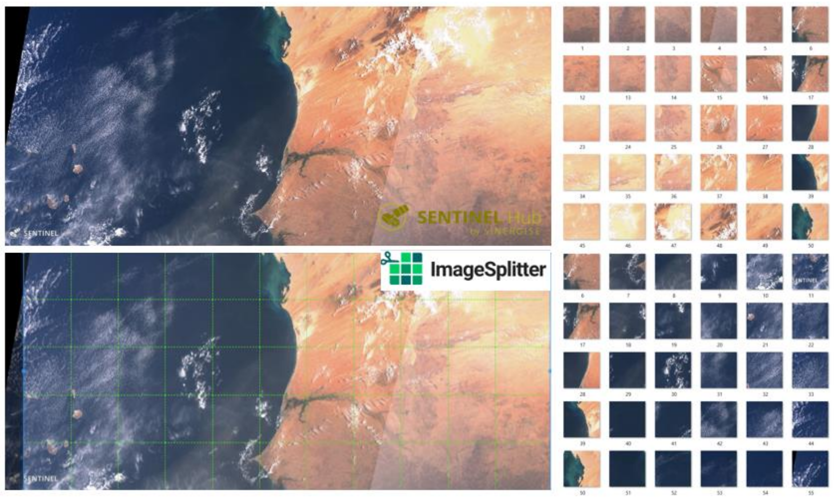


Figure 28. Sentinel 3A downloaded from Sentinel Hub EO Browser can be split using ImageSplitter

3.1.3 Data Conditioning: Cleaning

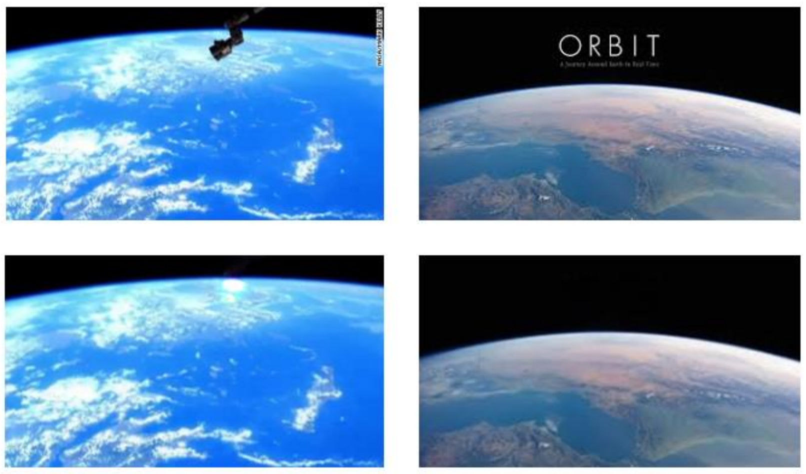
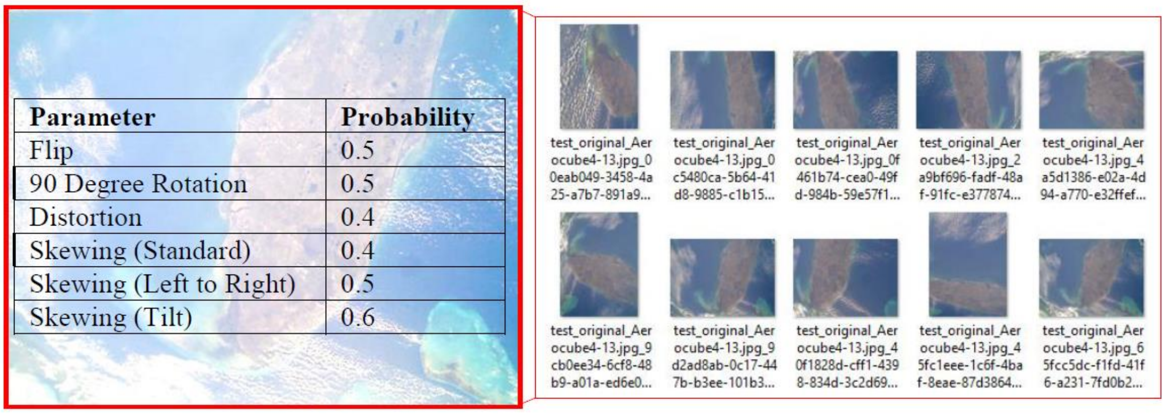


Figure 29. Objects and letters on the images have to be cleaned before augmentation

Data must be clean and uniform before it is augmented. Imagery could have watermarks, writings, spots, and objects that could reduce the overall quality. Removal of these noise from images is a critical step in creating a clean ML database. Adobe Lightroom's (AL) spot removal provides a simple, GUI based method to recreate the pixels lost while deleting unwanted parts. Results are shown in Fig. 29. AL has batch image processing which allows multiple images to be edited. However, AL is a paid software. A free alternative to AL is GIMP, an image editor that has similar spot removal feature called healing. The bottom half of Fig. 29 shows how objects and watermarking/writings have been removed from image using AL.

3.1.4 Data Augmentation



Parameter	Probability
Flip	0.5
90 Degree Rotation	0.5
Distortion	0.4
Skewing (Standard)	0.4
Skewing (Left to Right)	0.5
Skewing (Tilt)	0.6

test_original_Aerocube4-13.jpg_00eab049-3458-4a25-a7b7-891a9...
test_original_Aerocube4-13.jpg_0c5480ca-5b64-41d8-9885-c1b15...
test_original_Aerocube4-13.jpg_0f461b74-cea0-49fd-984b-59e57f1...
test_original_Aerocube4-13.jpg_2a9bf696-fadf-48af-91fc-e377874...
test_original_Aerocube4-13.jpg_4a5d1386-e02a-4d94-a770-e32ffef...
test_original_Aerocube4-13.jpg_9cb0ee34-6cf8-48b9-a01a-ed6e0...
test_original_Aerocube4-13.jpg_9d2ad8ab-0c17-447b-b3ee-101b3...
test_original_Aerocube4-13.jpg_40f1828d-cff1-4398-834d-3c2d69...
test_original_Aerocube4-13.jpg_45fc1eee-1c6f-4baf-8eae-87d3864...
test_original_Aerocube4-13.jpg_65fcc5dc-f1fd-41f6-a231-7fd0b2...

Figure 30. Data is augmented by combined flipping, rotation, distortion and skewing.

Data augmentation is a widely applied technique not only to increase the dataset but to improve a ML model through invariance and regularization. Invariance is a property where a ML can classify input data better in different image orientations. This simulates conditions where a CubeSat takes images when ADCS is either passive or is absent. Augmentation also helps to increase regularization. Overfitting is a common problem in ML models. Overfitting occurs when model has high accuracy on the training dataset but low on the validation. Overfitting is reduced by regularization through data augmentation.

This thesis uses Augmentor [81] python library for augmenting the data. The package has specific features designed for biomedical imaging but is general enough to be used as an image processing library for generating dataset for CubeSat images as well. A random combined effect of flipping, rotation, distortion, and skewing is placed on the images to extend the data. The values of each effect along with example results are shown in Fig. 30.

3.1.5 Data Conditioning: Uniformity

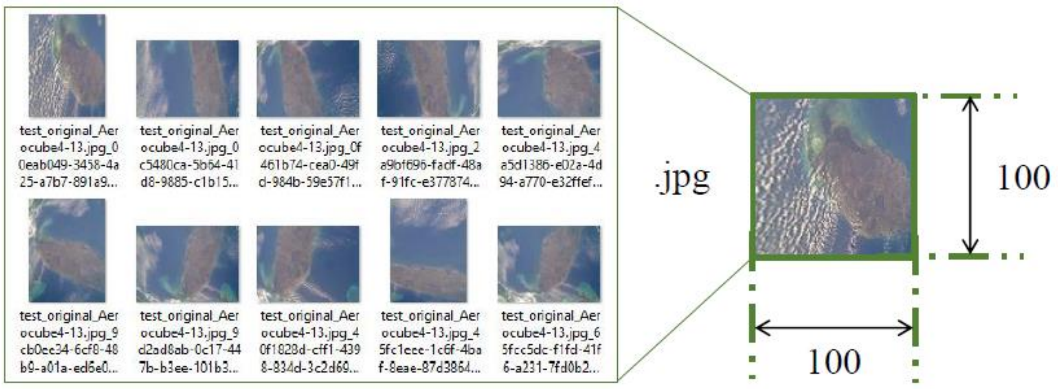
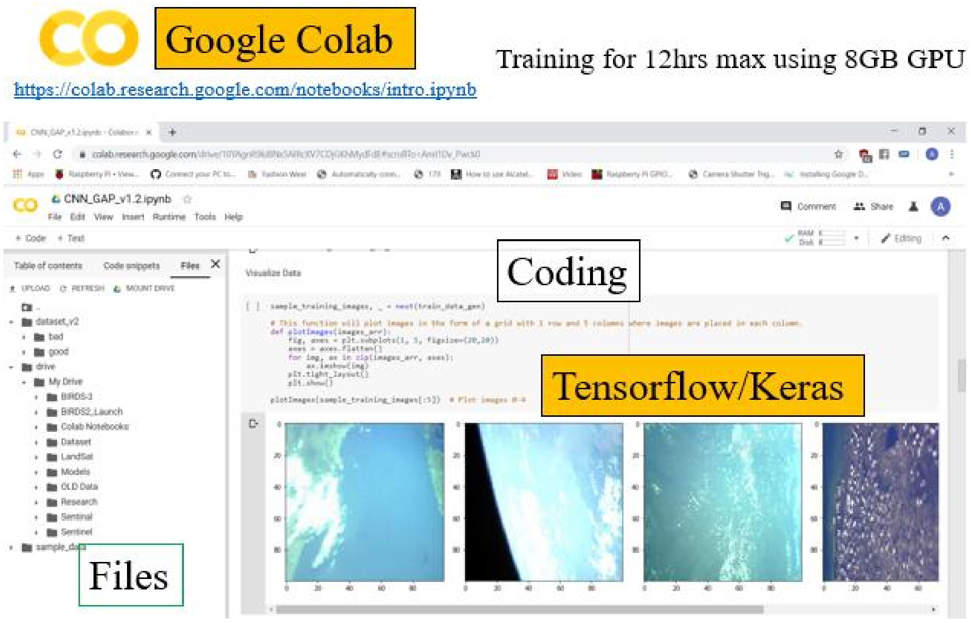

test_original_Aerocube4-13.jpg_00eab049-3458-4a25-a7b7-891a9...
test_original_Aerocube4-13.jpg_0c5480ca-5b64-41d8-9885-c1b15...
test_original_Aerocube4-13.jpg_0f461b74-cea0-49fd-984b-59e57f1...
test_original_Aerocube4-13.jpg_2a9bf696-facf-48af-91fc-e377874...
test_original_Aerocube4-13.jpg_4a5d1386-e02a-4d94-a770-e32ffef...
test_original_Aerocube4-13.jpg_9cb0ec34-6cf8-48b9-a01a-ed6e0...
test_original_Aerocube4-13.jpg_9d2ad8ab-0c17-447b-b3ee-101b3...
test_original_Aerocube4-13.jpg_40f1828d-cff1-4398-834d-3c2d69...
test_original_Aerocube4-13.jpg_45fc1ece-1c6f-4baf-8eae-87d3864...
test_original_Aerocube4-13.jpg_65fcc5dc-f1fd-41f6-a231-7fd0b2...
.jpg
100
100

Figure 31. All images are resized to 100x100 pixels and compressed in .jpg

OpenCV library in python is applied on these images to create uniform size and format. The size will depend on input of the ML architecture. For demonstration purposes, the paper has resized all the images to 100x100 pixels and changed the format to .jpg as shown in Fig. 31.

3.2 Training

3.2.1 Training Library and Platform



```
sample_training_images, _ = next(train_data_gen)

# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

plotImages(sample_training_images[:5])  # Plot images 0-4
```

Figure 32. Browser platform Google Colab allows researchers to build and train models for free

In 2019, Pytorch library overtook Tensorflow as the most popular library for researchers to do ML [82]. Pytorch is an open source ML library developed by Facebook. The API is simple to use and can run on python. Tensorflow is an open source ML library developed by Google. Tensorflow (TF) has a high level Keras API which simplifies library execution and coding. The coding is done in python. While Pytorch is popular with researchers, the industry relies on TF [82] as TF has options for embedded systems through TF Lite (TFlite). Trained TF models can be changed to TFlite models and deployed on mobile phones and now microcontrollers (MCU).

The research in the thesis is done on TF with Keras API. Fig. 32 shows how Google Colab is used while training the model on the browser. When the research began, TF was the most popular ML library [83]. In 2020, Google released TF 2.0. The latest version is faster and more resource efficient [84]. TF models can be converted to TFlite for mobile phone/Raspberry Pi (Rpi) deployment or further changed to quantized TFlite (qTFlite) to MCU applications. Running TF models on a resource constraint MCU began experimentally in 2018 and improved in 2019. Additionally, Google's Colab is a free Jupiter notebook platform that allows researchers to train ML models on a dedicated GPU on the cloud. This research is building a model to deploy on an MCU and has a minimal approach to resource use. Therefore, TF library is used on Colab to train the model.

3.2.2 MCU and Module Selection

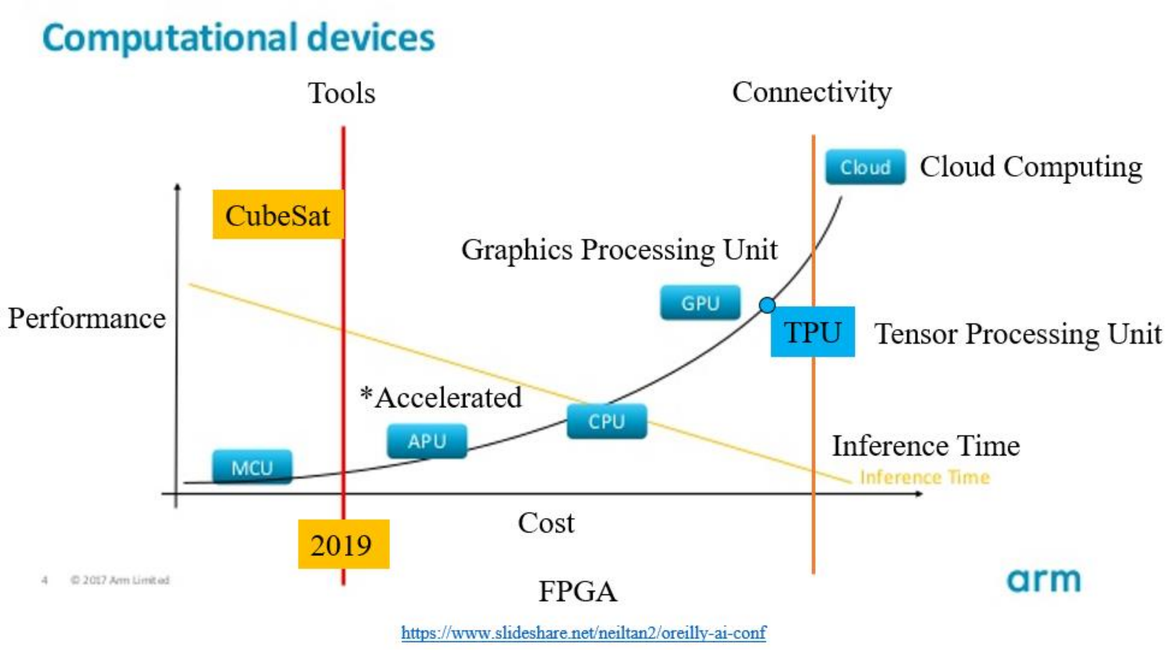


Figure 33. Performance vs Cost for different embedded and processing platform for ML [85]

SWaP requirements of 1U CubeSats mean that only low powered CPUs like Rpi zero or MCUs can be used for running ML models. Fig 33 shows that MCU has the least performance, takes most time for inference but is least costly to operate. MCU also fulfills the stringent SWaP requirements for a 1U CubeSat. If a model can be designed in such a way that it can fit on an MCU, the same model can fit on any processor. MCU is selected as it is the worst-case hardware to run the model.

Before 2019, running a ML model on an MCU was extremely labor-intensive process. However, due to availability of new ML tools and improvements in MCU architecture, ML inference can be done on MCUs as well. MCUs from ARM Cortex-M STM32 series developed by STMicroelectronics are high-performance, low-powered 32-bit controllers. According to ASPENCORE 2017 Embedded

Market Study published in April 2017, 2/3rd of the individuals surveyed were considering using STM32 for their next embedded project [86]. Survey in 2015 had the same trend before [87]. STM32 MCUs are known for its performance, fast clock speed, larger internal memory, and low power consumption. Table 3 shows some of CubeSats which have used STM32 in either their bus or subsystem. STM32 have just enough memory and processing speed to run a qTFlite model.

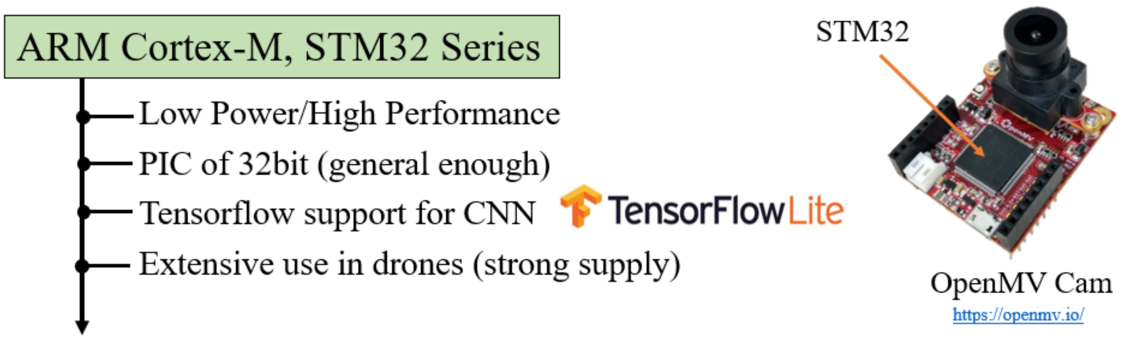

ARM Cortex-M, STM32 Series
Low Power/High Performance
PIC of 32bit (general enough)
Tensorflow support for CNN
TensorFlow Lite
Extensive use in drones (strong supply)
STM32
OpenMV Cam
https://openmv.io/

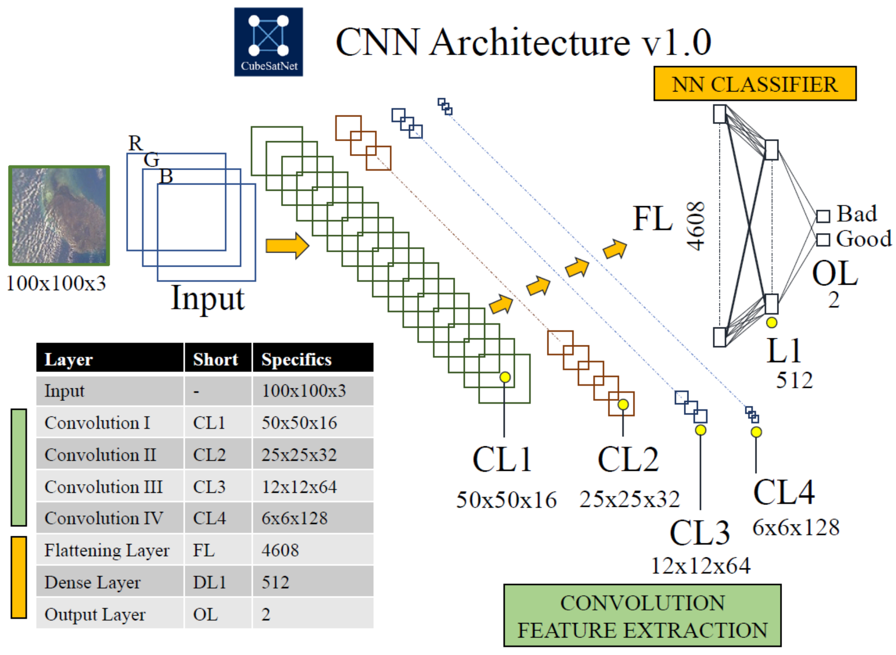
Figure 34 Show OpenMV Cam with STM32 ARM Cortex-M MCU which can load TFlite models

OpenMV produces machine vision camera modules that are based on STM32 chips. Fig. 34 shows the module. The camera modules can be programmed using their user-friendly Integrated Development Environment (IDE) in MicroPython, has TFlite support and has active customer support in their forum if issues arise. This thesis uses OpenMV Cam H7 with 400MHz STM32H743VI ARM Cortex M7 processor. The module consumes 170mA at 3.3V and can load model of about 100kB in size. Furthermore, MCU has been used in space. In April 1, 2019, Nanoavionic's M6P mission in LEO had SatBus 3C2 carried the same 400MHz STM32H7 ARM Cortex M7 processor into orbit [88].

Table 3. CubeSats that have used ARM Cortex M STM32 MCUs in their system design

CubeSat	**Type**	**Developer**	**Launch Year**	**MCU**
EstCube-1	1U	University of Tartu	2013	STM32F1
SNUSAT-1/1b	2U	Seoul National University	2017	STM32F4
FOX-1C	1U	AMSAT	2018	STM32L1
PW-Sat2	2U	Warsaw University of Tech.	2018	STM32F1
SNUSAT-2	3U	Seoul National University	2018	STM32F4
PicSat	3U	Observatoire de Paris	2018	STM32F3
KrakSat	1U	AGH Uni. of Sci. and Tech., SatRevolution S.A.	2019	STM32xx
M6P	6U	Nanoavionics	2019	STM32H7

3.2.3 CNN Architecture Design



Layer	Short	Specifics
Input	-	100x100x3
Convolution I	CL1	50x50x16
Convolution II	CL2	25x25x32
Convolution III	CL3	12x12x64
Convolution IV	CL4	6x6x128
Flattening Layer	FL	4608
Dense Layer	DL1	512
Output Layer	OL	2

Figure 35 CNN architecture of CubeSatNet v1.0

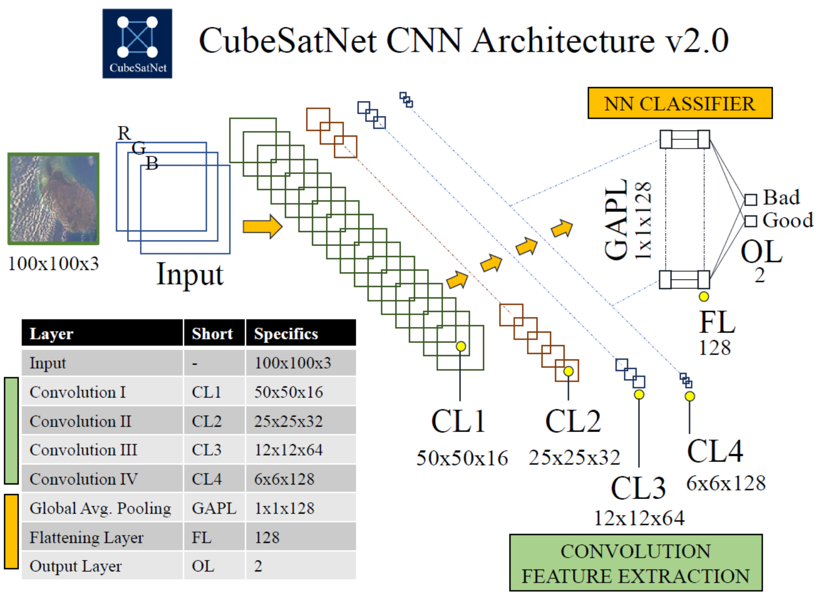
Two CNN models are named as CubeSatNet v1.0 and CubeSatNet v2.0. Fig. 35 shows that CubeSatNet v1.0 architecture has four Convolutional Layers (CL), a Flattening Layer (FL), a Fully Connected Layer (FCL) and an Output Layer (OL). CL has kernel of 3x3, same padding and ReLu activation layer. After each CL, a max-pooling layer of matrix 2x2 and stride 2 is applied. 16, 32, 64 and 128 features are extracted as the image passes through each subsequent CL. Feature size is reduced by the pooling layer to 50x50, 25x25, 12x12 and 6x6 respectively. Dropouts are imposed after FL and FCL. Softmax function is implemented in OL to provide probabilities for each class. During training, categorical cross-entropy calculates the loss and Adam optimizes during backpropagation. Table 4 summarizes each layer and its parameters. Table 4 also enlists properties for each convolutional layer.

Table 4. Layer definition and CL parameter properties for CubeSatNet v1.0

Layer	**Type**	**Shape**	**Param #**
CL1	Conv	(100,100,16)	448
CL2	Conv	(50,50,32)	4,640
CL3	Conv	(25,25,64)	18,496
CL4	Conv	(12,12,128)	73,856
FL	Flattening	4608	0
FCL	Dense	512	2,359,808
OL	Output	2	1,026
Total Parameters			2,458,274

CL	**Type**	**Padding**	**Stride**
Kernel	3x3	Same	1
Activation	ReLu	-	-
Pooling	Max (2x2)	-	2

Table 4 shows that there are almost 2.5 million parameters that need to be trained in CubeSatNet_v1. The network size is proportional to the number of parameters. To reduce the number of trainable parameters, CubeSatNet v2.0 has Global Average Pooling Layer (GAPL) instead of FCL. The total number of parameters is now under 100,000. Fig. 36 shows the CNN architecture and Table 5 enlists the layers of CubeSatNet v2.0.



Layer	Short	Specifics
Input	-	100x100x3
Convolution I	CL1	50x50x16
Convolution II	CL2	25x25x32
Convolution III	CL3	12x12x64
Convolution IV	CL4	6x6x128
Global Avg. Pooling	GAPL	1x1x128
Flattening Layer	FL	128
Output Layer	OL	2

Figure 36 CNN architecture of CubeSatNet v2.0

Table 5. Layer definition for CubeSatNet_v2. FL has been replaced by GAPL

Layer	**Type**	**Shape**	**Param #**
CL1	Conv	(100,100,16)	448
CL2	Conv	(50,50,32)	4,640
CL3	Conv	(25,25,64)	18,496
CL4	Conv	(12,12,128)	73,856
GAPL	**Global Avg.**	**128**	**0**
FL	Flattening	128	0
OL	Output	2	258
Total Parameters			97,698

3.2.4 Training and Optimization

The training dataset is further divided into training and validation dataset in 80:20 ratio. 48,000 images are used for training while 12,000 is used for validation. Three key hyper-parameters are selected for optimization; dropout $dropout$ before OL, batch size $batch\_size$ of images during training and Adam's learning rate $\eta^{initial}$ for backpropagation. Table 6 shows the number of search parameters for $dropout$, $batch\_size$ and $\eta^{initial}$ are 4, 7 and 5 respectively. The total number parameters in the search space is 140. Liashchynskyi and Liashchynskyi [89] states that grid search is a better approach than genetic algorithm when search space is small. For $dropout$, $batch\_size$ and $\eta^{initial}$, the optimum values through grid search are 64, 0.0001 and 0.3 respectively. The values have been bolded in Table 6. Each training took about an hour using the platform explained in 3.1.3. TF model is generated in the process.

Table 6. Summary of the total number of parameters for hyper-parameter tuning

Hyper-parameter	**Values**	**No.**
Batch Size $batch\_size$	32, **64**, 128, 512, 1024, 2048	7
Learning Rate $\eta^{initial}$	0.1, 0.01, 0.001, **0.0001**, 0.00001	5
Dropout $dropout$	0.2, **0.3**, 0.4, 0.5	4
Total Search Space	BS*LR*DO	140

Fig. 37 shows that the model achieved a final accuracy of 90.47% at 43 epochs. After 43 epochs, the plots for training and validation diverge. A red line is marked on Fig. 37 to illustrate the diverging point which is the optimum value. The divergence after the red line in training and validation losses is because of overfitting. Bigger fluctuations are observed as the overfit model is sensitive to minor changes in the training data.

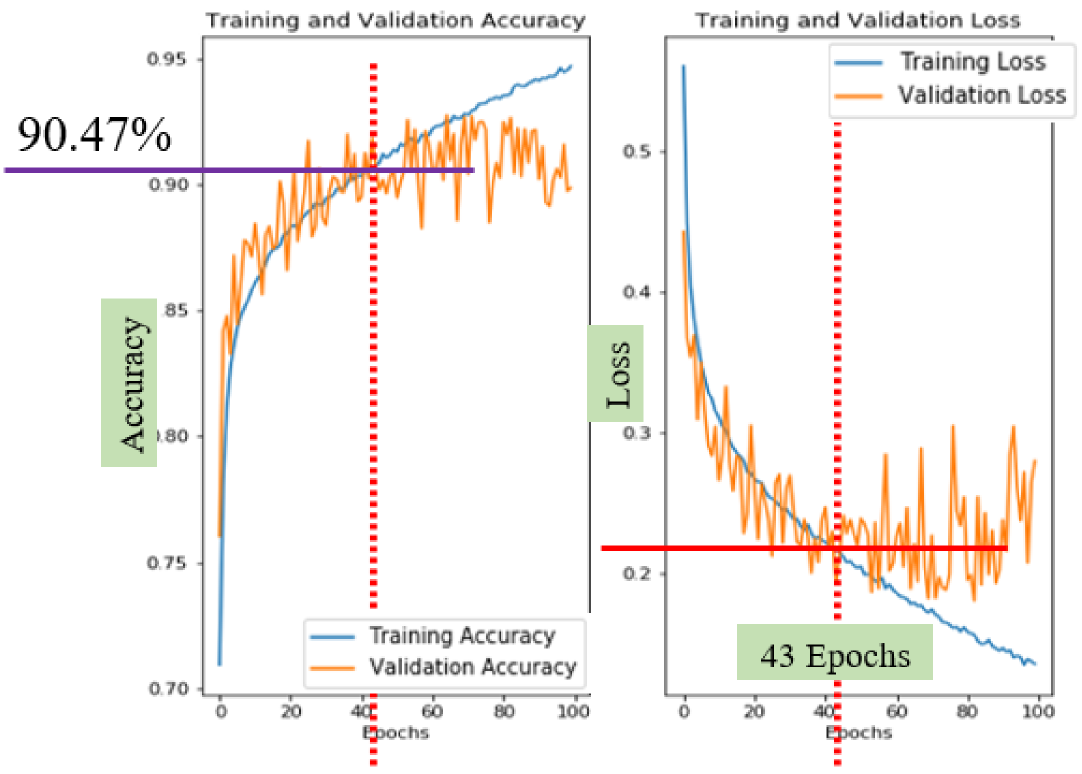


Figure 37. Training and Validation Accuracy on the left and Loss on the right.

3.2.5 Optimized Model Comparison

Table 7. Test accuracy for different variations (layers, input size) of models

CNN Model	**Layer Selection**	**Input Size**	**Parameters**	**Test Accuracy**
CubeSatNet_v1	CLx4, FCL	100x100	2,458,274	90.12%
CubeSatNet_v2	**CLx4, GAPL**	**100x100**	**97,698**	**90.47%**
CubeSatNet_v3	CLx4, GAPL	75x75	97,698	88.06%
CubeSatNet_v4	CLx4, GAPL	50x50	97,698	88.04%
CubeSatNet_v5	CLx3, GAPL	50x50	23,714	86.92%
CubeSatNet_v6	CLx2, GAPL	25x25	5,154	85.05%

Table 7 shows a comparison of CubeSatNet_v2 with different versions of CNN model. CubeSatNet_v1 took 100x100 input with 3 channels (RGB). CubeSatNet_v2 reduced the number of parameters significantly while improving accuracy when FCL is replaced by GAPL. However, with progressive removal of CL and reduction of input size showed accuracy to reduce. Table 8 shows how original selection of base reference values of 16, 32, 64, 128 feature extraction for subsequent CL in CubeSatNet_v2 showed highest accuracy. Reducing depth of CL showed lower accuracies as documented in Type A and Type B where 4, 8, 16, 32 and 8, 16, 32, 64 respectively are selected as the depth of CL. The optimum model CubeSatNet_v2 is about 104kB.

Table 8. Test accuracy for different variations (convolutional depth) of models

CubeSatNet_v2	**CL1**	**CL2**	**CL3**	**CL4**	**Parameters**	**Test Accuracy**
Type A	4	8	16	32	6,282	89.88%
Type B	8	16	32	64	24,658	89.47%
Original	**16**	**32**	**64**	**128**	**97,698**	**90.47%**

4 Results

4.1 Test Dataset

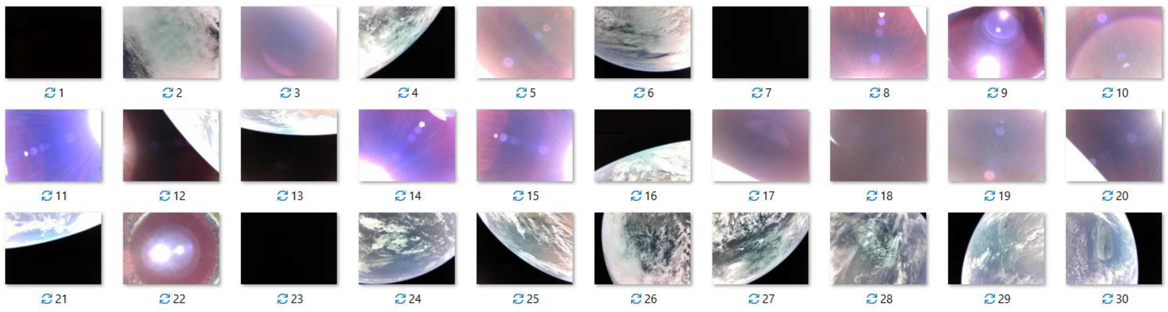


Figure 38 shows 30 images taken by BIRDS-3 satellites which are used as a test dataset

To find how CubeSatNet would perform in space, the CNN should be tested on a completely new dataset that is as close to the real satellite image as possible. BIRDS-3 CubeSats have been taking images from 380km ISS orbit since June 2019. A test dataset of first thirty images taken by BIRDS-3 are used to create a test dataset for CubeSatNet. None of these images were used for training or validation. Fig 38 shows all the images used for test dataset.

4.2 Quantization and Model Performance

The testing has been done on three different TF models; the original generated from training, a Tensorflow lite (TFlite) model and an 8-bit quantized Tensorflow lite (qTFlite) model through post-training quantization process. TFlite is designed for inference on embedded systems. The 8-bit qTFlite is the actual model that is deployed on the MCU of the CubeSat. The TFlite conversion and the quantization process is done following the documentation provided by Google.

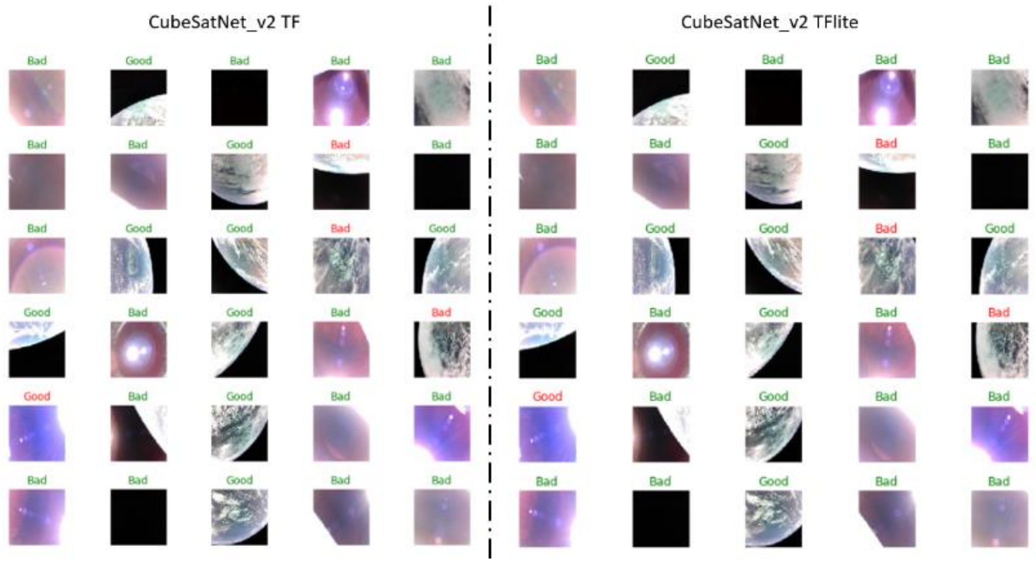


Figure 39. CubeSatNet_v2 TF and TFlite model's performance on BIRDS-3 images. Red shows images that are incorrectly classified. The accuracies for both models are equal at 86.67%.

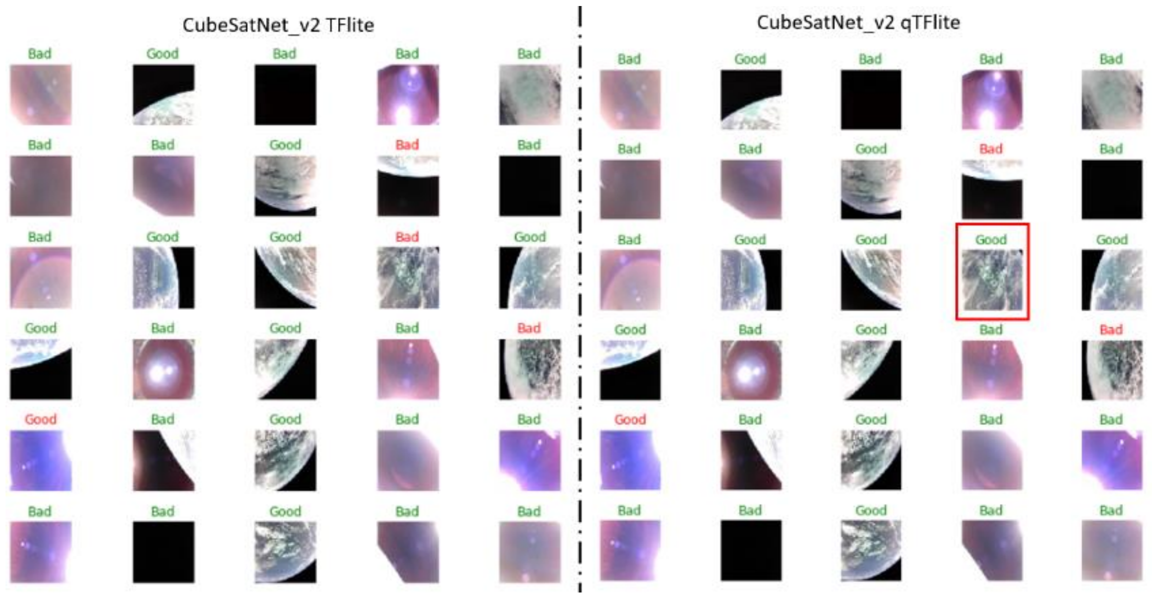


Figure 40. CubeSatNet_v2 TFlite and qTFlite model's performance on BIRDS-3 images. Red shows changes in classification. TFlite's accuracy is 86.67% while qTFlite's accuracy increased to 90% which is unexpected

The visual representation of the image classification are shown in Fig. 39 and 40. Fig. 39 shows the difference in the inference between TF and TFlite models. Fig. 39 shows that there is no change in accuracy when TF is converted to TFlite. Both have a test accuracy of 86.67% and is about 4% lower than the validation accuracy during training. Interestingly, qTFlite model performs better than TFlite and TF with 90% accuracy (see Fig.40). This is unexpected as factorial reduction in size should have maintained or reduced the accuracy. Table 9 shows how the probability calculation from output layer (confidence values) changed for the image boxed in Fig. 40. The model's prediction for "bad" and "good" for the image is borderline. Initially, the model is slightly more confident that the image is "bad." Post-training quantization changed the value in such a way that qTFlite is slightly more confident that the same image is "good." Since the original classification was incorrect, the qTFlite classified it as correct. This explains the increase in accuracy.

Table 9. Confidence value of the image that flipped classification while changing models

Bad			**Good**		
TF	TFlite	qTFlite	TF	TFlite	qTFlite
0.543	0.543	**0.497**	0.456	0.456	**0.502**

Table 10 outlines all the models discussed so far, their size and their respective accuracies. The final model of CubeSatNet_v2 is in bold. qTFlite is 10x smaller than TF model that was trained and has almost equal accuracy of 90%. Performance comparison between SVM, AE and DBN are given in section 4.3.

Table 10. Summary of changes of size and accuracy of different models

Model	**Model**	**Size**	**Image Type**	**Training**	**Test**
CubeSatNet_v1	TF	28MB	RGB	89.76%	-
CubeSatNet_v2	TF	1MB	RGB	90.47%	86.67%
	TFlite	387kB	RGB	-	86.67%
	qTFlite	**104kB**	**RGB**	**-**	**90%**

Table 11. Confusion matrix for CubeSatNet_v2 qTFlite along with F1 score

n=30	**Actual**				
Predicted	Bad	Good			
Bad	17[TP]	2 [FP]	Recall (R)	$\frac{TP}{TP+FN}$	0.94
Good	1 [FN]	10[TN]	Precision (P)	$\frac{TP}{TP+FP}$	0.90
			F1 Score	$\boldsymbol{2*\frac{P*R}{P+R}}$	**0.92**

The confusion matrix is presented on Table 11. The total number (n) of images classified is 30. Among them, 17 are True Positives (TP) where CubeSatNet_v2 qTFlite correctly classified "bad" images. Likewise, 1 is False Negative (FN) where the CNN incorrectly classified as "good" image. The CNN has 2 False Positives (FP) image where "good" images are classified as "bad" and 10 True Negative (TN) where images are correctly classified as "good." F1 score is calculated to be 0.92. The maximum achievable score is 1.

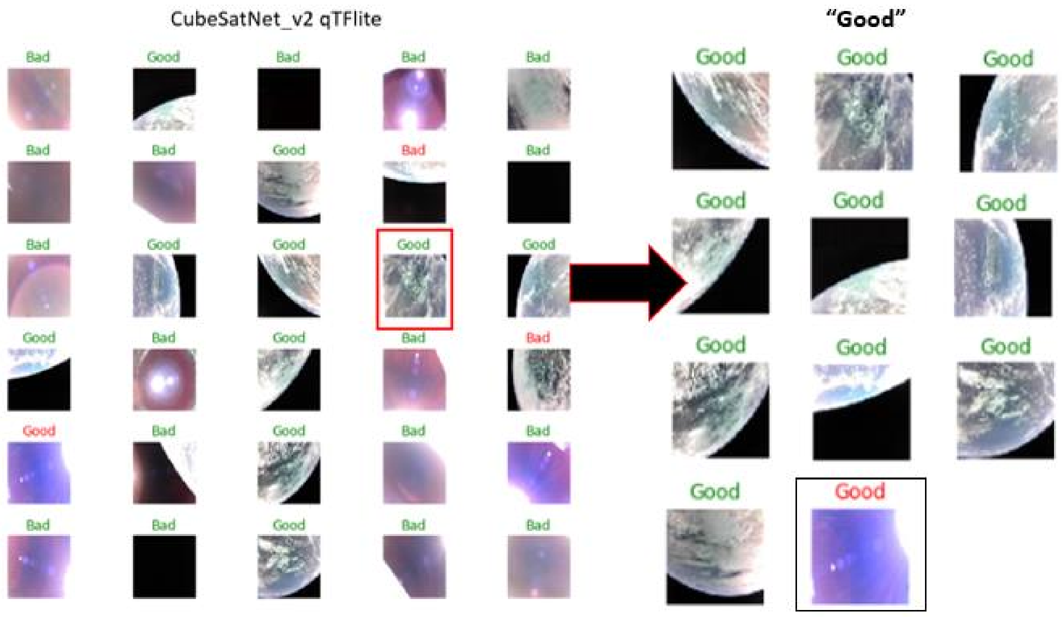


Figure 41. Shows images that were classified as good

Fig. 41 shows the images that would have been downlinked if hypothetically the CNN is applied on BIRDS-3 from deployment. All 18 images classified as "bad" would have been deleted. In that 2 actually "good" images would have been lost. The remaining 11 images would have been taken about a month to downlink cutting down the operation time to by 2/3rd. Out of the 11 images downlinked, 1 would have been incorrectly classified as good. The CNN has the potential to save operation time and manual work while significantly improving the quality of image downlinked.

4.3 Model Comparative Performance

Table 12. CNN is compared to SVM, DBN and AE

Model	Framework	Size	Input
SVM	Matlab	500kB	B&W
CNN	**Tensorflow**	**104kB**	**RGB**
DBN	Tensorflow	324kB	B&W
AE	Tensorflow	621kB	RGB

For simplicity, CubeSatNet_v2 qTFlite is referred to as CNN in this section. The CNN performance must be compared to other machine learning classification methods. The CNN model is compared to SVM, DBN and AE trained on the same dataset summarized in Table 12. SVM is built by Kyutech's BIRDS-4 team. The model is currently part of the ICU on-board BIRDS-4's 1U CubeSat constellation which is scheduled to launch late 2020. The size is 500kB, takes grayscale input and is trained in Matlab. DBN consists of two hidden layers and is based on the DBN classification library designed by DBNAlbert [90]. To minimize size, the DBN takes grayscale input, is limited to 325kB and uses TF framework. AE is unsupervised and the design algorithm is based on Ardamavi [91]. The size is 621kB, takes RGB input and is trained using high level Keras API to access TF framework.

Table 13. CNN shows highest performance as compared to SVM, DBN and AE

	Confusion Matrix Values				**Performance on BIRDS-3 Test Dataset**				
Model	**TP**	**FP**	**FN**	**TN**	**Total**	**A**	**R**	**P**	**F1**
SVM	15	5	3	7	30	73.33%	0.83	0.75	0.79
CNN	**17**	**2**	**1**	**10**	**30**	**90%**	**0.94**	**0.90**	**0.92**
DBN	14	3	4	9	30	76.67%	0.78	0.82	0.80
AE	14	6	4	6	30	66.67%	0.78	0.70	0.74

Table 13 lists out the results on BIRDS-3 test dataset. Among the four models, CNN has the smallest size with 104KB. CNN also takes RGB input like AE but opposed to grayscale inputs of SVM and DBN. CNN shows highest test accuracy (A) with 90% followed by DBN, SVM and AE with 76.67%, 73.33% and 66.67% respectively. CNN displayed the highest recall (R) and precision (P). The F1 score for CNN is 0.92. DBN, SVM and AE have F1 score of 0.80, 0.74 and 0.79 respectively. The results show that CNN outperforms DBN, SVM and AE for CubeSat image classification.

5 Conclusion

This thesis presented an innovative method to tackle the limited data downlink capability of a 1U CubeSat. An ultralight CNN architecture called CubeSatNet is proposed and trained on a novel CubeSat image database of 60,000 augmented images to prioritize quality image data for downlink. The final model is just over 100kB in size and is small enough to load on an ARM Cortex MCU and has an accuracy of 90%. Test is done on first thirty on-orbit images from Kyutech's BIRDS-3 CubeSats. The results showed that, if implemented, operation time could be cut by about 2/3 while significantly improving on quality of image received. The CNN outperformed SVM, DBN and AE tested on the same BIRDS-3 test dataset.

5.1 Future Potential

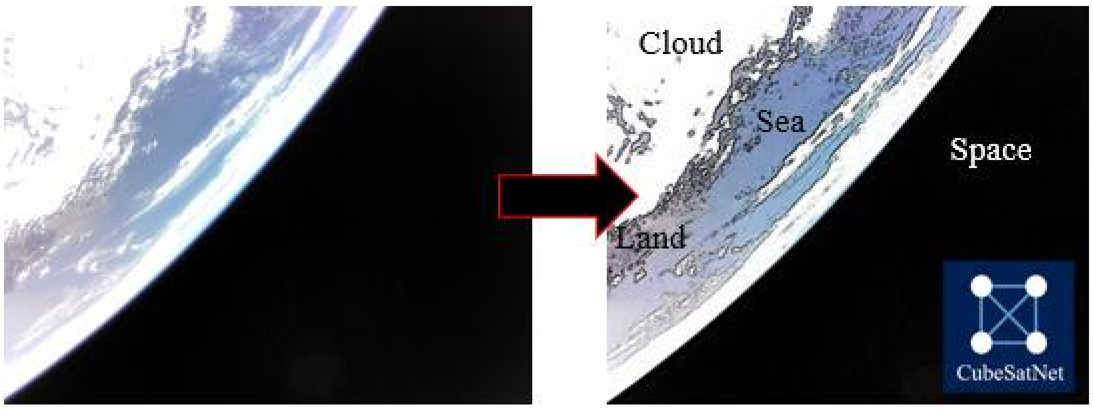


Figure 42. Images taken in space can be segmented into different classifications

The CNN model proposed in this thesis classifies the whole image. The next step would be to train the model to segment the image into sections and classify the parts individually. This is called image segmentation. Image segmentation is achieved by placing a small classification window over the image, classifying the small portion, shifting the window to the next group of pixels, and classifying again. The process is repeated until all the image is covered. Similar classifications are grouped together to create a boundary. An example is shown in Fig 42. The image is segmented into cloud, sea, land and space. A practical use would be to calculate cloud percentage cover in an image. Based on that, the ground operator can decide to download the image or skip it.

There is also the potential to merge CubeSatNet with other NN. Like combining convolutional layers to extract features from an image, CNN can be combined with Recursive Neural Network (RNN) to generate text from the image. RNN is used for generating content for speech recognition, translation, natural language processing and image description. Natural language descriptions can be placed to caption an image by first extracting features using CNN. Using RNN, the features are then digested to form a sentence long description of that image. Fig. 43 shows a hybrid NN that is used to describe an image taken by a CubeSat.

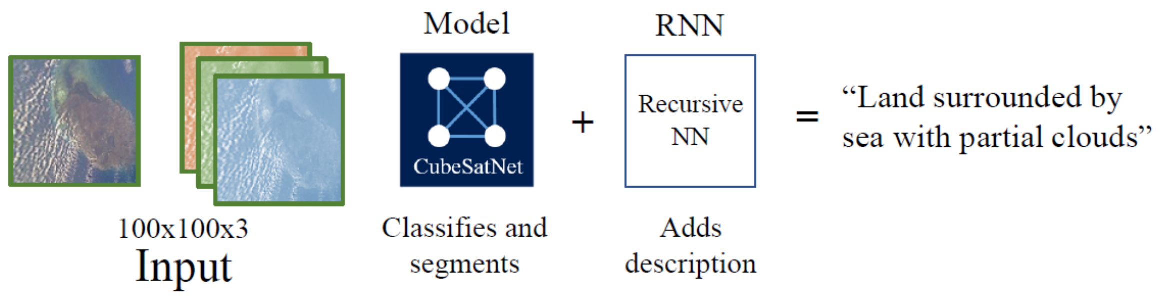


Figure 43. CNN models can be combined with RNN to describe the images taken by CubeSats

Hybrid NN could be used on CubeSats to generate a separate text file that has all the information about each image the satellite has taken. While operating, the ground operator sends a command to downlink the file. The operator can then select which image to downlink based on the individual image descriptions. This provides more freedom of choice; for instance, depending on what the operator needs, he/she could select an image that is an island, land partly surrounded by sea, image of sea or just land.

This can further improve to identify landmass if enough labelled dataset is created. In Fig. 43, the image is described as "Land surrounded by sea with partial clouds." The word "land" can be replaced by "Sri Lanka" or "Kyushu" or "Korean Peninsula." The ground operator can then select the image based on the geographical description.

6 Appendix

Complete Training/Testing code on google colab:

https://colab.research.google.com/drive/1W5VH_s1-3cHYi541XxoZABZstu4JkhaK?usp=sharing

Complete 60,000 Training dataset with 30 Test dataset:

https://data.mendeley.com/datasets/47vtp22vs7/1

7 References

[1] H. Heidt, J. Puig-Suari, A. Moore, S. Nakasuka, and R. Twiggs, “CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation,” 2000.

[2] “Suppliers — CubeSat.” https://www.cubesat.org/new-index (accessed Jun. 18, 2020).

[3] S. S. Technology, “State of the Art Small Spacecraft Technology,” 2018. Accessed: Jun. 18, 2020. [Online]. Available: http://www.sti.nasa.gov.

[4] N. Saeed, A. Elzanaty, H. Almorad, H. Dahrouj, T. Y. Al-Naffouri, and M.-S. Alouini, “CubeSat Communications: Recent Advances and Future Challenges,” *IEEE Commun. Surv. Tutorials*, pp. 1–1, Aug. 2019, Accessed: Jun. 18, 2020. [Online]. Available: http://arxiv.org/abs/1908.09501.

[5] “SwissCube - eoPortal Directory - Satellite Missions.” https://directory.eoportal.org/web/eoportal/satellite-missions/s/swisscube (accessed Jun. 18, 2020).

[6] “Nanosats Database | Constellations, companies, technologies and more.” https://www.nanosats.eu/ (accessed Jun. 18, 2020).

[7] “Timeline of first artificial satellites by country - Wikipedia.” https://en.wikipedia.org/wiki/Timeline_of_first_artificial_satellites_by_country (accessed Jun. 18, 2020).

[8] “Planet | Insights - Our Constellations.” https://storage.googleapis.com/planet-ditl/day-in-the-life/index.html (accessed Jun. 18, 2020).

[9] “Spire @ Nanosats Database.” https://www.nanosats.eu/org/spire (accessed Jun. 18, 2020).

[10] O. Koudelka, R. Kuschnig, M. Wenger, and P. Romano, “Nanosatellite missions-the future.”

[11] A. Poghosyan and A. Golkar, “CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions,” *Progress in Aerospace Sciences*, vol. 88. Elsevier Ltd, pp. 59–83, Jan. 01, 2017, doi: 10.1016/j.paerosci.2016.11.002.

[12] S. Asmar and S. Matousek, “Mars cube one (MarCO) shifting the paradigm in relay deep space operations,” 2016, doi: 10.2514/6.2016-2483.

[13] “MarCO - Satellite Missions - eoPortal Directory.” https://directory.eoportal.org/web/eoportal/satellite-missions/content/-/article/marco (accessed Jun. 18, 2020).

[14] “JPL | Cubesat | MarCO.” https://www.jpl.nasa.gov/cubesat/missions/marco.php (accessed Jun. 18, 2020).

[15] K. Devaraj, R. Kingsbury, M. Ligon, J. Breu, V. Vittaldev, and B. Klofas, “Dove High Speed Downlink System.”

[16] S. Palo, D. O. Connor, E. Devito, and R. Kohnert, “SSC14-IX-1 Expanding CubeSat Capabilities with a Low Cost Transceiver,” 2012.

[17] “Watch 3 ‘BIRDS’ Take Flight from the International Space Station | Space.” https://www.space.com/space-station-deploys-birds-3-cubesats-video.html (accessed Jun. 18, 2020).

[18] EnduroSat, “UHF Transceiver II CubeSat Communication | CubeSat by EnduroSat.” https://www.endurosat.com/cubesat-store/cubesat-communication-modules/uhf-transceiver-ii/ (accessed Dec. 20, 2019).

[19] EnduroSat, “S-Band Transmitter - CubeSat Communication Module | EnduroSat.” https://www.endurosat.com/cubesat-store/cubesat-communication-modules/s-band-transmitter/ (accessed Oct. 12, 2019).

[20] EnduroSat, “X-Band Transmitter CubeSat Communication Module | EnduroSat.” https://www.endurosat.com/cubesat-store/cubesat-communication-modules/x-band-transmitter/ (accessed Dec. 20, 2019).

[21] O. Cristea, P. Dolea, and P. V. Dascăl, “S-band ground station prototype for low-earth orbit nanosatellite missions,” *Telecomunicatii*, no. 2, pp. 64–71, 2009.

[22] S. Chien *et al.*, “Onboard autonomy on the intelligent payload experiment CubeSat mission,” *J. Aerosp. Inf. Syst.*, vol. 14, no. 6, pp. 307–315, 2017, doi: 10.2514/1.I010386.

[23] D. R. Thompson *et al.*, “Onboard machine learning classification of images by a cubesat in Earth orbit,” *AI Matters*, vol. 1, no. 4, pp. 38–40, 2015, doi: 10.1145/2757001.2757010.

[24] J. L. Rash and C. P. Dent, “Space applications of artificial intelligence; Proceedings of the Annual Goddard Conference, Greenbelt, MD, May 16, 17, 1989,” 1989.

[25] C.-C. Lee, “Intelligent control based on fuzzy logic and neural net theory,” 1991.

[26] L. Feruglio and S. Corpino, “Neural networks to increase the autonomy of interplanetary nanosatellite missions,” *Rob. Auton. Syst.*, vol. 93, pp. 52–60, 2017.

[27] M. A. C. Silva, M. Shan, A. Cervone, and E. Gill, “Fuzzy control allocation of microthrusters for space debris removal using CubeSats,” *Eng. Appl. Artif. Intell.*, vol. 81, pp. 145–156, 2019.

[28] K. Fukushima, “Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition,” 1988.

[29] Y. Lecun, L. Eon Bottou, Y. Bengio, and P. H. Abstract|, “Gradient-Based Learning Applied to Document Recognition.”

[30] M. A. Hedjazi, I. Kourbane, and Y. Genc, “On identifying leaves: A comparison of CNN with classical ML methods,” in *2017 25th Signal Processing and Communications Applications Conference (SIU)*, 2017, pp. 1–4.

[31] H. Wang *et al.*, “Comparison of machine learning methods for classifying mediastinal lymph node metastasis of non-small cell lung cancer from 18F-FDG PET/CT images,” *EJNMMI Res.*, vol. 7, no. 1, 2017, doi: 10.1186/s13550-017-0260-9.

[32] K. T. Islam, R. G. Raj, and A. Al-Murad, “Performance of SVM, CNN, and ANN with BoW, HOG, and Image Pixels in Face Recognition,” *2nd Int. Conf. Electr. Electron. Eng. ICEEE 2017*, no. December, pp. 1–4, 2018, doi: 10.1109/CEEE.2017.8412925.

[33] N. J. L. Marfu’ah and A. Kurniawardhani, “Comparison of CNN and SVM for Ship Detection in Satellite Imagery,” *AUTOMATA*, vol. 1, no. 1, Jan. 2020, Accessed: Jul. 22, 2020. [Online]. Available: https://journal.uii.ac.id/AUTOMATA/article/view/13973.

[34] B. Kumar, O. Dikshit, A. Gupta, and M. K. Singh, “Feature extraction for hyperspectral image classification: a review,” *Int. J. Remote Sens.*, vol. 41, no. 16, pp. 6248–6287, Aug. 2020, doi: 10.1080/01431161.2020.1736732.

[35] L. Ma, Y. Liu, X. Zhang, Y. Ye, G. Yin, and B. A. Johnson, “Deep learning in remote sensing applications: A meta-analysis and review,” *ISPRS J. Photogramm. Remote Sens.*, vol. 152, pp. 166–177, Jun. 2019, doi: 10.1016/j.isprsjprs.2019.04.015.

[36] H. Parikh, S. Patel, and V. Patel, “Classification of SAR and PolSAR images using deep learning: a review,” *Int. J. Image Data Fusion*, vol. 11, no. 1, pp. 1–32, Jan. 2020, doi:

10.1080/19479832.2019.1655489.

[37] A. Ozdemir and K. Polat, “Deep Learning Applications for Hyperspectral Imaging: A Systematic Review,” *J. Inst. Electron. Comput.*, vol. 2, no. 1, pp. 39–56, Feb. 2020, doi: 10.33969/jiec.2020.21004.

[38] Y. Zhang and S. Prasad, “Locality Preserving Composite Kernel Feature Extraction for Multi-Source Geospatial Image Analysis,” *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 8, no. 3, pp. 1385–1392, Mar. 2015, doi: 10.1109/JSTARS.2014.2348537.

[39] N. Buonaiuto *et al.*, “SSC17-WK-56 Satellite Identification Imaging for Small Satellites Using NVIDIA,” *Small Satell. Conf.*, 2017.

[40] A. P. Arechiga, A. J. Michaels, and J. T. Black, “Onboard Image Processing for Small Satellites,” in *Proceedings of the IEEE National Aerospace Electronics Conference, NAECON*, 2018, vol. 2018-July, pp. 234–240, doi: 10.1109/NAECON.2018.8556744.

[41] J. Manning *et al.*, “Machine-Learning Space Applications on SmallSat Platforms with TensorFlow,” *32nd Annu. AIAA/USU Conf. Small Satell.*, pp. 1–8, 2018, [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-.

[42] S. Greenland, M. Ireland, C. Kobayashi, P. Mendham, M. Post, and D. White, “Design & Prototyping of a Minaturised Forwards Looking Imager using Deep Learning for Responsive Onboard Operations,” in *The 4S Symposium*, 2018, pp. 1–9.

[43] M. I. Bappy and S. Siddique, “AI-OBC : Conceptual Design of a Deep Neural Network based Next Generation Onboard Computing Architecture for Satellite Systems,” no. March, 2019.

[44] A. D. Braun, “Investigation of Deep Neural Network Image Processing for Cubesat Size Satellites,” 2018.

[45] Z. Zhang, G. Xu, and J. Song, “CubeSat cloud detection based on JPEG2000 compression and deep learning,” *Adv. Mech. Eng.*, vol. 10, no. 10, pp. 1–10, 2018, doi: 10.1177/1687814018808178.

[46] G. Giuffrida *et al.*, “CloudScout: A Deep Neural Network for On-Board Cloud Detection on Hyperspectral Images,” *Remote Sens.*, vol. 12, no. 14, p. 2205, Jul. 2020, doi: 10.3390/rs12142205.

[47] W. S. Mcculloch and W. Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity.,” 1943.

[48] “Handbook of Neural Computation - 1st Edition.” https://www.elsevier.com/books/handbook-of-neural-computation/samui/978-0-12-811318-9 (accessed Jun. 18, 2020).

[49] F. Rosenblatt, “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,” 1958.

[50] M. Minsky and S. Papert, *Perceptrons; an introduction to computational geometry*. MIT Press, 1969.

[51] O. Zapletal, “Image Recognition by Convolutional Neural Networks - Basic Concepts,” 2016.

[52] “The Perceptron - Jonty Sinai.” https://jontysinai.github.io/jekyll/update/2017/11/11/the-perceptron.html (accessed Jun. 18, 2020).

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, doi: 10.1038/323533a0.

[54] H. Robbins and S. Monro, “A Stochastic Approximation Method,” *Ann. Math. Stat.*, vol. 22,

no. 3, pp. 400–407, Sep. 1951, doi: 10.1214/aoms/1177729586.

[55] P. J. Werbos, “Beyond regression: new tools for prediction and analysis in the behavioral sciences,” 1974.

[56] “Yann LeCun’s Home Page.” http://yann.lecun.com/ (accessed Jun. 18, 2020).

[57] Yann LeCun, “A Theoretical Framework for Back-Propagation,” in *Proceedings of 1998 Connectionist Models Summer School*, 1988, pp. 21–28.

[58] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *J. Physiol.*, vol. 160, no. 1, pp. 106–154, Jan. 1962, doi: 10.1113/jphysiol.1962.sp006837.

[59] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “Training algorithm for optimal margin classifiers,” in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, 1992, pp. 144–152, doi: 10.1145/130385.130401.

[60] G. E. Hinton, S. Osindero, and Y. W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, May 2006, doi: 10.1162/neco.2006.18.7.1527.

[61] “Google Trends.” https://trends.google.com/trends/ (accessed Jun. 18, 2020).

[62] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[63] “ImageNet - Wikipedia.” https://en.wikipedia.org/wiki/ImageNet (accessed Jun. 18, 2020).

[64] “From Fiction to Reality: A Beginner’s Guide to Artificial Neural Networks.” https://towardsdatascience.com/from-fiction-to-reality-a-beginners-guide-to-artificial-neural-networks-d0411777571b (accessed Jun. 19, 2020).

[65] Anna Gummeson, “Prostate Cancer Classification using Convolutional Neural Networks,” Lund University, Lund, 2016.

[66] J. Wu, “Convolutional Neural Networks.” https://cs.nju.edu.cn/wujx/teaching/15_CNN.pdf (accessed Aug. 26, 2020).

[67] C. Enyinna Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of Trends in Practice and Research for Deep Learning.”

[68] S. Sun, Z. Cao, H. Zhu, and J. Zhao, “A Survey of Optimization Methods from a Machine Learning Perspective,” *IEEE Trans. Cybern.*, pp. 1–14, Jun. 2019, Accessed: Jun. 19, 2020. [Online]. Available: http://arxiv.org/abs/1906.06821.

[69] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” Dec. 2015.

[70] “Backpropagation In Convolutional Neural Networks | DeepGrid.” https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/ (accessed Sep. 25, 2020).

[71] “PolySat (@PolySat) / Twitter.” https://twitter.com/polysat (accessed Jun. 18, 2020).

[72] “Profile / Twitter.” https://twitter.com/MYSAT_1 (accessed Jun. 18, 2020).

[73] “Chinese smartphone camera photographs Earth from space | Space.” https://www.space.com/chinese-smartphone-camera-images-earth-from-space.html (accessed Jun. 18, 2020).

[74] “Space Station Videos | NASA.” https://www.nasa.gov/mission_pages/station/videos/index.html (accessed Jun. 18, 2020).

[75] “pytube3 — pytube3 9.6.4 documentation.” https://python-pytube.readthedocs.io/en/latest/ (accessed Jun. 18, 2020).

[76] “Welcome To Colaboratory - Colaboratory.” https://colab.research.google.com/ (accessed Jun. 18, 2020).

[77] “youtube_python_3.py.” https://gist.github.com/erykml/6a1fe38763664567e6052e78e047ebb5 (accessed Jun. 18, 2020).

[78] “Sentinel Hub EO Browser.” https://apps.sentinel-hub.com/eo-browser/ (accessed Jun. 18, 2020).

[79] “Resize, convert, split, crop your images online - ImageSplitter.” https://postcron.com/image-splitter/en/ (accessed Jun. 18, 2020).

[80] “image-slicer · PyPI.” https://pypi.org/project/image-slicer/ (accessed Jun. 18, 2020).

[81] M. D. Bloice, P. M. Roth, and A. Holzinger, “Biomedical image augmentation using Augmentor,” *Bioinformatics*, 2019.

[82] “The State of Machine Learning Frameworks in 2019.” https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/ (accessed Jun. 18, 2020).

[83] “5 Most Popular Machine Learning Libraries in Python.” https://yourstory.com/mystory/5-most-popular-machine-learning-libraries-in-pytho-ws215d0wec (accessed Jun. 18, 2020).

[84] “Effective TensorFlow 2 | TensorFlow Core.” https://www.tensorflow.org/guide/effective_tf2 (accessed Jun. 18, 2020).

[85] “O’Reilly AI Conf.” https://www.slideshare.net/neiltan2/oreilly-ai-conf (accessed Jun. 18, 2020).

[86] AspenCore Global Media, “2017 Embedded Markets Study,” 2017. [Online]. Available: http://m.eet.com/media/1246048/2017-embedded-market-study.pdf.

[87] R. Quinnell, “Embedded Markets Study: Changes in Today’s Design, Development & Processing Environments,” 2015.

[88] NanoAvionics, “High-performance Multi-purpose 6U nano-satellite Platform,” 2018. [Online]. Available: https://nanoavionics.com/wp-content/uploads/2018/08/NanoAvionics-M6P-Platform-Brochure-Website.pdf.

[89] P. Liashchynskyi and P. Liashchynskyi, “Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS,” *arXiv Prepr. arXiv1912.06059*, 2019.

[90] albertbup, “A Python implementation of Deep Belief Networks built upon NumPy and TensorFlow with scikit-learn compatibility,” 2017. https://github.com/albertbup/deep-belief-network (accessed Jul. 22, 2020).

[91] ardamavi, “Using Autoencoders for classification as unsupervised machine learning algorithms with Deep Learning.,” 2018. https://github.com/ardamavi/Unsupervised-Classification-with-Autoencoder (accessed Jul. 22, 2020).