Abstract

CubeSat is a nano satellite with mass less than 10 kg. CubeSat due to its limitations on size, power and downlink capabilities, it cannot send the data quickly. The amount of information received in ground station is low compared with bigger satellite. If the satellite sends a bad data type, then the amount of time and efficiency of overall system will be reduced. So, A model which can classify the type of images will be very helpful. Feature extraction from images shows supervised techniques provide better accuracy than unsupervised techniques. CNN is the most popular algorithm in deep learning for remote sensing. CNN has higher overall accuracy and kappa values. The downside of CNN is that the training requires a very large dataset. CNN could cut down operation time by about 2/3 while significantly improving the quality of received data.

Keywords: Neural Network, CNN, Ground Station, nano satellite, Hidden Layer.

List of Figures

Abbreviations

Short form	Full Form
SVM	Support Vector Machine
ML	Machine Learning
CNN	Convolution Neural Network
AI	Artificial Intelligence
GSD	Ground Sampling Distance
UHF	Ultra-High Frequency
MCU	Micro Controller
GS	Ground Station
NN	Neural Network
ICU	Image Classification Unit
NS	Nano Satellite

Table of Contents

Chapter 1: Introduction

1.1. Background

CubeSats are a form of nano satellite which measures less than 10kg. There are thousands of nano satellites in orbit around the earth. CubeSats have enabled educational institutions, small commercial entities and countries with limited to resources to build and launch very small satellites piggybacked as secondary payloads to space. The power consumption of CubeSat is less. There are solar panels but the power is lost during storing and supplying energy. CubeSat has capability to downlink 700 kB per orbit at maximum. In a day, this is equivalent to 10 MB which is the theoretical limit of the amount of data downlinked per day assuming continuous communication with 100% communication window. Due to communication window becoming 35-45 minutes per day, CubeSat downlinks 25–30 kB of real data per day per satellite which is far less than the theoretical limit of 1MB per day. An eight-bit RAW RGB image for 640 × 480 has 307.2 kB of data. When used compression, the compressed data is anywhere between 7 kB to 74 kB which is between 2.3% to 24.1% of the original size. So, this is why it is necessary to classify the quality of images.

We can use various models like SVM, kNN etc, for classification of images but CNN is the most popular algorithm for remote sensing. CNN has higher overall accuracy and kappa values. The downside of CNN is that the training requires a very large dataset. CNN could cut down operation time by about 2/3 while significantly improving the quality of received data.

1.2. Objectives

The primary objective is to classify satellite images.

1. To classify good and bad images recorded by the satellite.
2. To increase the efficiency of nano satellites.

3. To collect valuable information about different types of images that a nano satellite can record with its limited capacity.
4. To study the recording pattern of satellite and train them to take better pictures time ahead.

1.3. Motivation and Significance

Recently Nepal launched its own satellite called NepaliSat-1 which was a satellite in collaboration with Nepal, Raavana-1 from Sri Lanka and Uguisu from Japan. The satellites have been operational since June 2019. The NepaliSat-1 is a nano satellite which was equipped with a simple RGB camera to take images from space. The satellite had limited capacity like downlink limited to 4800 bps in the amateur Ultra High Frequency (UHF) band with 35–45 min of communication window with one ground station (GS) per day. The image highest image resolution of satellite is limited to VGA (640 × 480).

The mission of the nano satellite was to take images mainly for and media purpose. So, the image should look good. The Nepalese satellite have active attitude stabilization but no control so there is no definite way of knowing whether the camera is pointed to space or the Earth. Only thirty images were captured in its first three months of operation due to the limitation of satellite communication window. In those thirty images only twelve images were good images which faced earth. Rest of the images were bad which is almost two third of the images captured. So, in its limited amount of operation, the efficiency was only one third of its capacity.

1.4. Image classification

The satellite is a nano satellite with limited hardware capability. The image highest image resolution of satellite is limited to VGA (640 × 480) and due to communication window being very short, the amount of information transmitted to Ground Station is also limited. If the satellite is not able to classify good and bad images from within the system itself then the efficiency of overall system becomes very low.

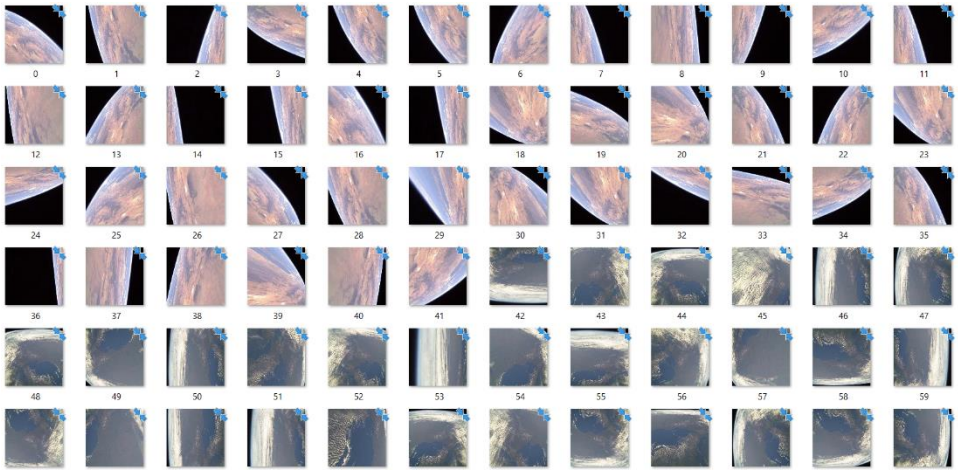


Figure 1: Images captured by satellite

The good images are the images captured by satellite which is tilted more towards the earth rather than a vacant space. Our satellite might cover a whole area of an image capturing the planet or only partially fill the earth in the image. The part where our satellite captures at least two thirds of the area in image of earth is regarded as good, and covering hundred percent of image area with earth is the best. Due to curvature of earth, the probability of horizon in the image is high. The curvature area should be calculated to determine the quality of image taken. Here are some images which is considered a good image.

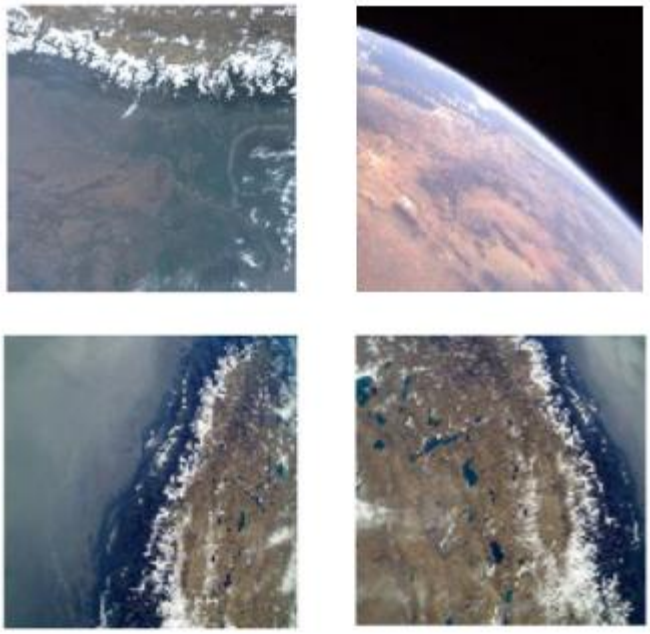


Figure 2: Good Images Example

Most of the pictures captured by satellite is bad image. In Nepal Sat out of thirty images, eighteen were classified bad by professionals. This means about two thirds of images will be bad images in general. Bad images are the one which shows outer space in most part of images rather than our earth. Even though the image shows the planet area more, there are some factors which makes an image bad like blur or too bright image due to position of satellite and sun and the other most important factor is the presence of cloud and fog. The probability of having cloud in images is high. Some clouds do not make images necessarily bad but sometimes cloud block whole vision of planet creating a bad image where detail cannot be identified. Some bad images examples are listed below.

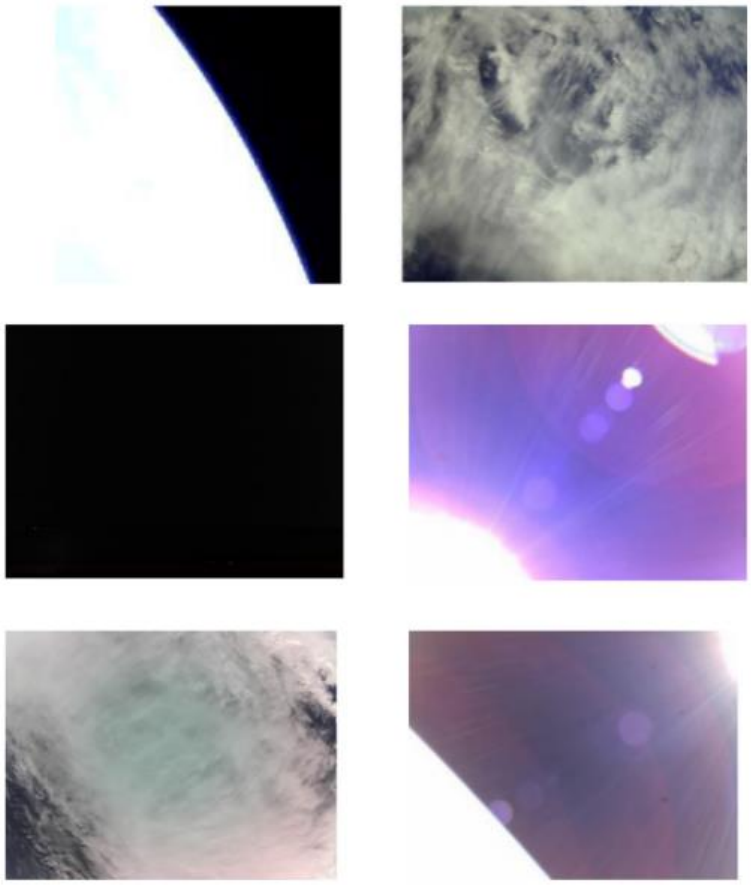


Figure 3: Bad images example

Chapter 2: Design and Implementation

2.1. System Requirement Specification

2.1.1. Software Specification

2.1.1.1. ML Library

a. NumPy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

b. OS:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. It provides functions to operate with the operating system.

c. CV2:

CV2 is a huge open-source library of Python for computer vision, machine learning, and image processing.

d. Pandas:

Pandas is a python package to make working with relational and labeled data intuitively. Pandas provides fast, flexible, and expressive data structures and becomes fundamental high-level building block for doing practical, real-world data analysis in Python.

e. Matplotlib:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

f. TensorFlow:

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks.

g. keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. t wraps the efficient numerical computation libraries Theano and TensorFlow and allows to define and train neural network models in just a few lines of code.

2.1.1.2. IDE

We have used web IDE for our project.

a. Kaggle:

Kaggle is an online community of data scientists and machine learning practitioners which allows users to find and publish data sets, explore and build models in a web-based data-science environment. Kaggle offers a no-setup, customizable, Jupyter Notebooks environment. Access free GPUs and a huge repository of community published data & code.

b. Google Colab:

Google Colab is a web IDE that allows to write and execute arbitrary python code through the browser. Google Colab well suited to machine learning, data analysis and education. It is a hosted Jupyter notebook that requires no setup and has an excellent free version, which gives free access to Google computing resources such as GPUs and TPUs.

2.2. System Design

2.2.1 Procedure

STEP 1: Dataset Collection

- Professor and technical person working in nano satellite and cube satellite was contacted.
- Find the reliable data.
- Find enough data to train the module.

STEP 2: Classify Data with expert supervision

- Under supervision of professor or experts the images were classified based on good or bad record.
- The good images are the one which is tilted toward earth and bad images are titled in space.

STEP 3: Use ML Packages to create suitable Training Environment

- Different ML libraries were used such as pandas, os, matplot, cv2, pickle, pa and TensorFlow, Keras.
- Use of different Platform for training data such as Collab, Kaggle and Jupyter Notebook.

STEP 4: Model Training until required accuracy is achieve

- Tuning of hyperparameters to achieve higher accuracy.
- Using different models such as Deep NN, CNN to acquire higher accuracy.

STEP 5: Test and Debug the application

- Testing on Test Set.
- Testing on Validation Set.

2.2.1 Flow Chart

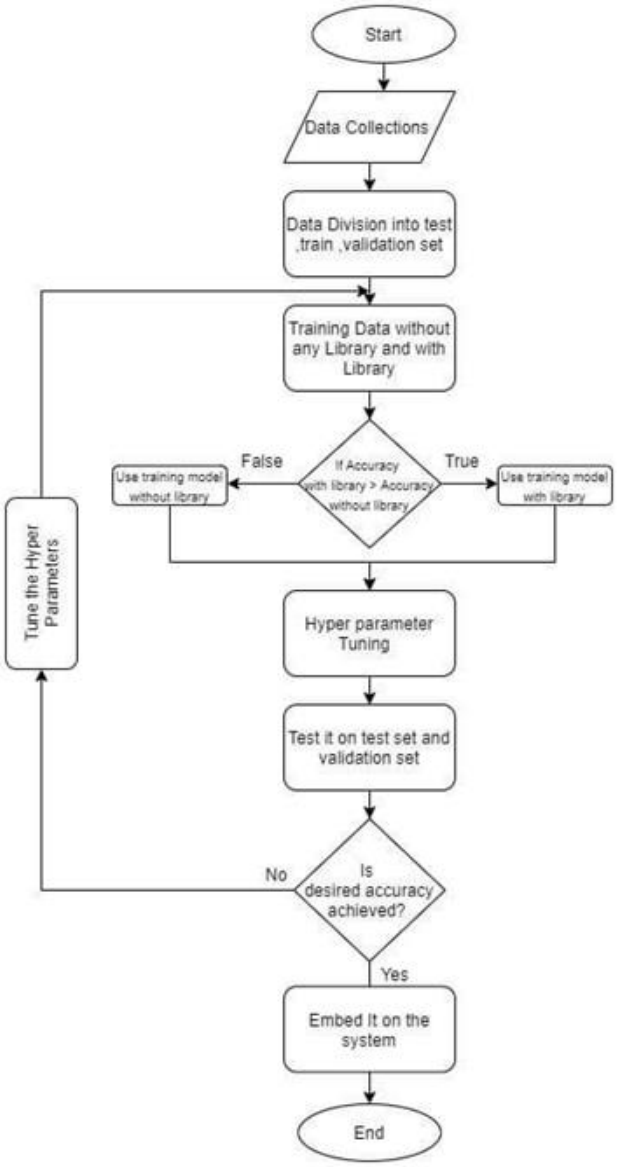


Figure 4: Flow Chart

2.2.1 Work Flow Diagram

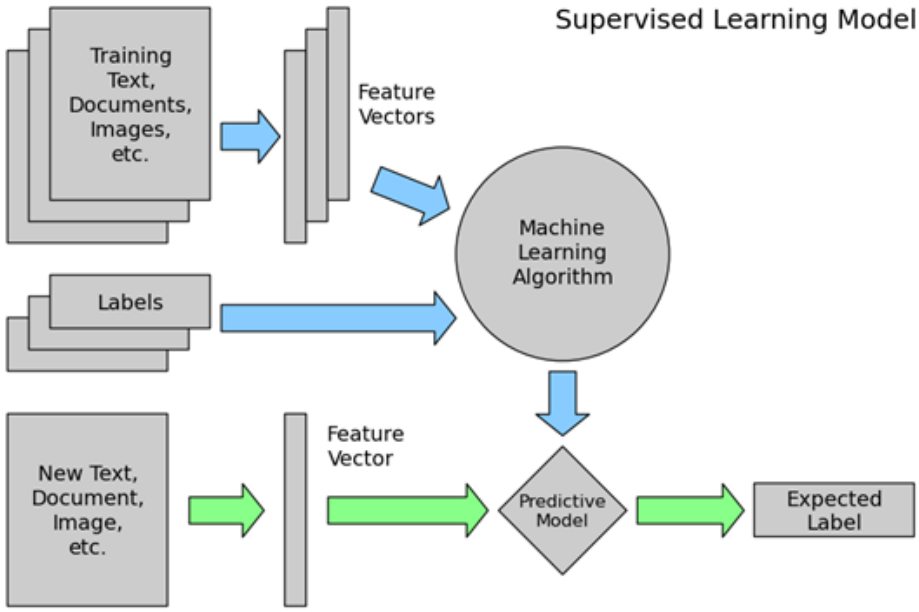


Figure 5: Work Flow Diagram

2.3 PEAS

Performance

Correct Image Categorization, shift position to capture more planet.

Environment

Orbit, Space, Upper Atmosphere, low earth orbit, space area, Ground Station, Downlink from orbiting satellite,.

Actuators

Display of scene categorization, Detect the environment and change phases for example if the amount of space is more than shift to a position where planet is completely seen in the image, Booster boosting the position, Solar panel changing position, microcontroller classifying images.

Sensor

Camera, Image Sensor, accelerometer, temperature sensor, light sensors, angle sensors, Color pixel Arrays.

Chapter 3: Discussion on the Model

3.1. Two Layer Deep Neural Network

- The model's structure is: *LINEAR -> RELU -> LINEAR -> SIGMOID*.
- The input is a (100,100,1) image which is flattened to a vector of size $(10000,1)$.
- he corresponding vector: $[x_0,x_1,...,x_{9999}]^T$ is then multiplied by the weight matrix $W^{[1]}$ of size $(n^{[1]},9999)$.
- Then, adding a bias term and take its Relu to get the following vector: $[a^{[1]}_0,a^{[1]}_1,...,a^{[1]}_{n^{[1]}-1}]^T$.
- Repeating the same process.
- Multiplying the resulting vector by $W^{[2]}$and add the intercept (bias).
- Finally, taking the sigmoid of the result. If it's greater than 0.5, classify it as a good image.

System Architecture:

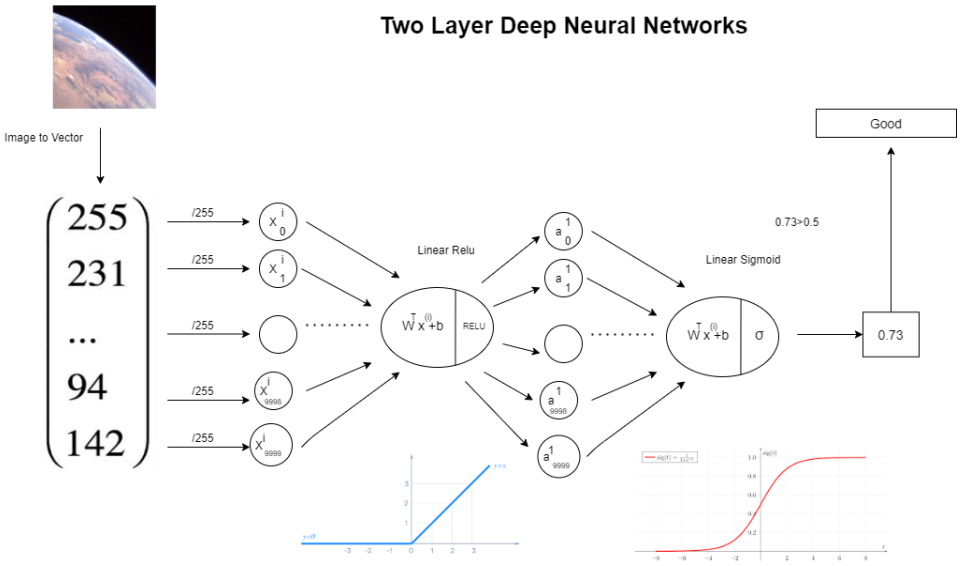


Figure 6: System Architecture for Deep Neural Network

Working Model:

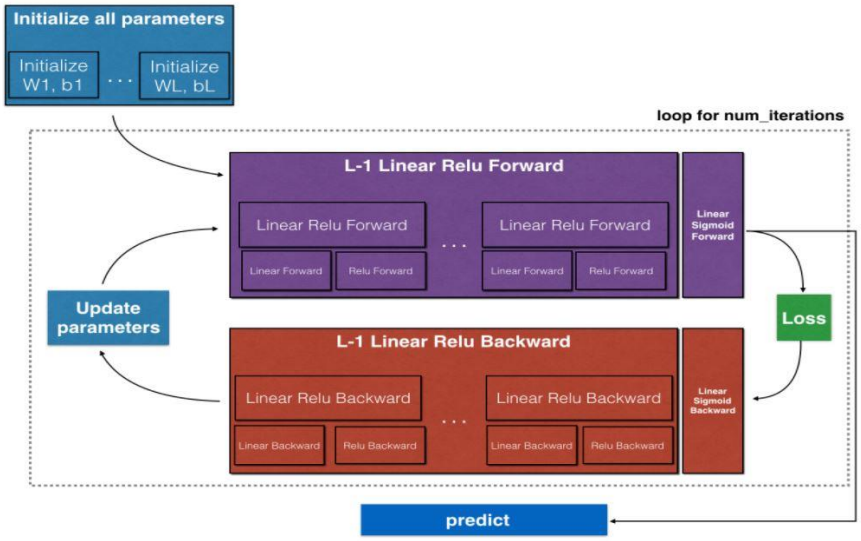


Figure 7: Working Model

Initialize Parameter

Linear Activation Forward

Implement `forward propagation` using the following equations:

$$Z_{[1]}=W_{[1]}X+b_{[1]} \quad (1)$$

$$A_{[1]}=\tanh(Z_{[1]}) \quad (2)$$

$$Z_{[2]}=W_{[2]}A_{[1]}+b_{[2]} \quad (3)$$

$$\hat{y}=A_{[2]}=\sigma(Z_{[2]}) \quad (4)$$

Compute Cost:

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right)+(1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right)$$

The following formula is implemented during the training process to compute the cost.

Linear Activation Backward

$dz^{[2]} = a^{[2]} - y$	$dZ^{[2]} = A^{[2]} - Y$
$dW^{[2]} = dz^{[2]} a^{[1]^T}$	$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$
$db^{[2]} = dz^{[2]}$	$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$	$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$
$dW^{[1]} = dz^{[1]} x^T$	$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$
$db^{[1]} = dz^{[1]}$	$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

Backpropagation is usually the hardest (most mathematical) part in deep learning. We have six equations on the left slide, since we are building a vectorized implementation we have equations in vector form in right side.

Update Parameters using Gradient Descent Algorithm

W1 = W1 - learning_rate * dW1

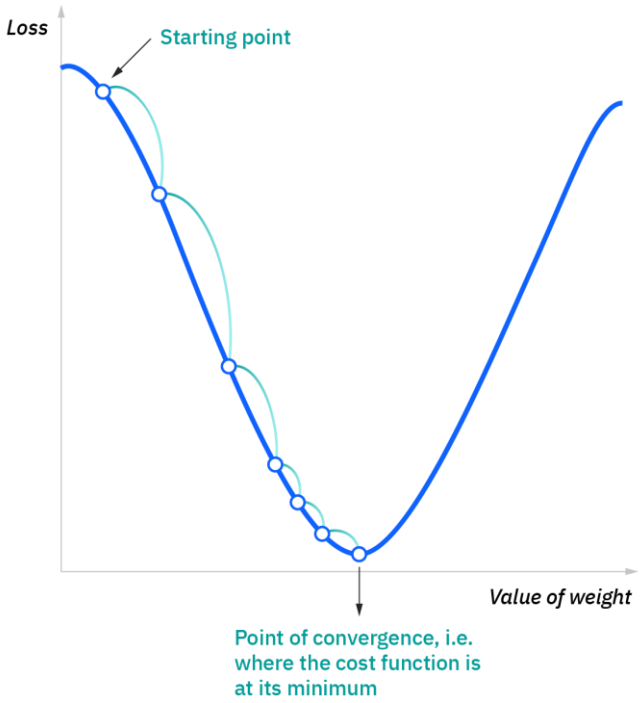
b1 = b1 - learning_rate * db1

W2 = W2 - learning_rate * dW2 b2 = b2 - learning_rate * db2

Each weight is subtracted from the gradients times the learning rate and are updated towards the new weight. Here the learning rate defines how fast the model is shifting its value to local optima.

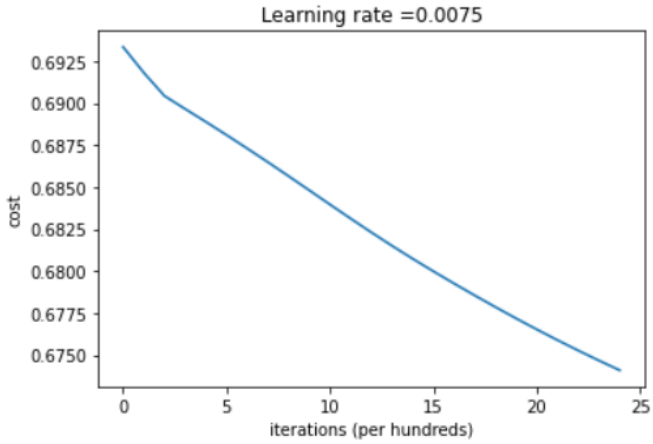
Algorithm for optimization:

Gradient Descent Algorithm



Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error. Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.

Result:



train accuracy: 57.35443400400844 %test accuracy: 57.73779776778277 %

3.2 Convolution Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

System Architecture:

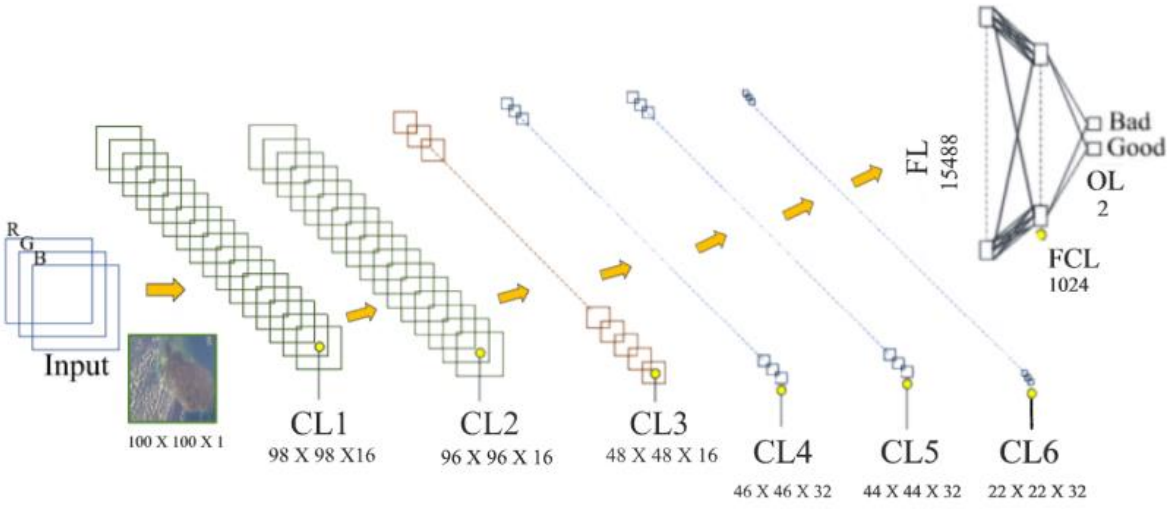


Figure 8: System Architecture for Convolution Neural Network

Working Model:

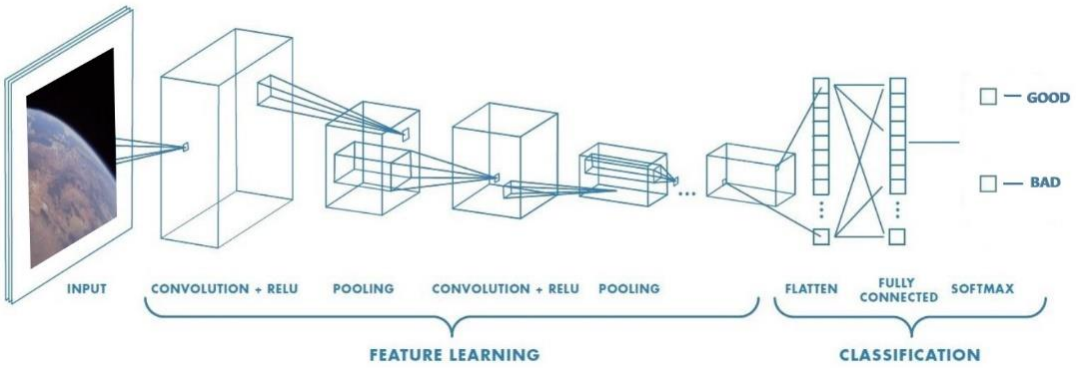
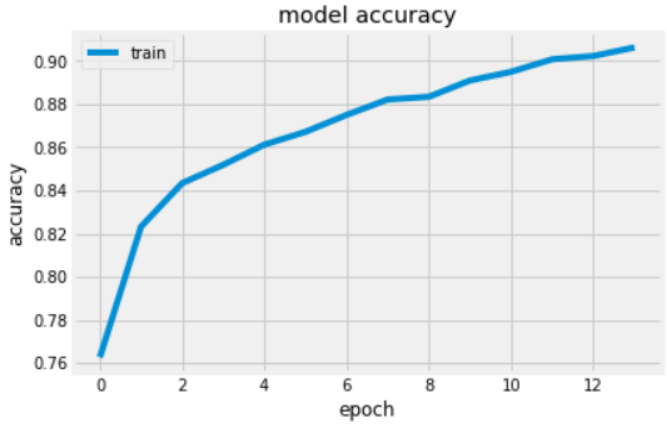
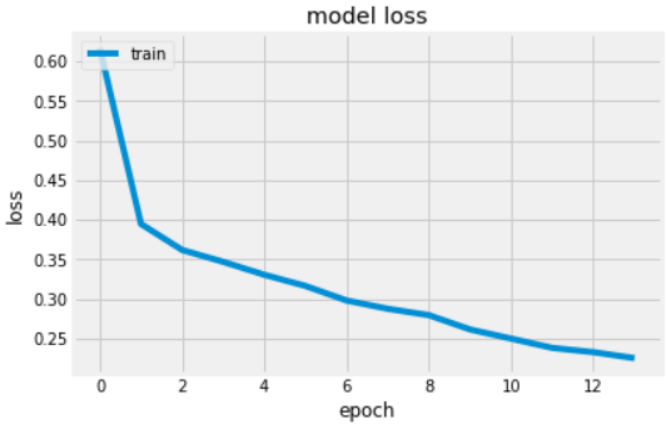


Figure 9: Working Model

The image is converted into the array and then passes through a layer of filters so that the features of the images could be calculated. First the image is passed through the layer of convolution network with relu activation function and then normalized and again fed into the convolution neural network which is later maxpooled . This process is repeated again till the size of image becomes 22*22*32. This image is now flattened resulting the shape of 15488. Then it is fed to two layers of dense layer resulting the final shape of 1024. During this entire training process, the no of total parameters are 8,474,290 out of which 8,474,290 is trainable and 192 is non- trainable parameter as given by our model.

Result:





Train Accuracy: 90.31

Test Accuracy: 83.38

3.3 Activation Functions:

Activation Function decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored is determined by the activation function. The activation function is the

nonlinear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.

Relu

The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. The main advantage of using the RELU function over other activation functions is that it does not activate all the neurons at the same time. With ReLU, there's less chance of having a vanishing gradient problem. For hidden layer ReLu should be used.

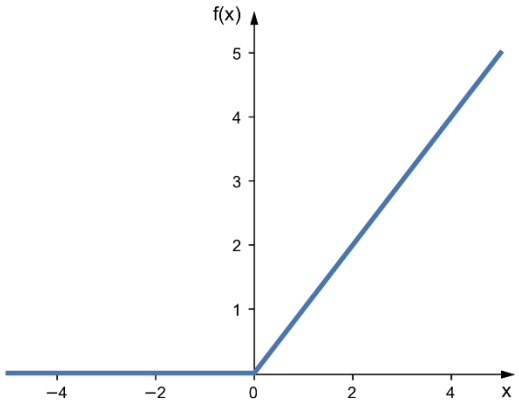


Figure 10: ReLu Activation

Sigmoid

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve. Sigmoid functions and their combinations generally work better in the case of classifiers. A sigmoid function is monotonic, and has a first derivative which is bell shaped. Its function is:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1} = 1 - S(-x).$$

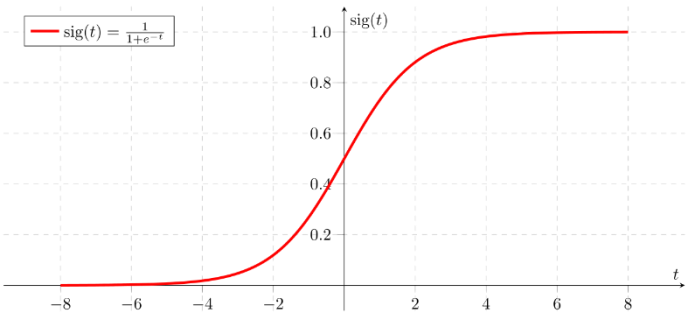


Figure 11: Sigmoid Function

Softmax

Softmax activation function is an auto-normalizing function on vectors: it maps a vector of unconstrained real numbers into a vector of small values (0..1) that sum to one. The output of softmax is probabilities of a set of categories but is not the intrinsic property of the softmax function, but it's a very useful interpretation to optimize a loss based on categorical cross-entropy.

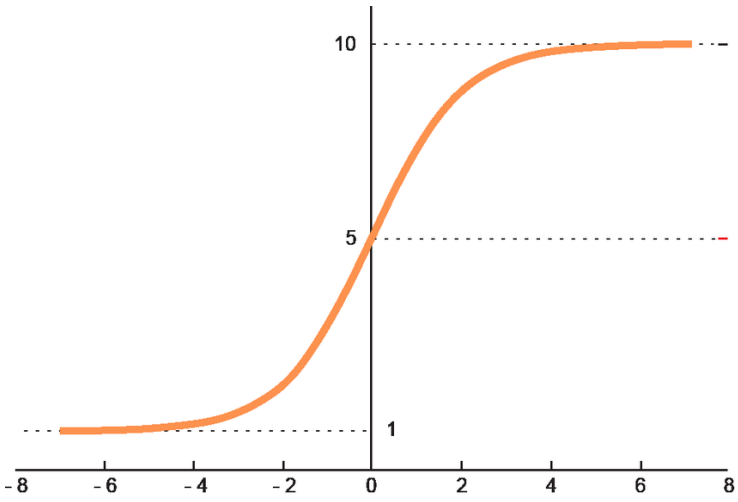


Figure 12: Softmax Activation

Results:

Hence, our model was able to classify good and bad images with 90 percent accuracy which means if our model were to be deployed in nano satellite than it would significantly improve the efficiency of satellite by transmitted majority of good images. From just one third of the images being good to over eighty percent images labeled good is a work efficiently done. Here in the picture below our model was able to properly classify between good and bad images.

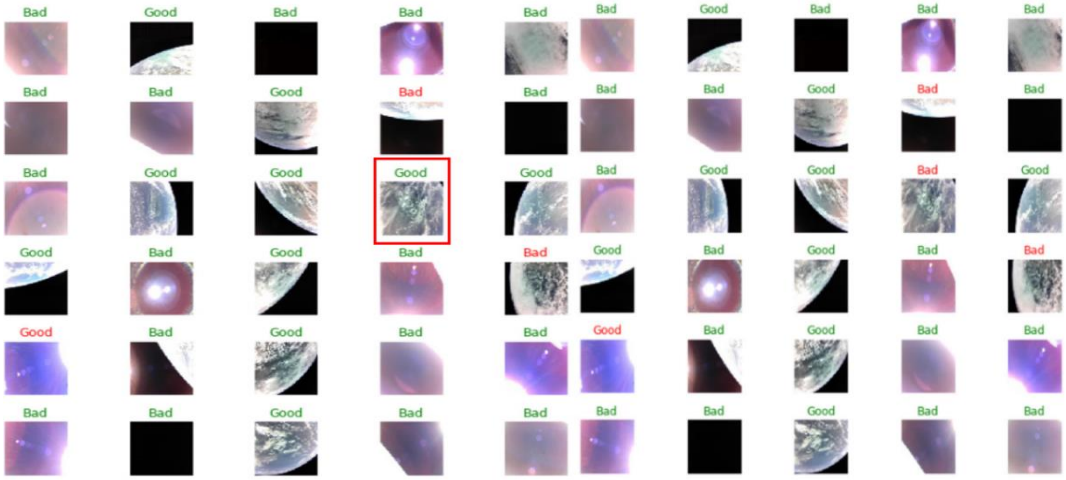


Figure 13: Results made by Convolution Neural Network

Chapter 4: Conclusion and Recommendation

In this way, Satellite Image Classification model was built. We collected the data set by requesting data from Professor Sergion Montenegro from University of Wurzburg and with professionals from NepalSat-1. We used the training environment of Kaggle, Collab and Jupyter Notebook using gradient descent algorithm, logistic regression, convolution model, neural network with hidden layer and CNN. We got our accuracy to be 57% train accuracy and 57% test accuracy from two-layer neural network using gradient descent algorithm and 90.31% train accuracy and 83.38% test accuracy deep learning with hidden layer. We did our best and worked for three months to create this model but due to time constraints, there are certain limitations in our model which we certainly will improve in future.

4.1. Limitations

We did our best and built a good model. However, there are some constraints of the project. Some of which are:

- The model is not implemented in a real nano satellite, the model should be supported by micro controllers of the satellite in order for the model to work and give value out of it.
- The model might be heavy for small electronic component used in nano satellites.
- Test accuracy is 83 percent which means still the model might not classify 17 percent of images properly which can be crucial for nano satellite due to its very short communication window.

4.2. Future Enhancements

Considering the limitations of the model, we will improve our model and make enhancements to the model based on practical models and deploy it in nano satellite.

- Increase the accuracy of the model.

- Decrease the size of model to support in microcontrollers of a nano satellite.
- Deploy model in a real satellite to test the practicality of the model in nano satellites.

Other Figures:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 16)        160
_________________________________________________________________
batch_normalization (BatchNo (None, 98, 98, 16)        64
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 96, 96, 16)        2320
_________________________________________________________________
batch_normalization_1 (Batch (None, 96, 96, 16)        64
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 48, 48, 16)        0
_________________________________________________________________
dropout (Dropout)            (None, 48, 48, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 32)        4640
_________________________________________________________________
batch_normalization_2 (Batch (None, 46, 46, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 44, 44, 32)        9248
_________________________________________________________________
batch_normalization_3 (Batch (None, 44, 44, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 22, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 22, 22, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 15488)             0
_________________________________________________________________
dense (Dense)                (None, 512)               7930368
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              525312
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 2050
=================================================================
Total params: 8,474,482
Trainable params: 8,474,290
Non-trainable params: 192
_________________________________________________________________
```

Figure 14: Other figures

```
Cost after iteration 0: 0.6933410660441872
Cost after iteration 100: 0.6918076753674155
Cost after iteration 200: 0.6904253529435009
Cost after iteration 300: 0.6896488219267791
Cost after iteration 400: 0.6888823765267719
Cost after iteration 500: 0.688096888845412
Cost after iteration 600: 0.6872949641549027
Cost after iteration 700: 0.6864783782919176
Cost after iteration 800: 0.6856458623388668
Cost after iteration 900: 0.684802005882342
Cost after iteration 1000: 0.6839565162465303
Cost after iteration 1100: 0.6831183631181078
Cost after iteration 1200: 0.6822962701259877
Cost after iteration 1300: 0.6814957819724272
Cost after iteration 1400: 0.6807226935815962
Cost after iteration 1500: 0.6799751807116567
Cost after iteration 1600: 0.6792481130693213
Cost after iteration 1700: 0.6785386614759781
Cost after iteration 1800: 0.6778460456581586
Cost after iteration 1900: 0.6771691401658051
Cost after iteration 2000: 0.6765129864149496
Cost after iteration 2100: 0.6758758185477379
Cost after iteration 2200: 0.675258926068598
Cost after iteration 2300: 0.6746618686931516
Cost after iteration 2400: 0.6740855521366604
```

```
Epoch 1/15
1501/1501 [==============================] - 673s 448ms/step - loss: 1.0145 - accuracy: 0.7099
Epoch 2/15
1501/1501 [==============================] - 673s 448ms/step - loss: 0.4096 - accuracy: 0.8183
Epoch 3/15
1501/1501 [==============================] - 674s 449ms/step - loss: 0.3647 - accuracy: 0.8424
Epoch 4/15
1501/1501 [==============================] - 678s 452ms/step - loss: 0.3428 - accuracy: 0.8510
Epoch 5/15
1501/1501 [==============================] - 680s 453ms/step - loss: 0.3348 - accuracy: 0.8590
Epoch 6/15
1501/1501 [==============================] - 682s 454ms/step - loss: 0.3182 - accuracy: 0.8633
Epoch 7/15
1501/1501 [==============================] - 684s 456ms/step - loss: 0.3054 - accuracy: 0.8697
Epoch 8/15
1501/1501 [==============================] - 689s 459ms/step - loss: 0.2888 - accuracy: 0.8784
Epoch 9/15
1501/1501 [==============================] - 690s 460ms/step - loss: 0.2819 - accuracy: 0.8811
Epoch 10/15
1501/1501 [==============================] - 681s 454ms/step - loss: 0.2612 - accuracy: 0.8888
Epoch 11/15
1501/1501 [==============================] - 680s 453ms/step - loss: 0.2529 - accuracy: 0.8917
Epoch 12/15
1501/1501 [==============================] - 677s 451ms/step - loss: 0.2524 - accuracy: 0.8933
Epoch 13/15
1501/1501 [==============================] - 679s 452ms/step - loss: 0.2448 - accuracy: 0.8976
Epoch 14/15
1501/1501 [==============================] - 683s 455ms/step - loss: 0.2234 - accuracy: 0.9058
Epoch 15/15
1501/1501 [==============================] - 682s 455ms/step - loss: 0.2289 - accuracy: 0.9031
376/376 [==============================] - 36s 95ms/step - loss: 0.5833 - accuracy: 0.8338

Test accuracy: 0.8338330984115601
```