ISSN: 0976-9102 (ONLINE)
DOI: 10.21917/ijivp.2021.0352

CNN TRANSFER LEARNING FOR DETECTION, COUNTING AND SEGMENTATION OF COCONUT PALMS FROM SATELLITE IMAGES

Niranjan D. Gholba[1], Shefali Agrawal[2] and Arun Babu[3]
Indian Institute of Remote Sensing, Indian Space Research Organisation - Dehradun, India

Abstract

Several Free and Open Source (FOSS) tools use Neural Networks for detection of objects from images and videos captured from hand-held imaging devices. Satellite based Remote Sensing images offer wide area coverage and hold potential for detecting, counting and mapping manmade objects, trees, etc., but, have embedded geospatial information and often have more than three bands. Hence, the existing FOSS tools are not able to directly process Remote Sensing images for Computer Vision (CV) applications. This research aims to devise a methodology to adapt a FOSS CV tool, namely the TensorFlow Object Detection (TFOD) API, for detection, counting and segmentation of coconut palms from satellite images and ascertain if the technique can facilitate automated census of coconut palms. Dataset of coconut palm crowns was custom-created using multi-band images from World View-3 satellite. The images were pan-sharpened, cropped and labelled. SSDLite MobileNet V2 CNN, which was pre-trained on COCO dataset, was subjected to transfer learning using coconut data on Tesla K80x GPU. This re-trained CNN could successfully detect and count coconut palms with F-1 score more than 96 %. Histogram thresholds were used to segment and delineate each detected coconut palm crown with 87 % accuracy. Assessment of relative health status of coconut palms was mapped using the Normalised Difference Red-Edge Index derived from satellite images. This study demonstrated that TFOD API can indeed be adapted for object detection and segmentation from Remote Sensing images, albeit with some limitations.

Keywords:

Computer Vision, Object Detection, Segmentation, TensorFlow Object Detection API, Satellite Image

1. INTRODUCTION

Artificial Intelligence (AI) aims to create systems that mimic human intelligence. AI includes robotics, cognitive modeling, expert systems, heuristic problem solving, knowledge representation, machine learning, etc. [1]. Computer Vision (CV) is a field that combines expertise from Computer Science and Artificial Intelligence and enables a computer to use algorithms for detection, classification, identification, labeling, segmentation of different objects from images and videos.

Object Detection is an important function of CV, which is achieved using a set of algorithms called as Convolutional Neural Networks (CNNs) [2], often classified under the sub-section named Deep Learning [3]. There are several open-source as well as commercial tools for Object Detection. TensorFlow Object Detection Application Programming Interface (TFOD API) [4] is one such open source compendium of algorithms, hosted by Google Inc. It is based on Python backbone and uses the Deep Learning libraries like TensorFlow and Keras. It provides a convenient and modular approach for object detection of 3-band or single band (grayscale) images. TFOD API includes CNNs pre-trained on image datasets like PASCAL VOC, COCO, ImageNet, etc. which comprise of open-source image collections of '.jpg', '.png', '.bmp' class of 3-layered images captured from hand-held or ground-based recording devices. TFOD API has been successfully used to detect dogs using SSD MobileNet V2 CNN from a custom set of image data [5], to detect animal movements from video cameras and initiate alerts signals on Raspberry Pi [6], to detect and count humans [7], to detect vehicle number plates from images using RetinaNet[8], to detect anomalies in medical images using SSD MobileNet V1[9], to detect hand-written mathematical digits and operators [10], to detect playing cards[11], etc.

Object detection using Computer Vision has the potential to detect specific objects like trees, buildings, vehicles, etc, from airborne or satellite images. Though the principles of Computer Vision can be applied for Remote Sensing (RS) applications, there are certain challenges as the RS data is quite different from the conventional images [2]. The Table.1 highlights the differences and also suggests solutions to use conventional CV tools for RS images. Owing to these differences in the image data, TFOD API or any other CV based application cannot directly process RS images for training CNNs, without deliberate customization.

Table.1. Challenges for Object Detection from Remote Sensing images compared to normal images.

Parameter	Ground-based Image	Remote Sensing Image	Suggested Modifications in CV Tools
File size	Small (Kb to Mb)	Very Large (Mb to Gb)	Crop RS image into small pieces
Spectral Information	Generally RGB (3-layer) or grayscale (1-layer)	Panchromatic, LiDAR, Microwave, Multispectral, Thermal, Hyperspectral	Use combinations of any 3 layers at a time and Image Fusion
Common File Formats	jpg/ jpeg, png, bmp, tiff	GeoTiff, img, jp2	2-way conversion between formats
Number of objects per image	Less / limited	High to Very high	Use robust CNNs
No of images available for training	Very high (in millions)	Few (hundreds or lesser)	Data Augmentation
Size of object relative to image dimensions	Large	Small to very small	Zoom-in to create training data

CV techniques hold tremendous potential to use multi-band satellite images to detect, count and segment objects like coconut palm crowns.

Coconut (Cocos nucifera Linnaeus) is a major plantation crop. India ranks among the top three nations in global statistics of acreage, production and productivity of coconut [12]. In India, livelihood of around 12 million people depends on coconut palms [13]. However, there is a vast scope to improve productivity. For knowing the areas requiring attention, a detailed census of coconut palms is essential, that would provide count of the number of palms, area under coconut palms and map of their health status. Manual census is costly, time consuming and prone to human biases and errors. Contrary to this, remote sensing images provide undisputed truth of large areas in short time.

Therefore, this study explores the feasibility of modifying the TensorFlow Object Detection API for aiding object detection and segmentation from satellite images achieving all the above-mentioned facets of the census.

2. MATERIALS AND METHODS

2.1 RESOURCES AND TOOLS

The study area was selected in Shivamogga district in Karnataka state, India measuring 10 square km size. This area is predominant in plantations of coconut as well as arecanut, in addition to a few fruit plantations. Coconut plantations are available in sparse as well as dense spacing. Ground Truth (GT) was collected by visiting the coconut plantations, recording the GPS co-ordinates, ground photos and counting the number of palms. This data was uploaded in Android based forms in ODK Collect App. Using the QRealTime plugin in QGIS software, this data was collated and converted into shapefiles for subsequent use during validation.

High resolution cloud-free images of date 05.01.2018 were obtained from World View-3 commercial satellite of Digital Globe Inc. [14]. They contained a Panchromatic image (01 band with 0.4 m spatial resolution) and a Multispectral image (8 bands each with 1.2 m spatial resolution) covering the visible and Near-Infra Red (NIR) region of the Electro-Magnetic spectrum.

This study was undertaken in Python 3.6 through Anaconda-3 interface. Annotation of coconut palms was done on images using labelImg GUI. TFOD API (models ver 1.12.0) was cloned from GitHub and was activated [5]. CNN training was undertaken on nVIDIA Tesla K80x GPU. The progress of training and validation was monitored real-time using TensorBoard graphical web-based interface [15]. Testing of the CNN was done using the Evaluation Metrics, namely, Precision, Recall and F-1 score [16], [17].

2.2 METHODOLOGY

Pictorial depiction of the methodology devised for this research is depicted in Figures 1 and 2.The training and testing samples have to be essentially separate when dealing with CNNs. If there is an overlap then the testing results depict unnaturally high accuracy. Therefore, water-tight compartments were created out of the study area. The western 65 % of the study area was used to generate training and validation images, while eastern 35 % area was used for test images [18].

Basic Image Processing was undertaken in ERDAS Imagine software. The Panchromatic (Pan) and Multispectral (MX) satellite images were fused using Subtractive Resolution Merge technique to generate Pan-sharpened 8-band MX images giving an output of 0.4 m spatial resolution. The Pan-sharpened images were cropped into square images of 386 x 386 pixel size to ensure feeding a square image into the CNN. They were then converted into '.jpg' format with nil compression. The Pan images were 1-layered, while the pan-sharpened MX images were 3-layered colour-composites with combinations of band numbers 7, 5, 3 (False Colour Composite: NIR, Red, Green) and 5, 3, 2 (Natural Colour Composite: Red, Green, Blue).

The '.jpg' images were opened in 'labelImg' tool, which is a Python based GUI for annotating objects [19] and rectangular boxes were marked on coconut palm crowns on the images. It was ensured that the annotation boxes were compact and covered only the coconut palm crown. Conscious effort was made to minimize the inclusion of adjacent features like background soil or crown shadow inside the annotation box. This would assist the CNN in correctly identifying the coconut palms [18]. The bounding co-ordinates of these annotations were saved as '.xml' files in the PASCAL VOC format.

The images and corresponding '.xml' files were randomly split into training data (65 %) and validation data (35 %). The TFOD API requires the annotation data to be in a customized format called as 'TF Record'. The code for this conversion was adopted from Raccoon Dataset Detector [20]. The '.xml' files were first combined and converted into Comma Separated Value ('.csv') files, namely 'train_labels.csv' and 'val_labels.csv'. Further on, they were converted to TF Records with names, 'train.record' and 'val.record' respectively. Since the object to be detected was coconut crown, the label name was fed as 'coco' in the 'label.pbtxt' file. All the above mentioned files were then stored in a specific directory structure [21] to facilitate TFOD API to access it during training and validation.

The main focus of this study was to develop a workable methodology to address the objectives of this study. Therefore, the amended code and the modified methodology hereafter are explained in the next section along with the corresponding results and discussion.

3. RESULTS AND DISCUSSION

3.1 ALGORITHM TRAINING

Model Zoo [22] of the TensorFlow Object Detection API has pre-trained algorithms. From this, SSDLite MobileNet V2 algorithm was selected due to its better performance [18] and was subjected to Transfer Learning (also known as Fine Tuning) with coconut data from satellite images. Contents of the '.config' file were amended to cater to image dimensions, include data augmentations and activate dropout and L-2regularization (weight: 0.00004) and increase the maximum detections to 500 per image.

During training, it is important to select and save those checkpoints with lowest loss and highest values of evaluation metrics. By default, TFOD API records loss values every 100 epochs, but saves checkpoints every 600 seconds. Also, only last five checkpoints are retained, and all earlier ones are erased from the memory. This created problems in finding the best checkpoint.

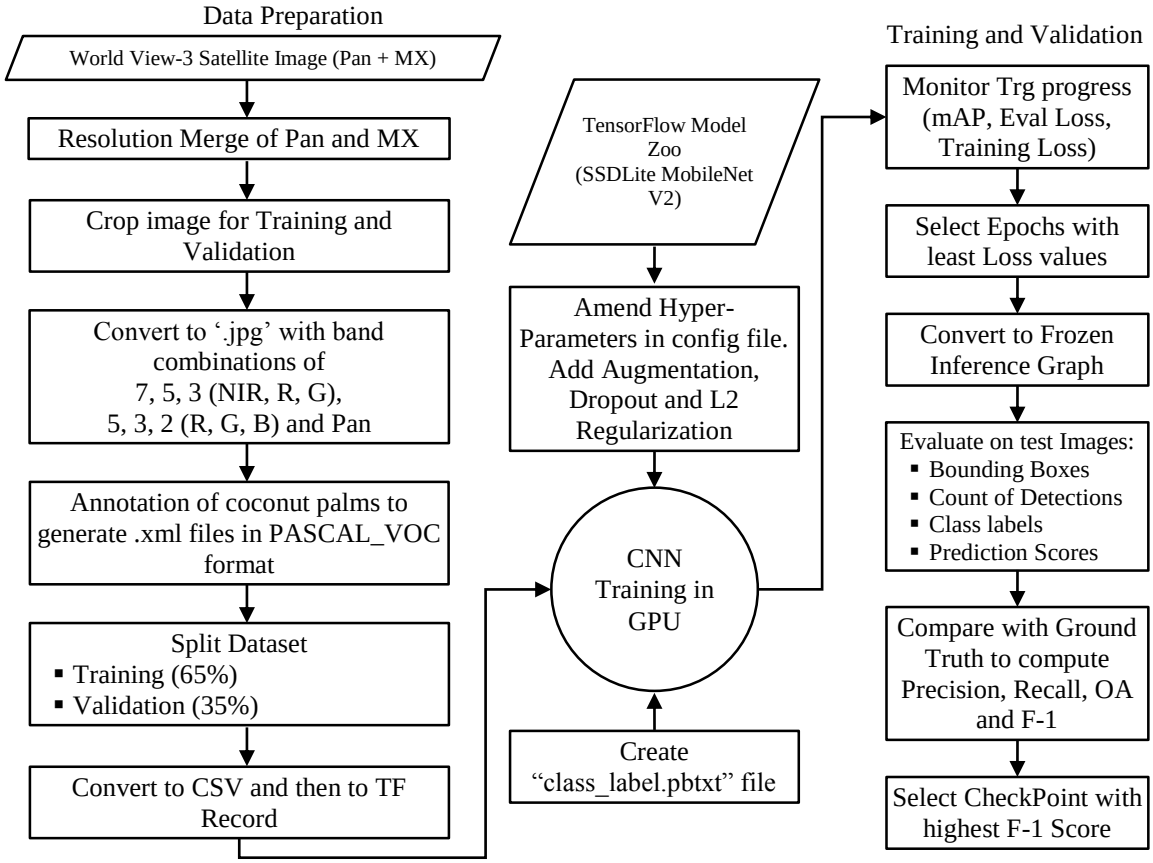


Fig.1. Flow chart of steps followed during Data Preparation, Training and Validation of the CNN

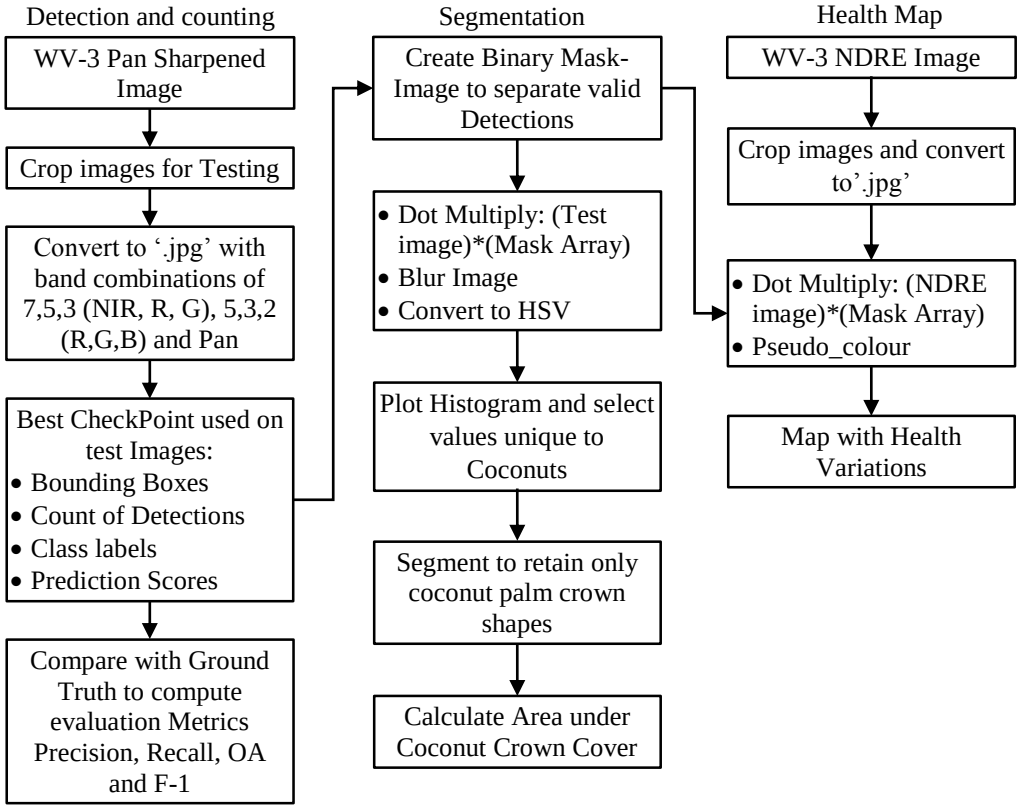


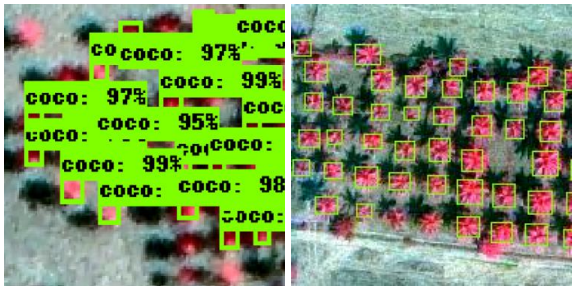
Fig.2. Flow chart of steps followed during testing of algorithm. Output comprises of Coconut Crown Detection, Counting of palms, Segmentation of Crown Cover and computation of Area and generation of Map of Health Variations

From the graph, epoch with lowest loss could be found, but that particular checkpoint was very often not saved. To ensure synchronization between recording of parameters and saving of checkpoints, the code was modified in 'model_main.py' file to enable recording and saving of checkpoints every 100 epochs and all the epochs were saved without overwriting (save_summary_steps=100, save_checkpoints_steps = 100, keep_checkpoint_max = None).The algorithm was trained for 1,00,000 epochs. Several parameters like Mean Average Precision (mAP), Training Loss, Evaluation Loss, Average Recall, etc. were monitored real-time. Checkpoints with low values of training loss were selected and were converted into Frozen Inference Graphs for validation.

3.2 VISUALIZATION AND TESTING OF ALGORITHM

The "object_detection_tutorial.ipynb" notebook in TFOD API was used for visualization of the detected objects.

But, TFOD output has default bounding boxes with thick borders and the large sized labels. Since, coconut palms are closely spaced, such labels caused cluttering (Fig.3 (a)). In order to de-clutter the detections, the line thickness was reduced from 8 to 1. Code was amended to enable the algorithm to draw up to maximum 500 boxes (default value is 100). Also, the default class labels and prediction scores were deactivated to improve the visualization (Fig.3 (b)).



(a) (b)

Fig.3. Visualization of detections. (a) Detections with default settings caused overlapping of labels. (b) De-cluttered detection boxes

Default detections are on 3 layered colour images. (Natural and False Color Composite images). To test the 1-layered Panchromatic images, the code was modified by reshaping the image numpy array [23].

In TFOD API, the output in form of 'detection boxes' gets saved in the memory as an output dictionary referred to as output_dict['detection_boxes'] in the notebook. The corresponding scores of these detections got stored as 'detection scores' in float format in the range of 0 to 1. The number of boxes equal the number of scores. Hence, the total number of boxes generated by the CNN was computed from the number of scores as below.

total_no_of_boxes=output_dict['detection_scores'].size

Jaccard Index, also called as Intersection over Union (IoU) was used to decide whether the detection boxes were valid. The ratio of the common part (intersection) between the ground truth and detection box to the total part (union) is IoU. Its value varies between 0 and 1. When there is no overlap, IoU equals zero. When the overlap is complete and exact, then IoU is 1. Though it is desirable to have IoU values closer to 1, practically, detection boxes having IoU≥0.5 were considered valid. To count the number of valid detections, a counter was used in a loop as below.

count=0
  for *j* in range(*total_no_of_boxes*):
    if *output_dict*['detection_scores'][*j*] ≥ 0.5:
    count = *count* +1
  End
End

Each detection box was stored in the dictionary as a row with four values corresponding to the pixel locations of top-left corner and bottom-right corner. The sequence was y_{min}, x_{min}, y_{max} and x_{max} with respect to the origin of the image at the top-left corner. The default values were normalized between 0 and 1 and hence were difficult to perceive. They had to be converted to the actual scale of the image using the image dimensions. The images used in this study were square images. Hence the image width and height were same. The bounding-box coordinates were converted into the pixel-based values by multiplying them with the image width (number of pixels).

convert_to_image_coord=output_dict['detection_boxes']*im_height

The output_dict['detection_boxes'] was a numpy array which contained vertical stack of individual arrays for all the boxes. To extract the valid detections from this array, it was spilt into arrays of individual detection boxes[24].

split_image_coord=np.vsplit(convert_image_coord_to_int, total_no_of_boxes)

From the spilt array, those having detection_scores>= 0.5 (IoU) were selected and then stacked into a new numpy array and concatenated to get one final array as below[25].

$array_1$ = []
j = 0
for *j* in range(*total_no_of_boxes*):
  if detection_scores_array[j] ≥0.5:
    $array_1$.*append*(*split_image_coord*[*j*])
    j+=1
    output = *np.vstack*($array_1$)
    k = 0
    concat_valid_boxes=
    np.concatenate($array_1$[:*k*+1]+$array_1$[*k*+1:], *axis*=0)
  End
End

3.3 EVALUATION METRICS

Testing of the algorithm's performance was undertaken using the Evaluation Metrics [16], [17] described by the formulae shown in Equations 1-3. The predicted detection boxes were checked for correctness as per the Ground Truth data. Only those detections, which correctly covered the coconut palm crowns were counted manually and its value was fed to find Precision, Recall and F-1 Score.

Precision=(No of boxes containing coconut palms)/(Total number of bounding boxes) (1)

Recall= (No of boxes containing coconut palms)/(Total number of coconut palms in image) (2)

F1-Score= 2*(*Precision***Recall*)/(*Precision*+*Recall*) (3)

3.4 SEGMENTATION BASED ON HISTOGRAM THRESHOLDS

Segmentation is used in CV for delineating the shape (outer form) of objects. This is useful in gaming applications, self-driven vehicles, sorting fruits, etc. Segmentation can be achieved using CNNs like Mask-RCNN, U-Net, Seg-Net, etc. which generate output if form of instance as well as semantic segmentation [26]. Else, segmentation can also be achieved using thresholds from the image histograms. K-means clustering was used to achieve segmentation of ripe strawberries [27], while thresholding was used for identifying ripened strawberries to assist a robotic harvester [28].

In this study, it was aimed to demarcate the coconut palm crown boundary and then compute the area under the crown cover. To enable this, the output image was cropped to the limits of the valid bounding boxes. This was done by masking all else other than the valid bounding boxes. Towards this, a numpy array with value = 0 was created as below.

mask_zero=np.zeros([im_height, im_width], int)

A copy of this array was used in a loop to check each pixel, whether it falls within the co-ordinates of a valid bounding box. If found true, then the pixel value was changed to 1, else, the pixel value was retained as 0. This was looped for all the valid bounding boxes and the output was a binary mask [28], [29].

for *k* in range (count):

y_{min}=*concat_valid_boxes*[*k*,0]

x_{min}=*concat_valid_boxes*[*k*,1]

y_{max}=*concat_valid_boxes*[*k*,2]

x_{max}=*concat_valid_boxes*[*k*,3]

for *i* in range (*im_height*):

for *j* in range (*im_width*):

if $i \geq y_{min}$ and $i \leq y_{max}$ and $j \geq x_{min}$ and $j \leq x_{max}$:

mask[*i*,*j*] = 1

else:

mask[*i*,*j*]=mask[*i*,*j*]

End

End

End

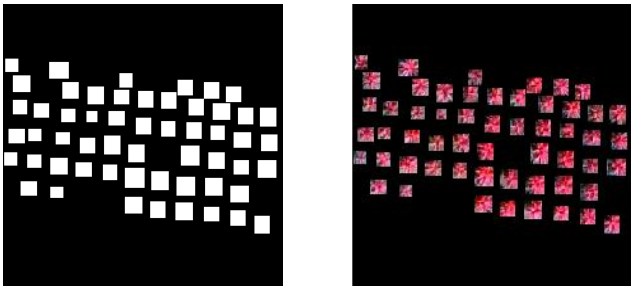
The binary mask was a single layered numpy array. However, for it to be compatible with 3-layered test images, the mask was converted into a three-layered mask. To crop the test image, it was dot-multiplied with the three-layered mask array (Fig.4).

mask_3lyr = np.stack((mask,)*3, axis=-1)

subset_to_boxes=mask_3lyr*image

Coconut crown is a 3-dimensional object made up of multiple leaves radiating out from the center at various angles. The leaf vein and the leaflets have different tones. The leaves facing sunlight appear brighter than others. Though all the leaves together make the crown, the tonal heterogeneity poses a challenge to segmentation. Therefore, it was necessary to subdue the tonal variations and it was achieved by blurring the image using Gaussian Blur in OpenCV.

image_blur = cv2.GaussianBlur(image, (9, 9), 0)

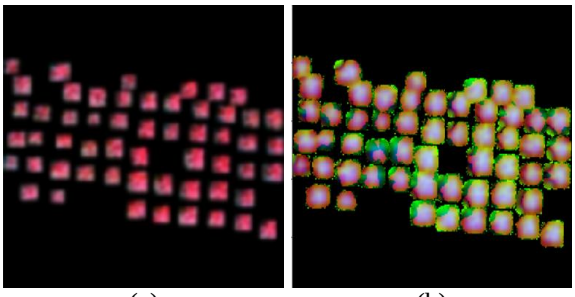


(a) Binary Mask highlighting valid detection boxes (b) Test image cropped to extent of bounding boxes

Fig.4. (a) Binary mask has pixels inside detection boxes with IoU≥0.5 with value of 1. Rest pixels have value of 0. (b) Binary mask image was dot-multiplied with test image to get image cropped to the detections only

In the test image with a False Colour Combination, the healthy coconut crowns appeared red or pink in colour. This colour was to be retained and rest all colours were to be blanked out. The histogram of the test image in Red-Green–Blue (RGB) domain was not good enough to isolate the regions corresponding to coconut crown. So the blurred RGB image was converted into Hue-Saturation-Value (HSV) domain[28] for better separation of the coconut crowns (Fig.5).

image_blur_hsv = cv2.cvtColor(image_blur, cv2.COLOR_RGB2HSV)



(a) (b)

Fig.5. (a) Gaussian Blur applied on cropped image (b) Blurred image converted to HSV domain

Histogram of the HSV image was plotted and displayed in RGB colours (Fig.6).

The histogram values for HSV image could easily separate the regions corresponding to the coconut crowns, red being the dominant colour. Input threshold values from the histogram were then combined to create an output image [28].

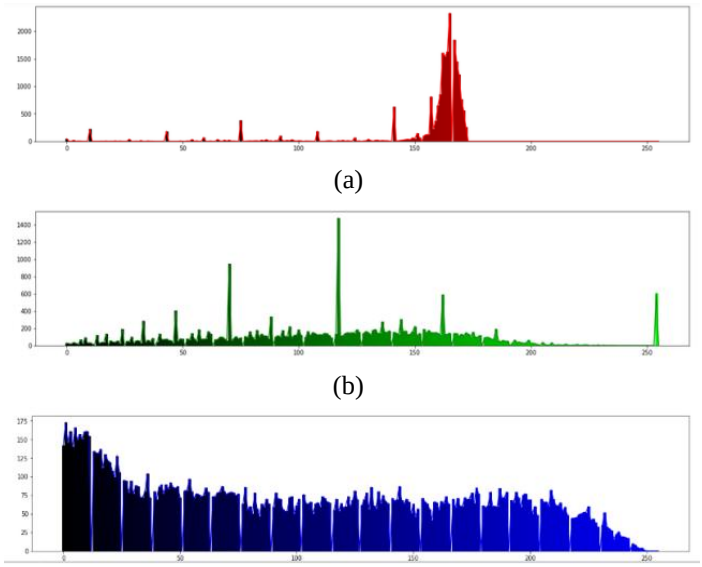
min_red = np.array([50, 80, 50])

max_red = np.array([256, 250, 250])

image_red1 = cv2.inRange(image_blur_hsv, min_red, max_red)

The above output image gave the edges of the coconut crown. It was an 8-bit image with DN values ranging from 0-255. Before it could be used to crop the test image, it had to be converted into

a binary mask. The pixels depicting the coconut crown had DN values between 245 and 255. Therefore, a threshold was applied to convert all the DN values ≥ 245 to 1 and convert all other DN values to 0. This generated a binary mask which was stacked into a 3-layered mask and then was then dot-multiplied with the test image to get the crown cover (Fig.7).

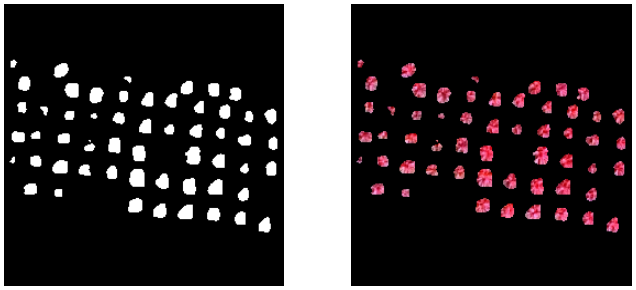


(a)

(b)

(c)

Fig.6. Histogram of (a) Hue (b) Saturation (c) Value components of HSV image



(a) Binary mask of coconut crown boundary extracted using HSV Histogram values

(b) Test image cropped to extent of coconut crown boundary

Fig.7. (a) Binary mask generated from histogram of HSV image includes only the shape of the coconut crowns. (b) Extracted coconut palm crown shapes

3.5 ESTIMATION OF CROWN COVER AREA

From the binary mask of the coconut crowns generated earlier, loops were invoked to count the number of pixels with value 1, which covered all the pixels of the coconut crowns only. This count was then multiplied with the square of the image spatial resolution to get the area under the coconut crowns. Crown cover of the coconut palms for the image depicted in Fig.7 was estimated from GT and it was found to be 1004.91 m^2. The algorithm estimated crown cover to be 879.68 m^2, which was 87.54 % of the GT value. The coconut crown is not a perfect circle, since the radiating leaves in the coconut crown create gaps at the fringes. This gap cannot be measured during GT, but is sensed on images. The algorithm could identify the pixels in such gaps and removed them during segmentation. But, histogram thresholding does have limitations as it tends to exclude certain pixels at the crown fringes due to lesser tones of red. Therefore, at these places, the shape and area of coconut crown demonstrate variation in the GT and the predicted values.

3.6 MAP OF HEALTH VARIATIONS

After counting detecting and the coconut palms, it was essential to segregate the healthy palms from the lesser healthy ones. This would help to identify the palms that are infected or need more nutrition.

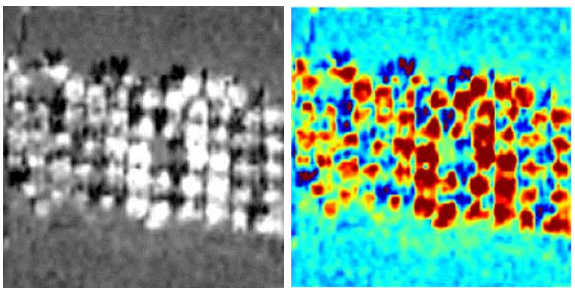
Red-Edge is the region between 700-750 nm wavelengths of the Electro-Magnetic spectrum, which helps to characterize stress in plants related to nitrogen, chlorophyll, senescing and water content. Band-6 of World View-3 satellite corresponds to Red-edge (698 -749 nm) and Band-7 corresponds to NIR (765–899 nm). Using these two bands from the Pan-sharpened image, Normalized Difference Red Edge Index (NDRE) was computed using the formula (Eq.(4)) [30], [31], where DN stands for Digital Number (pixel value).

$$NDRE = \frac{(DN_{Band\,7}) - (DN_{Band\,6})}{(DN_{Band\,7}) + (DN_{Band\,6})}, \quad (4)$$

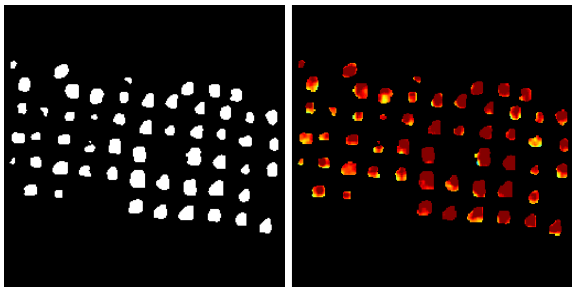
NDRE image is a single-band grayscale image in which the pixel values range from -1 to 1. The darker pixels (lower values) represent less healthy vegetation, while brighter pixels (higher values) represent healthy vegetation. To highlight the tonal variations the gray-scale image was rescaled to 0-255 and then converted to a pseudo-colour image using the 'Jet' colour scheme in OpenCV [32]. In this pseudo-colour image, red colour depicted healthy palm crowns, while orange /yellow corresponded to less healthy crowns. It was further dot-multiplied with the mask array to retain only the pixels pertaining to the coconut crowns. So the background area was removed and thus, the Health Variation Map of the Coconut Crown Cover was obtained (Fig.8).

```
im_255=im*255
im_color_255=cv2.applyColorMap(im_255,cv2.COLORMAP_JET)
crown_health=im_color_255*coconut_mask
```

Since the coconut crowns are made of leaves radiating outwards from the center, the canopy density is maximum at the center and reduces towards the fringes. NDRE is directly proportional to canopy density and is inversely proportional to gap fraction and senescence [31]. Hence, the inner part of the crown is dense and so appears red in NDRE pseudo-colour image.

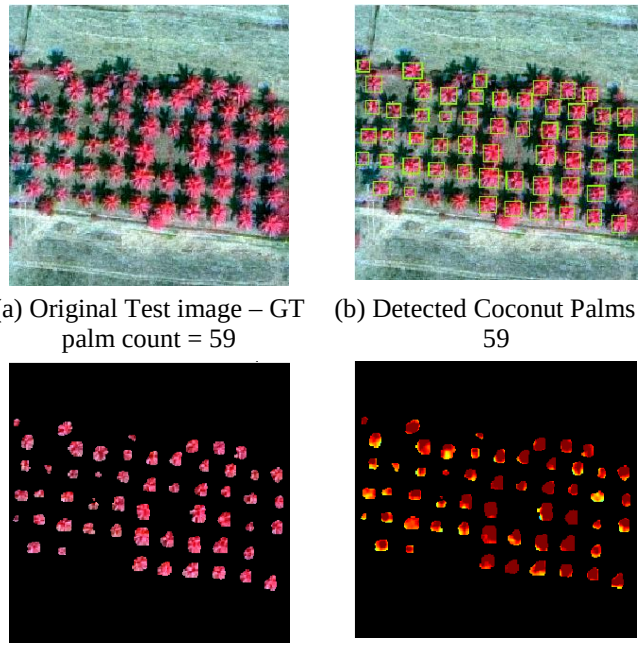


(a) (b)



(c) (d)

Fig.8. (a) Grayscale NDRE image with values ranging from -1 to 1. (b) NDRE image rescaled to 0-255 and converted pseudo-colour image. Red colour depicts healthy crowns and yellow colour depicts less healthy. (c) Binary mask of the coconut crown shape. (d) Map of the Health Variations.



(a) Original Test image – GT palm count = 59 (b) Detected Coconut Palms = 59

(c) Coconut Crown Cover = 879 m^2 (d) Path Health: Red – Healthy; Yellow – Less Healthy

Fig.9. Pictorial Summary of the automated coconut census from satellite image

At the fringes, the leaf density is less, so the NDRE is also less (yellow colour). Therefore NDRE should be compared with the visual image to cross-check if the lower values of NDRE are due to background effect or due to actual poor health. Study of this NDRE image reveals that yellow colour is more prevalent in the western edge of the image. That means the palms on the west side are more stressed and solicit agricultural measures to mitigate the stress.

3.7 AUTOMATED COCONUT SURVEY

Combination of all the above results produced a holistic and automated approach for coconut palm detection (bounding boxes), mapping the crown shapes and health variations from World View-3 satellite images. In addition to it tabular data consisting of count of palms and crown cover area was generated. Summary of the automated census for a small area has been demonstrated in Fig.9. Thus, it was proved on a limited area that automated census of coconut palms was possible using transfer learning of pre-trained CNNs to detect, count, segment the coconut palm crowns and to measure the crown cover area to get the estimate of extent of crop cover and also get the map of relative health status of the coconut palms using multiband satellite image.

4. CONCLUSION

Need of the hour is to expand the CV techniques to assist in solving practical challenges. The technology should be customized to reduce the human effort on ground. Therefore, this research expanded the domain of the CV techniques beyond just detection and counting of the coconut palms. It not only delineated the crown shape, but also found out the relative levels of stress in the coconut palms. This would facilitate identifying the plants needing additional agricultural management practices.

This study used TensorFlow Object Detection API, a FOSS tool, and customized its code to enable it for automatic census of coconut palms using very high resolution images from World View-3 satellite. It used three band combinations of the Pan-sharpened Multispectral images and single-band panchromatic images. The study used SSDLite MobileNet V2 algorithm that was pre-trained on COCO dataset. It was subjected to transfer learning on the custom coconut dataset. This algorithm trained well and was able to successfully detect and count the palms with more than 96 % F-1 score.

The coconut crowns were segmented using histogram thresholding technique, and the crown cover area was computed with 87 % accuracy. Also, mapping of the relative health status (stress variations) was done using the Normalized Difference Red-Edge Index image. The combination of outputs from all these techniques delivered a holistic output in form of automated coconut census using Remote Sensing images. This study was done on a small area, but has demonstrated that the TensorFlow Object Detection API can be successfully used for detection and segmentation of objects from Remote Sensing images. This study would be further extended to include testing of the CNN algorithm on larger sized images and getting the output of detecting boxes in the geospatial domain with the information of latitude and longitude.

REFERENCES

[1] R. Vargas, A. Mosavi and L. Ruiz, "Deep Learning: A Review", *Advances in Intelligent Systems and Computing*, Vol. 5, No. 2, pp. 1-14, 2017.

[2] Y. Li, H. Zhang, X. Xue, Y. Jiang and Q. Shen, "Deep Learning for Remote Sensing Image Classification: A Survey", *Data Mining and Knowledge Discovery*, Vol. 8, No. 6, pp. 1-17, 2018.

[3] I. Goodfellow, Y. Bengio and A. Courville, "*Deep Learning*", MIT Press, 2016.

[4] TensorFlow, "TFOD API-GitHub", Available at https://github.com/tensorflow/models/tree/master/research/object_detection, Accessed at 2020.

[5] K. Patel, "Custom Object Detection using TensorFlow from Scratch", Available at:

https://towardsdatascience.com/custom-object-detection-using-tensorflow-from-scratch-e61da2e10087, Accessed at 2020.

[6] Evan, "TensorFlow-Object-Detection-on-the-Raspberry-Pi", Available at: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi, Accessed at 2020.

[7] R. Balsys, "TensorFlow Object Detection Merged with Grabscreen Tutorial Part 2", Available at: https://pylessons.com/Tensorflow-object-detection-merged-grab-screen-faster/, Accessed at 2020.

[8] S. Paul, "Vehicle Number Plate Detection", Available at: https://github.com/sayakpaul/Vehicle-Number-Plate-Detection, Accessed at 2020.

[9] V. Sodha, "TensorFlow Object Detection API Tutorial - Training and Evaluating Custom Object Detector", Available at: https://becominghuman.ai/tensorflow-object-detection-api-tutorial-training-and-evaluating-custom-object-detector-ed2594afcf73, Accessed at 2019.

[10] S. Obadja, "Math Operators Object Detection", Available at: https://github.com/stevenobadja/math_object_detection, Accessed at 2020.

[11] Evan, "TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10", Available at: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10, Accessed at 2020.

[12] Coconut Development Board, "Statistics - Area, Production, Productivity of Coconut", Available at: https://www.coconutboard.gov.in/Statistics.aspx, Accessed at 2020.

[13] V.G. Chandrasekharan, V.C. Vasanthkumar, P.V. Preethakumari, R.P. Viswam and E.S. Vinod, "Consolidated Report Concurrent Estimation of Coconut Production in Kerala 2012-13", Available at: https://www.coconutboard.in/images/Survey/report-kerala-2012-13.pdf, Accessed at 2013.

[14] Digital Globe, "World View-3", Available at: https://dg-cms-uploads-production.s3.amazonaws.com/uploads/document/file/95/DG2017_WorldView-3_DS.pdf, Accessed at 2019.

[15] Hands-On Tensor Board, "TensorFlow Dev Summit 2017", Available at: https://www.youtube.com/watch?v=eBbEDRsCmv4, Accessed at 2019.

[16] J. Hui, "mAP (Mean Average Precision) for Object Detection", Available at: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173, Accessed at 2019.

[17] R. Padilla, "Metrics for Object Detection", Available at: https://github.com/rafaelpadilla/Object-Detection-Metrics, Accessed at 2019.

[18] N.D. Gholba, "Detection of Coconut Palms and Allied Species from High Resolution Satellite Images using Deep Learning Techniques", Master Thesis, Department of Computer Science, Andhra University, pp. 1-120, 2019.

[19] T.T. Lin, "labelImg", Available at: https://github.com/tzutalin/labelImg, Accessed at 2019.

[20] D. Tran, "Raccoon Detector Dataset", Available: https://github.com/datitran/raccoon_dataset, Accessed at 2019.

[21] L. Vladimirov, "Training Custom Object Detector — TensorFlow Object Detection API tutorial documentation," TensorFlow, 2018. Available at: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html. Accessed: 27-Mar-2019.

[22] Tensorflow, "Tensorflow Detection Model Zoo", Available at: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md, Accessed at 2019.

[23] M. Hugi, "Tensorflow Numpy Image Reshape [Grayscale Images] - Stack Overflow", Available at: https://stackoverflow.com/questions/51872412/tensorflow-numpy-image-reshape-grayscale-images, Accessed at 2019.

[24] SciPy, "NumPy v1.17 Manual: numpy.vsplit", Available at: https://docs.scipy.org/doc/numpy/reference/generated/numpy.vsplit.html, Accessed at 2020.

[25] P. Sharma, "Computer Vision Tutorial: A Step-by-Step Tutorial on Image Segmentation Techniques (Part 1)", Available at: https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/, Accessed at 2019.

[26] X. Liming and Z. Yanchao, "Automated Strawberry Grading System based on Image Processing", *Computers and Electronics in Agriculture*, Vol. 71, No. 1, pp. 32-39, 2010.

[27] A. Danusasmita, "Image Detection Project Finding Strawberries", Available at: https://github.com/andridns/cv-strawberry/blob/master/strawberry.ipynb, Accessed at 2019.

[28] Rochan, "Crop Image in Tensorflow Object Detection API and Display It", Available at: https://stackoverflow.com/questions/51572429/crop-image-in-tensorflow-object-detection-api-and-display-it, Accessed at 2020.

[29] P.S. Thenkabail, P. Teluguntla, M.K. Gumma and V. Dheeravath, "*Hyperspectral Remote Sensing for Terrestrial Applications*", CRC Press, 2016.

[30] Q. Xie, "Vegetation Indices Combining the Red and Red-Edge Spectral Information for Leaf Area Index Retrieval", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, Vol. 11, No. 5, pp. 1482-1492, 2018.

[31] S. Mallick, "Apply ColorMap for pseudocoloring in OpenCV (C++ / Python)", Available at: https://www.learnopencv.com/applycolormap-for-pseudocoloring-in-opencv-c-python/, Accessed at 2020.