T.R.
SAKARYA UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY

PLANT DETECTION IN AERIAL IMAGES USING DEEP NEURAL NETWORKS FOR SMART AGRICULTURAL APPLICATIONS

M.Sc. THESIS

Zeynep BAYRAKTAR

Department : COMPUTER ENGINEERING

Supervisor : Assoc. Prof. Dr. Numan ÇELEBİ

September 2020

DECLARATION

I declare that the thesis entitled “Plant Detection in Aerial Images Using Deep Neural Networks in Smart Agricultural Applications” is carried out by me under the supervisor of Assoc. Prof. Dr. Numan Çelebi. The whole data in this study have obtained by me, and the whole written and visual information and results are presented within the scope of academic rules and regulations. I further declare that the thesis and the data used in the study are not submitted to any candidate examinations for any degree or diploma.

Zeynep BAYRAKTAR

22.11.2019

ACKNOWLEDGEMENTS

I respect and thank my advisor Mr. Assoc. Prof. Dr. Numan ÇELEBİ for everything he did with patience throughout the preparation of this study. I am thankful for all his guidance, help and support.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AI	: Artificial intelligence
CNN	: Convolutional neural network
RCNN	: Region-based convolutional neural network
YOLO	: You only look once

LIST OF FIGURES

LIST OF TABLES

SUMMARY

Keywords: Computer vision, smart agricultural applications, precision agriculture, image processing, deep neural networks.

Plant detection is an active research area in modern robotic applications, which use computer vision systems to contribute the smart agricultural processes. Detecting a plant within the image and counting its number in a specified area are vital functionalities to provide meaningful information about planting such as observing the growth rate or predicting the yield amount of a significant plant with the help of classical object detection algorithms and more efficiently with deep neural networks. Classical models employ image-processing techniques like segmentation and feature extraction whereas deep neural networks need only to fine-tune the parameters by training the exclusive datasets towards particular tasks.

In this study, we aim to compare the conventional computer vision methods with deep neural network outputs and to detect the plants in a plantation area from aerial images. DenseNet model is exploited as the base model for fine-tuning and an appropriate hysteresis color threshold is applied to determine the interested colors within the plantation field. In addition, object localization is performed using the deep neural network model as well. Additionally, YOLOv3 model is trained with our dataset for comparison of the accuracy. Our dataset includes 1800 images for 3 classes of plants and there exists 600 per class.

The main goal of this study is to provide an understanding of how precision agriculture is handled with computer vision technology and to make an improvement about the subject within the scope of our dataset.

AKILLI TARIM UYGULAMALARI İÇİN DERİN SİNİR AĞLARI KULLANILARAK HAVA GÖRÜNTÜLERİNDE BİTKİ TESPİTİ

ÖZET

Anahtar kelimeler: Bilgisayar görmesi, akıllı tarım uygulamaları, sürdürülebilir tarım, görüntü işleme, derin sinir ağları.

Bitki tanıma, akıllı tarım uygulamalarına katkıda bulunmak adına bilgisayar görme sistemlerini kullanan aktif bir araştırma alanıdır. Klasik nesne tanıma yöntemleri ve daha etkili olarak derin sinir ağları ile, bir resimdeki bir bitkiyi tanımak ve o bitkinin belirli bir alandaki adedini saymak, büyüme oranının gözlemlenmesi ya da belirli bir bitkinin verim miktarının tahmin edilmesi gibi bitkilendirme hakkında bilgiler sağlamak için çok önemli fonksiyonlardır. Klasik modeller segmentasyon ve özellik çıkarımı gibi görüntü-işleme teknikleri kullanırken, derin sinir ağları yalnızca parametrelerin, özel veri setlerinin belirli işlere uygun olarak eğitilmesi ile ince ayar yapma gereksinimi duyar.

Bu çalışmada, klasik bilgisayar görmesi yöntemleri ile derin sinir ağları çıktılarını karşılaştırmayı ve bir dikim alanındaki bitkileri havadan görüntülerden tespit etmeyi amaçladık. İnce-ayar için DenseNet modelinden faydalanıldı ve dikim alanındaki ilgili renkleri belirlemek adına uygun bir histerez renk eşik uygulandı. Ek olarak, derin sinir ağları modeli kullanılarak nesne lokalizasyonu da uygulandı. Ayrıca, hassasiyet kıyaslaması için YOLOv3 modeli de veri setimiz ile eğitildi. 3 bitki sınıfından oluşan veri setimiz, her bitki tipi için 600 adet olmak üzere toplam 1800 resimden oluşmaktadır.

Bu çalışmanın temel amacı, sürdürülebilir tarımın bilgisayar görmesi teknolojisi ile nasıl ele alındığının anlaşılmasını sağlamak ve veri setimiz çerçevesinde konu hakkında iyileştirmede bulunmaktır.

CHAPTER 1. INTRODUCTION

As machines have started to get smarter, artificial intelligence has become one of the top subjects that attract researchers' and developers' attention. Many studies are carried out in order to make machines see, act and even perceive the outside world as well as a human being. The subject has separated into branches namely machine learning, computer vision etc. in consequence of these studies. And it seems that the more the studies get deeper, the more their separation will move on.

Computer vision is the science area that aims to provide machines a high-level of understanding of the outside world through digital images or videos. Researchers work on interpreting an image, defining the objects and even counting them in an image, which are quite easy actions for humans whereas they all are considerably hard tasks for machines. Not only images, since videos are composed of images, action determination and emotion recognition through videos are also fields of study in the deep learning community.

In the recent past, there were studies developed with classical methods. In [1], Hung et al. proposed an algorithm that utilizes statistical learning and computer vision techniques in order to identify woody weeds in lands using their shadows. They used segmentation i.e. color and texture, for feature extraction. In [2], Yang et al. studied with two stages: one for training to classify each pixel in aerial images as tree or non-tree, and the other one for correlating a set of tree templates with classification results and locating candidate crowns.

In recent years, with the remarkable progress in machine learning, studies lead researchers to use deep learning models in these areas. There exist many studies in the literature on image processing, object/plant detection in aerial or digital images

etc. accomplished with deep neural networks. Li et al. [3] used a convolutional neural network (CNN) to detect and count oil palm trees in crowded plant areas. Olafenwa et al. [4] developed a computer vision and deep learning python library namely ImageAI, for integrating computer vision technologies easily in new applications. In [5], Dyrmann et al. used a fully convolutional network for detecting mono and dicotyledonous weeds in cereal fields. Tian et al. improved YOLO-v3 model by incorporating the DenseNet method for detecting apples in the main growth stages in orchards [6].

In this study, we studied on ornamental plant detection using convolutional neural networks with the aim of identifying the name of ornamental plants that are placed in almost everywhere in Sakarya for landscaping.

We started our work with classifying the plants. For this process, as it becomes easy to load and build pre-trained models with PyTorch, we used a network based on PyTorch and DenseNet. Dense convolutional network (DenseNet) shown in Figure 1.1. is a pre-trained model that makes forward feeding in connecting each layer and we used it as the base model for fine-tuning in this study.

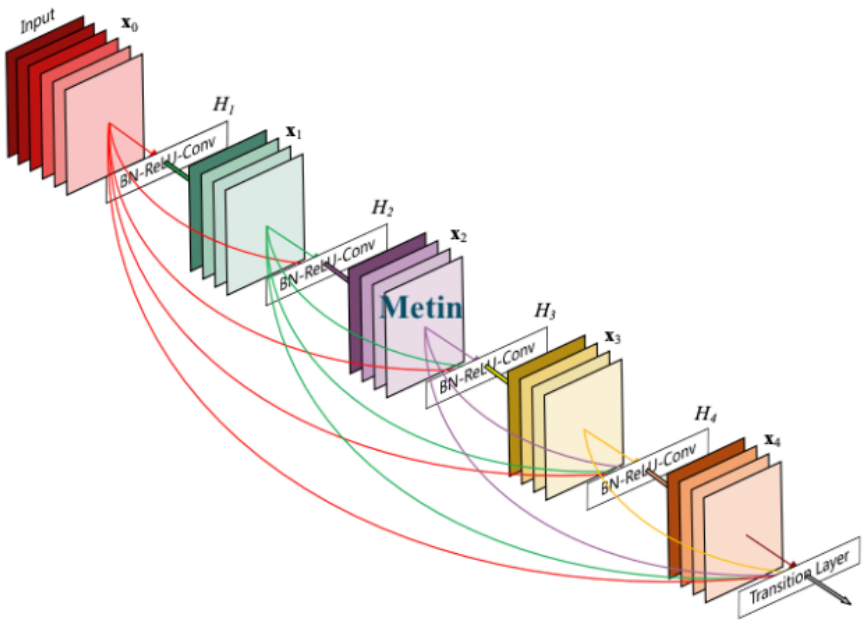


Figure 1.1. DenseNet [7]

In addition to classification, we also trained YOLOv3 model for detecting the ornamental plants in our dataset. For feature extraction in YOLO training, we used Darknet-53 network model detailed in Table 1.1., which uses successive 3x3 and 1x1 convolutional layers and has a total of 53 convolutional layers [8].

Table 1.1. Darknet-53 [8]

	Type	Filter	Size	Output
	Convolutional	32	3 x 3	256 x 256
	Convolutional	64	3 x 3 / 2	128 x 128
	Convolutional	32	1 x 1	
1x	Convolutional	64	3 x 3	
	Residual			128 x 128
	Convolutional	128	3 x 3 / 2	64 x 64
	Convolutional	64	1 x 1	
2x	Convolutional	128	3 x 3	
	Residual			64 x 64
	Convolutional	256	3 x 3 / 2	32 x 32
	Convolutional	128	1 x 1	
8x	Convolutional	256	3 x 3	
	Residual			32 x 32
	Convolutional	512	3 x 3 / 2	16 x 16
	Convolutional	256	1 x 1	
8x	Convolutional	512	3 x 3	
	Residual			16 x 16
	Convolutional	1024	3 x 3 / 2	8 x 8
	Convolutional	512	1 x 1	
4x	Convolutional	1024	3 x 3	
	Residual			8 x 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

The following chapters of this thesis are titled as Literature Review, Methodology, Results and, Discussion and Conclusions respectively.

CHAPTER 2. LITERATURE REVIEW

In this chapter, we made references to the studies about the topic in the literature, starting from the major title Computer Vision.

2.1. Computer Vision

Computer Vision is one of the fields of Artificial Intelligence, which can recognize and understand images through computers and software systems [4]. It covers a variety of subjects such as image recognition and object detection [4]. Computer Vision is the branch of computer science that has enabled computers to see through a web camera and record the scene in its own language.

2.2. Machine Learning

Machine learning is the area that occurred as an answer to the question of whether computers can go beyond "what we know how to order it to perform" [9]. With machine learning, the classical programming method of giving rules and data and getting output as a result has moved to a new paradigm as giving data and output and getting rules as the output [9]. This simple paradigm of machine learning is being used among a wide scale of tasks ranging from image classification to speech recognition [9]. There are researches, which are done by using machine learning-based methods for oil palm tree detection [3], image prediction and object detection [10].

Machine learning can also be described as making the computer understand through a given pattern. The system or namely the model is trained over some number of samples and made capable of learning from the data.

2.3. Deep Learning

Deep learning is one of the subfields of machine learning that provides a mathematical framework for learning representations from data [9]. Despite the traditional way of programming that analyzes the data in a linear way, deep learning method processes the data in a nonlinear way with its hierarchical structure [10]. Deep learning is a fast and effective tool for counting trees from airborne optical imagery [10].

"Deep" in deep learning corresponds to the successive layers of representations [9]. The number of layers that contribute to a model of data is the *depth* of the model [9]. Deep learning maps inputs i.e. images to targets i.e. label "plant" by a deep sequence of data transformations called layers, which are learned through examples [9].

In learning process, the transformation implemented by a layer is parameterized by its weights [9]. Learning means finding a set of values for the weights of all layers in a network so that example inputs are mapped to the related targets correctly by the network [9].

In a deep neural network, there exist a large amount of parameters, which makes it necessary to measure how far the output is from what is expected for finding the correct value for the weights, and this task is handled by the loss function of the network [9]. Loss function of a network does the computation of a distance score using the predictions of the network and the specified output i.e. target [9]. The score is then used as a feedback signal for adjusting the weight values in a direction that will lower the loss score for the current example [9].

2.4. Convolutional Neural Networks

Convolutional neural networks (CNNs) were built in the 1970s and proposed as the early architectures of deep neural networks [11]. The development of deep neural networks has facilitated CNNs to improve rapidly [11]. They are used among a large

scale of computer vision studies such as object detection [4], tree counting [11], oil palm tree detection and counting [3], fig plant segmentation [12] and so on.

CNNs, or more precisely, the gold standard for image classification since 2012, have improved a lot in a way, which enables them to not only classify objects but also detect and locate the objects (with bounding boxes) in images [13].

CNNs have been used in a variety of studies in the literature. In [5], Dyrmann et al. used a fully convolutional network with the aim of detecting the locations of weeds that are distinguished as monocots or dicots in images from cereal fields. They stated that the reason for not choosing RCNN (region-based CNN), fast-RCNN or faster-RCNN was the architectures of them limiting their usage due to long processing times on standard hardware [5]. In [14], Xu et al designed a CNN for cotton bloom detection because of its effectiveness in recognizing flower species and the advantages of it over traditional machine learning methods in feature extraction. Fuentes-Pacheco et al. proposed a CNN with an encoder-decoder architecture for studying the problem of plant segmentation at the granularity of pixel on the grounds of convolutional layers' highly robust approximation functions and their achievements with different image-related tasks [12]. Fan et al. built a convolutional neural network with the aim of classifying candidate regions as tobacco plant regions or nontobacco plant regions [15]. In [3], Li et al. used high-resolution remote sensing images of oil palm trees from Malaysia with the aim of detecting and counting the trees with a convolutional neural network based framework.

In order to train a convolutional neural network, a number of training and test samples are needed. In [3], Li et al. have remarked that they have collected manually interpreted samples for this process. Then, by tuning its main parameters, they have optimized the CNN. In [10], Singh et al. have studied on swimming pool detection using aerial imagery, and they have used a pre-trained neural network that has been trained on over one million images from the ImageNet corpus in order to extract features and do the fine-tuning operation. Fuentes-Pacheco et al. used a total 110 RGB images with high-resolution of 4000 x 3000 pixels [12], whereas Dyrmann et

al. selected 1368 images for training and validation from a set of 118000 images collected by a camera system in various fields and at different times [5]. Xu et al. collected data with one flight of a drone and stated that they counted the blooms in each plot on the same day of drone flight [14].

2.4.1. Detection and classification of plants with CNNs

Object or more specifically plant detection in images has been a significant area in today's world enclosed with AI. Cultivators need to optimize their time and resources in order to keep pace with the business life. Plants should be monitored while they grow up, or a spray system should be accurate in detecting the crop centerline for the sake of productivity maximization.

With deep learning era, studies on image processing like object detection and classification have reached a level of almost 100% success rate with convolutional neural network methods such as RCNN, fast-RCNN and YOLO, which leaded different procedures in studies. The network that Dyrmann et al. studied uses the first five layers of VGG16 as basic feature extraction layers and a set of default bounding boxes at each location in the final feature maps to determine the locations of weeds [5]. Li et al. used LeNet in their CNN model in [3] while Fuentes-Pacheco et al. used a CNN model inspired by SegNet architecture for the process of pixel-wise semantic segmentation in [12].

2.5. YOLO: You-Only-Look-Once

In recent years, alongside the models of CNNs and RCNNs, a state-of-the-art model in object detection has emerged namely You-Only-Look-Once: YOLO.

YOLO is the new object detection approach developed by Redmon et al. [16] that frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities [16]. Unlike sliding window or region proposal-based techniques, YOLO can be trained on full images in one evaluation

[16]. With YOLO, a single convolutional network predicts multiple bounding boxes and class probabilities for those boxes at the same time and it directly optimizes detection performance [16].

Since its establishment, it has been improved to detect a large variety of different objects with more accuracy and speed [17][8]. YOLOv2 can predict detections for more than 9000 categories, and the detection can be held in real-time [17].

In [18], the developers of YOLO stated that, they applied a single neural network to the full image, which divides the image into regions and predicts bounding boxes and probabilities for each region.

YOLO divides the input image into an S x S grid in which each grid cell predicts only one object [19]. Each grid cell also makes a fixed number of bounding box guesses with each box has one box confidence score, but the one-object rule limits the closeness of detected objects [19]. Thus, some objects may be missed by YOLO if they are too close to each other [19]. In Figure 2.1., there are two bounding boxes made by the yellow grid cell in order to locate the person in the image [19].

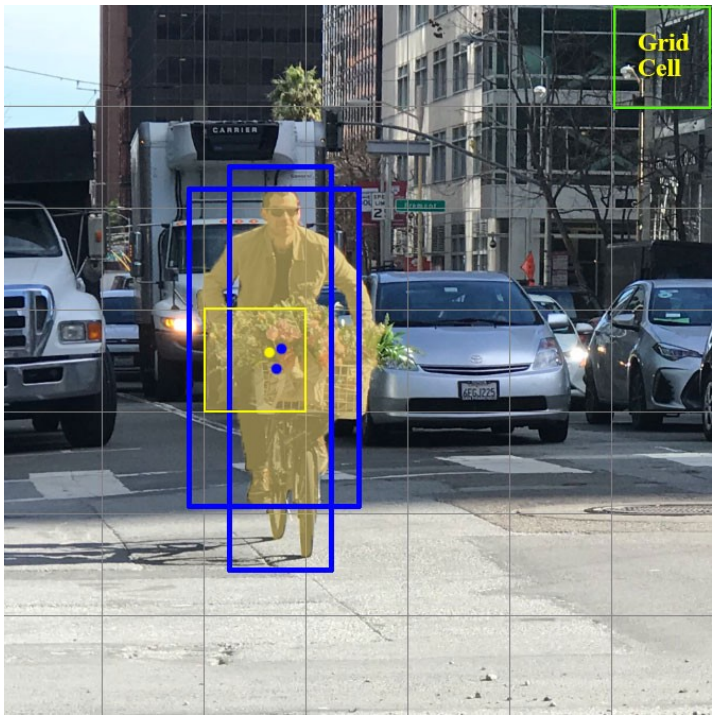


Figure 2.1. Bounding box guesses for an object made by a grid cell

In addition to guessing B bounding boxes each with a box confidence score, and predicting only one object regardless of the number B, each grid cell also predicts C conditional class probabilities –one per class for the likeliness of the object class- [19].

Each boundary box contains five elements namely *x, y, z, w* and box confidence score [19]. The confidence score reflects the likeliness of a box to contain an object (*objectness*) and the accuracy of the bounding box [19]. The bounding box width w and height h is normalized by the image width and height [19]. The x and y values are offsets to the corresponding cell [19]. Thereby x, y, w, h are all ranges between 0 to 1 [19]. The probability of belonging to a particular class is the conditional class probability [19]. As a result, the prediction of YOLO is (S, S, BX5 + C) which is equal to (7, 7, 30) for PASCAL VOC evaluation [19].

YOLO has 24 convolutional layers followed by 2 fully connected layers [19]. The network design is shown in Figure 2.2. YOLO reduces the spatial dimension to 7x7 with 1024 output channels at each location by using a CNN [19]. It performs a linear regression using two fully connected layers for making 7x7x2 bounding box predictions, which is shown in the middle images in Figure 2.3. [19]. The final predictions is made by taking the high box confidence scores that are greater than 0.25 shown in the right image in Figure 2.3. [19].

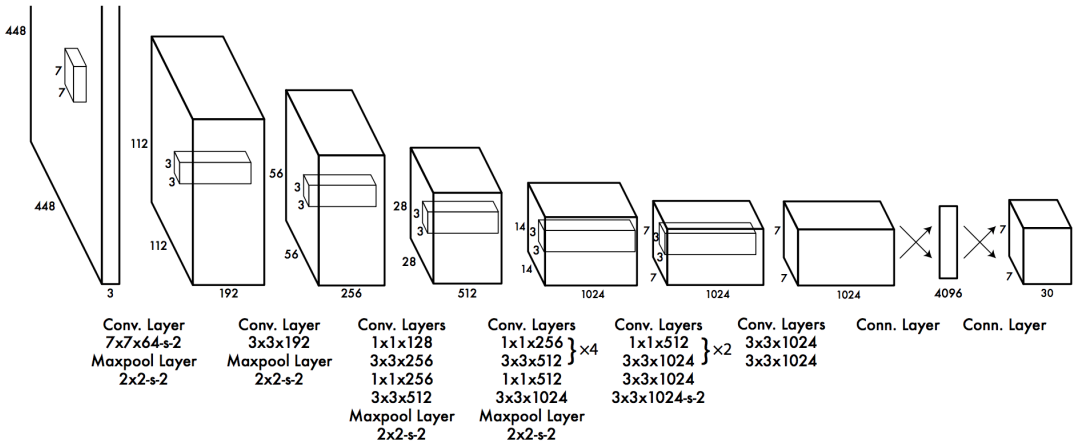


Figure 2.2. Network design of YOLO [19]

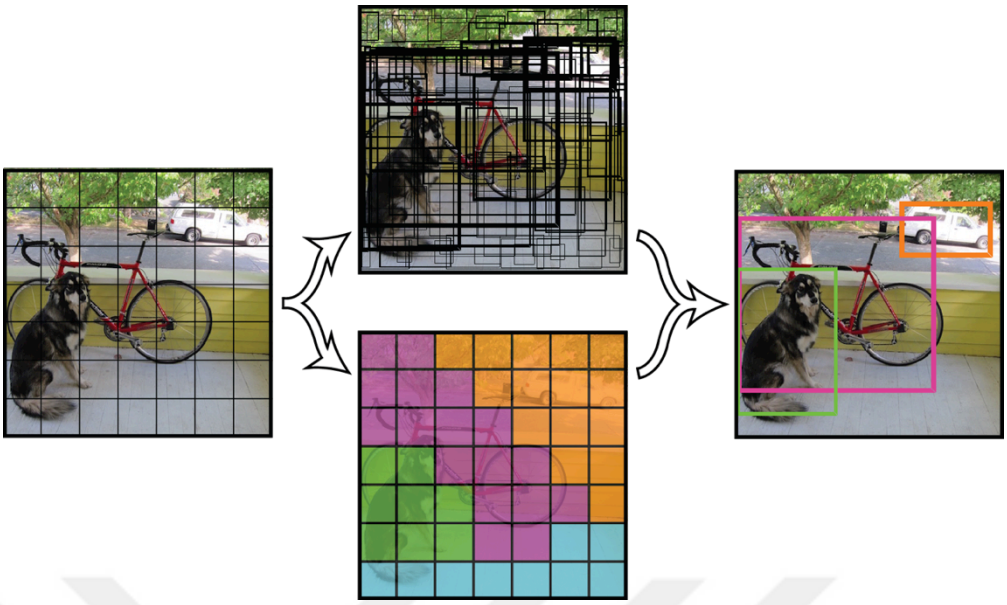


Figure 2.3. How Yolo works [18]

The class confidence score measures the confidence on the classification and the localization and it is calculated for prediction box as in Equation 2.1 [19]. The elements of the equation are given in Equation 2.2 and Equation 2.3 respectively.

class confidence score = box confidence score × conditional class probability (2.1)

Box confidence score ≡ $P_r(object) \cdot IoU$ (2.2)

Conditional class probability ≡ $P_r(class_i|object)$ (2.3)

Class confidence score ≡ $P_r(class_i) \cdot IoU$

= box confidence score × conditional class probability

Here,

$P_r(object)$ is the probability of a box to contain an object

IoU, which corresponds to intersection over union, is the area between the predicted box and the ground truth

$P_r(class_i|object)$ stands for the probability of an object belonging to $class_i$ given the object is presence

$P_r(class_i)$ is the probability the object belongs to $class_i$ [19].

In order to calculate loss, YOLO uses sum-squared error between the predictions and the ground truth [19]. The loss function is composed of the *classification loss*, the *localization loss* and the *confidence loss* [19].

If an object is detected, the squared error of the class conditional probabilities for each class gives the classification loss at each cell [19]. It is calculated by Equation 2.4:

$$\sum_{i=0}^{s^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{2.4}$$

In this equation, $\mathbb{1}_i^{obj} = 1$ if an object appears in cell *i*, 0 otherwise, and $\hat{p}_i(c)$ denotes the conditional class probability for class *c* in cell *i* [19].

The errors in the predicted boundary box locations and sizes give the localization loss [19]. Only the box responsible for detecting the object is counted [19]. Equation 2.5 is used to calculate localization loss:

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$
$$+ \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w}_i - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h}_i - \sqrt{\hat{h}_i}\right)^2 \right] \tag{2.5}$$

Here, $\mathbb{1}_{ij}^{obj} = 1$ if the *j* bounding box in cell *i* is responsible for detecting the object, 0 otherwise. λ_{coord} value increases the weight for the loss in the bounding box coordinates [19]. YOLO puts more emphasis on the bounding box accuracy by multiplying the loss by λ_{coord}, which is taken 5 by default [19].

The confidence loss that corresponds to measuring the objectness of the box is calculated in two ways separated by whether an object is detected or not [19]. If an object is detected in the box, Equation 2.6 is used, and Equation 2.7 otherwise [19].

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \ (C_i - \hat{C}_i)^2 \tag{2.6}$$

where

$\hat{C}_i$ is the box confidence score of the box *j* in cell *i*, and $\mathbb{1}_{ij}^{obj}$ = 1 if the *j* th boundary box in cell *i* is responsible for detecting the object, 0 otherwise [19].

$$\lambda_{noobj} \ \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \ (C_i - \hat{C}_i)^2 \tag{2.7}$$

where

$\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$, $\hat{C}_i$ is the box confidence score of the box *j* in cell *i*, and λ_{noobj} weights down the loss when detecting background [19].

Since most boxes do not contain any objects, in order to remedy the possible class imbalance problem, this loss is weighted down by a factor λ_{noobj}, which is taken 0.5 by default [19].

For calculating the final loss, localization, confidence and classification losses are all added together [19].

In YOLOv3, 3 predictions are made per location, which are each composed by a boundary box, objectness and 80 class scores [19].

Throughout the literature review process about plant detection, we saw a large variety of detection studies including apple detection with YOLOv3 [6], but we came up against a gap about ornamental plant detection.

Sakarya is one of the leading cities in ornamental plant industry in Turkey and there are some ornamental plants that are used for landscape design nearly all over the city without their names written around. Therefore, we wanted to study on detection of these ornamental plants for people, who are interested in flowers and want to learn the name of an ornamental plant that is placed in the park.

CHAPTER 3. METHODOLOGY

This chapter contains detailed information about the whole work we have done throughout the study.

3.1. Model Development

An ornamental plant is a plant that becomes prominent with its flowers, leaves or shape, and is grown as houseplants for decorating houses or gardens.

Center of Sakarya being in the first place, there exist a wide range of production areas in Arifiye, Sapanca and Pamukova. According to Turkish Statistical Institute's data, Sakarya has a significant place in interior and outdoor ornamental plant production in Turkey [20] so that it has its own sectorial festival named Sakarya Peyzaj ve Süs Bitkiciliği Festivali since 2018.

The ornamental plants we used in our study can be encountered almost everywhere in the city; in parks, in gardens, on sidewalks etc. People, who are interested in flowers, may want to buy the same plants from plant cultivators. However, if they haven't taken a photo of them, they would probably need to describe the plants as far as they remember, or maybe need to tour around the cultivators' products to find them.

In our study, we chose to study on ornamental plant detection using deep neural networks with the aim of solving or at least handling these issues. We started with classifying the plants using a network, which is a PyTorch implementation with DenseNet model. We then trained YOLOv3 on ornamental plants for comparing the accuracy of the two models.

To train a convolutional neural network in a reasonable amount of time, strong GPU is a must. Therefore, we trained our model on Google Colaboratory, which provides free GPU for machine learning applications. A preview of Google Colaboratory user interface is in Figure 3.1.

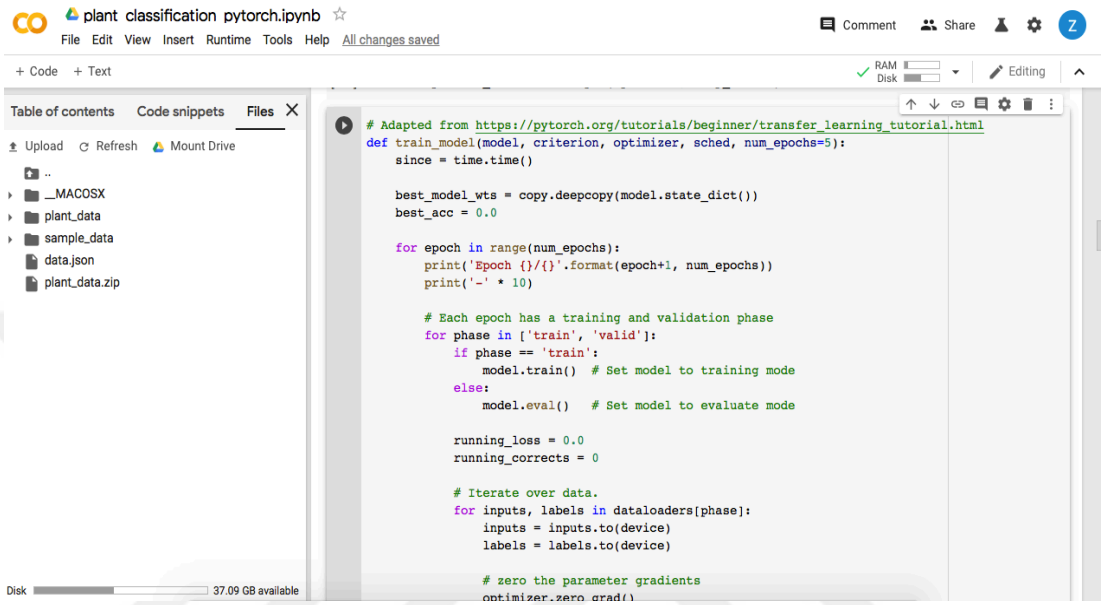


Figure 3.1. Google Colaboratory User Interface

3.2. Collecting Data / Creating the Dataset

Training a convolutional neural network for detecting plants in images, a vast amount of training images are needed. The dataset used in this study is composed of three ornamental plants namely thuja, gold euonymus and sacred bamboo. The data is collected by taking images with a drone and a mobile phone, and also by downloading from the Internet.

After sorting the data, in order to increase the number of instances in the dataset, some additional operations are done such as data augmentation, manual cropping, mirroring and beveling. In the end of the work, a dataset of 1800 images (600 per plant) was obtained.

Samples of each ornamental plant in our dataset can be seen in Figure 3.2.

Figure 3.2. The ornamental plants in our dataset

For the classification process, we separated our dataset as 1440 train and 360 valid images, whereas by creating *.txt* files for each image in the dataset, we obtained a total of 3600 objects in the *plants_data* directory for YOLO training. Details about *.txt* files are explained under *Methodology* caption.

3.3. Methodology

In order to do object detection with a CNN, we first need to collect data and separate it as train and test samples. When creating the model, the training data is given as input to the model, which will take the information of images and do the extraction of visual features. In addition to the feature extraction model, a classification model is needed.

After the classification model is created, testing data is used. All features are extracted from the testing data and then the system is moved on to the model to compare the features with the classes for determining which class the testing data belongs to.

The workflow is visualized in Figure 3.3.

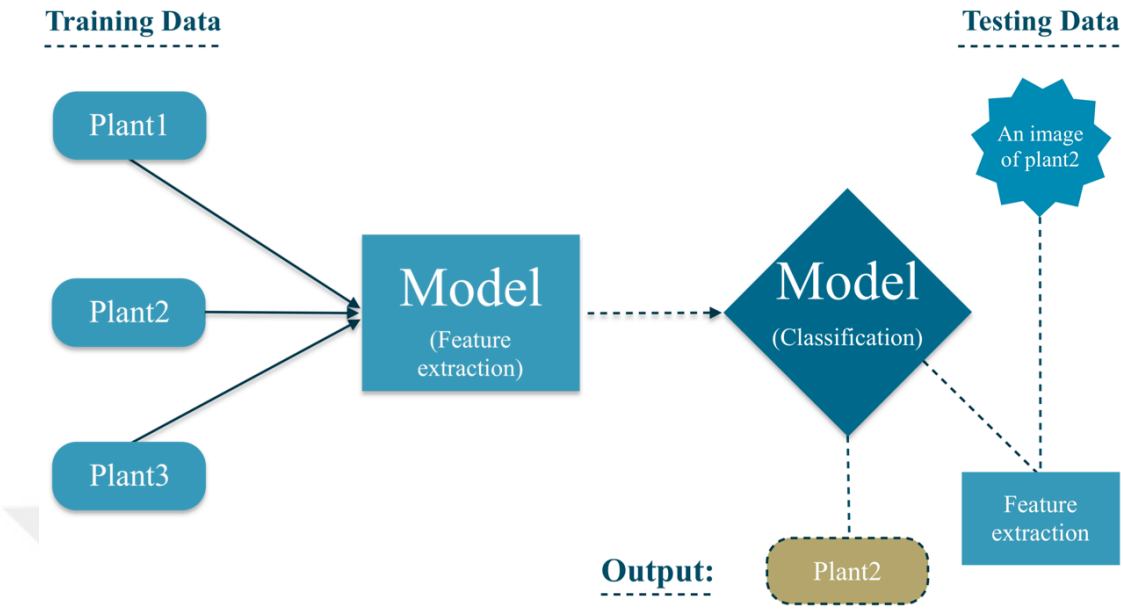


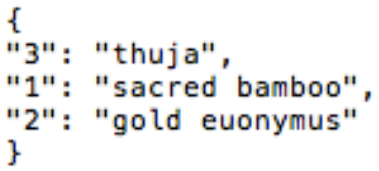
Figure 3.3. Workflow of a CNN

3.3.1. Plant classification with PyTorch

In this study, for the classification process, we used PyTorch as feature extraction model and DenseNet for classification model. In this step, we used a model that was originally developed by Anne Bonner [21]. We modified the code and our dataset according to each other so that our data can be trained with the model. We separated our dataset as train and valid sets as in Figure 3.4. and then, we created the *data.json* file shown in Figure 3.5. that is needed for applying the category names.

```
[5]  data_dir = 'plant_data'
     train_dir = data_dir + '/train'
     valid_dir = data_dir + '/valid'
```

Figure 3.4. Our dataset plant_data



```
{
"3": "thuja",
"1": "sacred bamboo",
"2": "gold euonymus"
}
```

Figure 3.5. data.json file preview

After uploading our dataset and *data.json* file on Google Colaboratory, we imported the needed libraries and downloaded the DenseNet model.

We used the same data transforms and classifier in the model, which are shown in Figure 3.6. and Figure 3.7. respectively.

```
[ ] # transforms for the training and testing sets
    data_transforms = {
        'train': transforms.Compose([
            transforms.RandomRotation(30),
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406],
                                 [0.229, 0.224, 0.225])
        ]),
        'valid': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406],
                                 [0.229, 0.224, 0.225])
        ])
    }

    # Load the datasets with ImageFolder
    image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                              data_transforms[x])
                      for x in ['train', 'valid']}

    # Using the image datasets and the trainforms, define the dataloaders
    batch_size = 64
    dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=batch_size,
                                                  shuffle=True, num_workers=4)
                   for x in ['train', 'valid']}

    class_names = image_datasets['train'].classes
```

Figure 3.6. Data transforms defined in the code [21]

```
[ ] # Create classifier
    for param in model.parameters():
        param.requires_grad = False

    def build_classifier(num_in_features, hidden_layers, num_out_features):

        classifier = nn.Sequential()
        if hidden_layers == None:
            classifier.add_module('fc0', nn.Linear(num_in_features, 102))
        else:
            layer_sizes = zip(hidden_layers[:-1], hidden_layers[1:])
            classifier.add_module('fc0', nn.Linear(num_in_features, hidden_layers[0]))
            classifier.add_module('relu0', nn.ReLU())
            classifier.add_module('drop0', nn.Dropout(.6))
            classifier.add_module('relu1', nn.ReLU())
            classifier.add_module('drop1', nn.Dropout(.5))
            for i, (h1, h2) in enumerate(layer_sizes):
                classifier.add_module('fc'+str(i+1), nn.Linear(h1, h2))
                classifier.add_module('relu'+str(i+1), nn.ReLU())
                classifier.add_module('drop'+str(i+1), nn.Dropout(.5))
            classifier.add_module('output', nn.Linear(hidden_layers[-1], num_out_features))
```

Figure 3.7. Classifier code [21]

For the training, we used the train model adopted from Transfer Learning for Computer Vision Tutorial [22] and we trained the model for 30 epochs. We also tried training for 20 epochs and got a high accuracy but nevertheless, we preferred to use 30 for better results.

The loss and accuracy values of training process are shown on Figure 3.8. Values for the validation process are shown on Figure 3.9. as well.

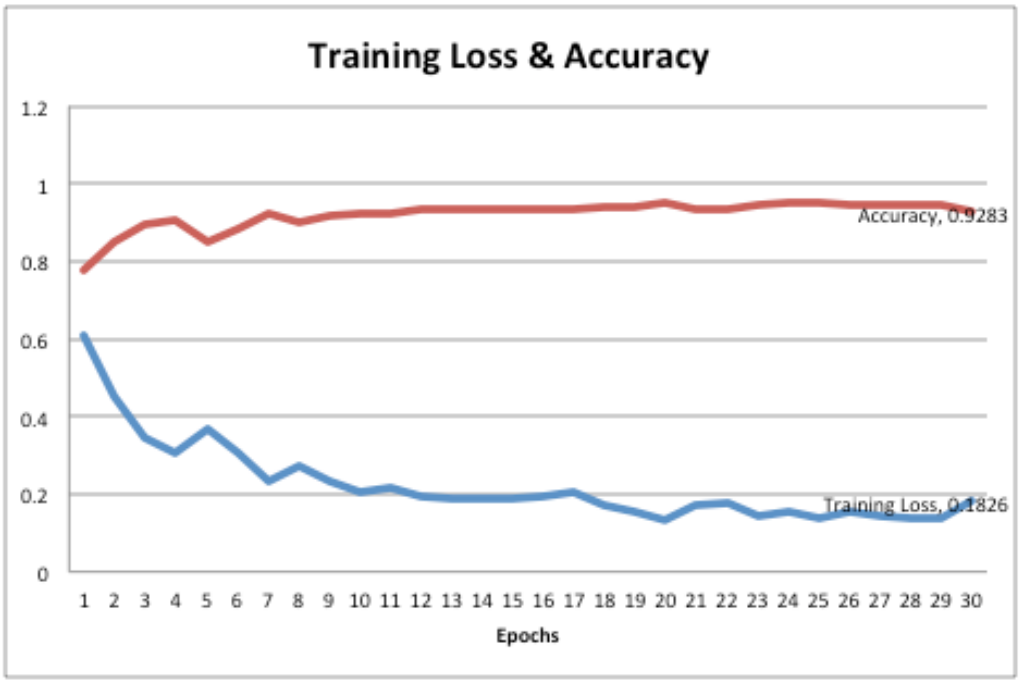


Figure 3.8. Training loss & accuracy chart

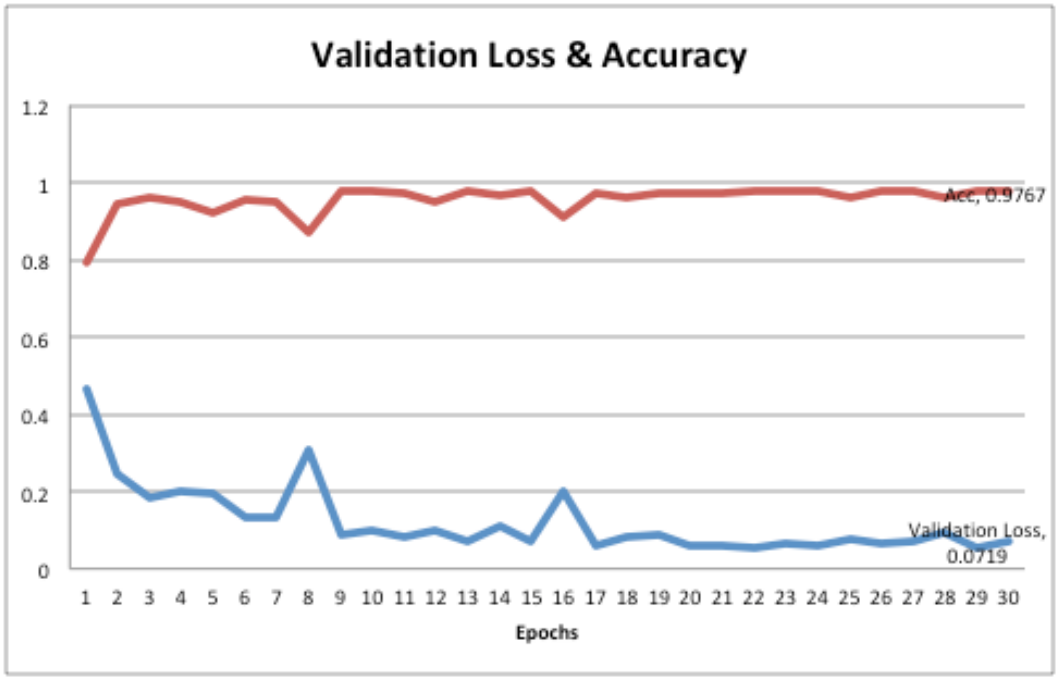


Figure 3.9. Validation loss & accuracy chart

After training process, we evaluated the model for validation on our valid (test) set with the script in Figure 3.10. and had an accuracy rate of 99%.

```
[ ]  # Evaluation

     model.eval()

     accuracy = 0

     for inputs, labels in dataloaders['valid']:
         inputs, labels = inputs.to(device), labels.to(device)
         outputs = model(inputs)

         # Class with the highest probability is our predicted class
         equality = (labels.data == outputs.max(1)[1])

         # Accuracy is number of correct predictions divided by all predictions
         accuracy += equality.type_as(torch.FloatTensor()).mean()

     print("Test accuracy: {:.3f}".format(accuracy/len(dataloaders['valid'])))

     Test accuracy: 0.991
```

Figure 3.10. Evaluation code for the model [21]

With DenseNet model, the training process has reached a high level of accuracy within a couple of epochs. At the end of 30 epochs, we run the script in figure 3.12.

for testing the ornamental plant in Figure 3.11. and it resulted with an accuracy of 99% in predicting the class of the plant. The result is also shown in Figure 3.12.

Figure 3.11. IMG_5751.jpg

```
img_path = '/content/IMG_5751.jpg'
probs, classes = predict2(img_path, model.to(device))
print(probs)
#print(classes)
plant_names = [data[class_names[e]] for e in classes]
print(plant_names)

plt.figure(figsize=(20, 3))

plt.subplot(133)
plt.bar(plant_names, probs)
plt.show()
```

```
[0.9999449253082275, 4.596586222760379e-05, 9.045640581462067e-06]
['sacred bamboo', 'gold euonymus', 'thuja']
```

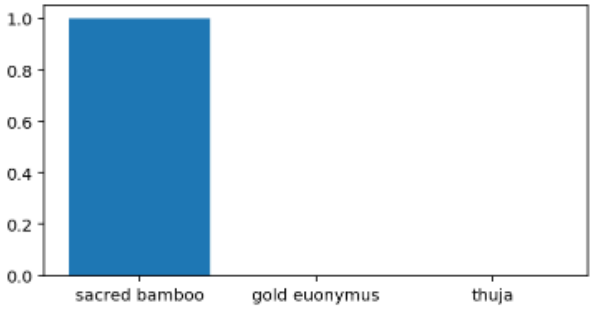


Figure 3.12. Result of classification with PyTorch

3.3.2. Plant detection with YOLOv3

In addition to classification, we also tried to detect plants in images. For this process, we used namely the state-of-the-art Yolov3 (you-only-look-once) model. Since it takes much time to train Yolov3, we also trained the tiny-Yolov3 model simultaneously for being on the safe side.

Before training YOLO, data annotation is needed, which means creating a file for each image in the dataset with the same name but with *.txt* extension containing *the object class number*, *x and y coordinates of the center of object's bounding box*, and *width and height of the bounding box* respectively. A sample of a txt file is shown in Figure 3.13.

There exists a row entry for every bounding box drawn in an image, representing the information about the box [23]. The first field *object class number* is an integer value that stands for the class of the object and ranges from 0 to number of classes-1 [23].

```
0 0.6366666666666667 0.6133333333333334 0.54 0.6666666666666667
0 0.7733333333333334 0.25333333333333335 0.4266666666666667 0.4666666666666667
0 0.2566666666666667 0.2066666666666667 0.43333333333333335 0.37333333333333335
0 0.21333333333333335 0.56 0.36000000000000004 0.6000000000000001
```

Figure 3.13. A sample of .txt file

The second and the third entry are *center-x and center-y* and they correspond to the x and y coordinates of the center of the bounding box [23].

The last two entries, namely width and height, stand for the width and height of the bounding box [23]. All entries other than the first one are divided by the image width and height respectively for normalization [23].

It is quite a tedious task to do the annotation process manually for a big dataset. Therefore we used a data annotation tool developed by Murugavel [24]. The GUI of the annotation tool is shown in Figure 3.14. It is quite simple and is opened as the commend *python main.py* is entered on terminal.

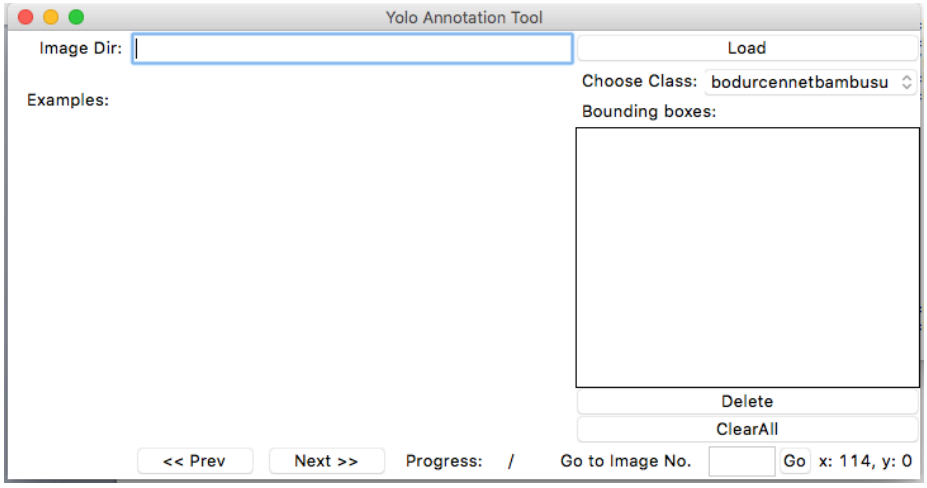


Figure 3.14. GUI of Yolo Annotation Tool [24]

As soon as the directory of the dataset is entered in the search bar, namely *Image Dir*, the images in the dataset are opened randomly one by one for drawing a bounding box around the object with labels listed as a dropdown menu on the right top of the window to specify the object being selected in the current image. Figure 3.15. shows the annotation of the image of a thuja.

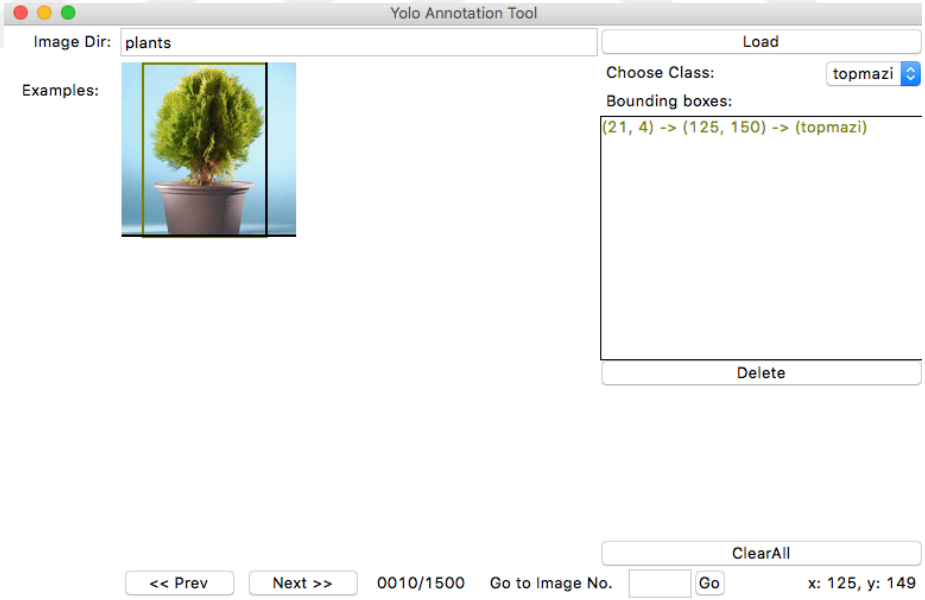


Figure 3.15. Annotation of an image

Each image is saved with the annotations when Next button on the GUI is clicked. Normally, the .txt files created with annotation tool should be converted into YOLO

format by normalizing. Luckily, Murugavel [24] has made this operation inside the main.py, so we did not be obliged to do a converting process.

When annotating process is done all the images in the dataset, in order to create the train and test data, we run the *process.py* script shown in Figure 3.16., which resulted in separating the dataset as *train.txt* and *test.txt* files according to the *percentage_test* value in the script.

```
import glob, os

# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))

print(current_dir)

current_dir = '<Your Dataset Path>'

# Percentage of images to be used for the test set
percentage_test = 10;

# Create and/or truncate train.txt and test.txt
file_train = open('train.txt', 'w')
file_test = open('test.txt', 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir,
"*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))

if counter == index_test:
        counter = 1
        file_test.write(current_dir + "/" + title + '.jpg' + "\n")
    else:
        file_train.write(current_dir + "/" + title + '.jpg' + "\n")
        counter = counter + 1
```

Figure 3.16. process.py script [24]

After data annotation, we continued with creating the files required for YOLO training, which are obj.data, obj.names, yolov3.cfg files. In our study, we named the files as plant-obj.data, plant-obj.names and plants-yolov3.cfg respectively. Besides, we edited the tiny-yolov3.cfg file as plants-tiny-yolov3.cfg for training tiny Yolov3 model.

Figure 3.17. shows the *plant-obj.data* file, which is a plain text file that contains the information about our detector.

```
classes= 3
train = data/train.txt
valid = data/test.txt
names = data/plant-obj.names
backup = backup/
```

Figure 3.17. plant-obj.data file

The class parameter is the number of classes. The train and valid parameters keeps the absolute paths of the train.txt and test.txt files respectively [23]. The names parameter is the path of file that contains the class names. Lastly, the backup parameter is the path to an existing directory where the weights file generated through training will be saved [23].

In Figure 3.18., the *plant-obj.names* file that consists of object names in the dataset each written on a new line is shown.

```
bodur_cennet_bambusu
top_mazi
gold_taflan
```

Figure 3.18. plant-obj.names file

For creating the plants-yolov3.cfg and plants-tiny-yolov3.cfg files, we followed the instructions that Murugavel used in Yolov3 training process [25]. We copied the original .cfg files and edited the batch, subdivisions, classes and filters values according to our dataset. Figure 3.19. shows the upper part of our configuration file.

```
[net]
# Testing
batch=24
subdivisions=8
# Training
# batch=64
# subdivisions=2
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1
```

Figure 3.19. plants-tiny-yolov3.cfg file preview

The *Batch* hyper-parameter in Yolov3 determines the number of images that are going to be used in each training step. Our batch value is 24, which means 24 images are used in one iteration for updating the parameters of the neural network [23].

Subdivisions is a configuration parameter in Yolov3 that stands for decreasing the GPU VRAM requirements by being divided by the batch value.

Since the *classes* value for our dataset is 3, we calculated the *filters* parameter with the formula filters=(classes+5)*3, and updated the values as 24 in our case.

Width, *height* and *channels* configuration parameters determine the input size and the number of channels [23]. The input training images are resized to width x height before training, which is 416 x 416 in our case. *Channels* show that the input images would be 3-channel RGB [23].

Momentum and *decay* parameters control how the weight is updated [23]. Momentum is used to penalize large weight changes between iterations, whereas decay controls overfitting issues [23].

Learning rate is typically a number between 0.01 and 0.0001 that controls how aggressively the learning process will be, based on the current batch of data [23].

Max_batches stands for how many iterations the training process will run for.

Since the training process starts with zero information, the learning rate is needed to be high in the beginning. However, as the neural network encounters a lot of data, it should be decreased over time. *Steps* parameter provides the control of learning rate decrease [23]. In our configuration, the learning rate will start from 0.001 and remain constant for 400000 iterations. Then, it will multiply by *scales* parameter and get the new learning rate [23].

Even though the learning rate should be high in the beginning of the training and low later on, the training speed tends to increase with a lower learning rate for a short period of time at the very beginning [23]. *Burn-in* parameter handles the control of this issue [23].

The *angle*, *saturation*, *exposure* and *hue* parameters can be used for data augmentation. *Angle* makes it possible to randomly rotate the given image by angle [23]. Besides, *saturation*, *exposure* and *hue* can be used to transform the colors of the picture [23]. We used the default values for these parameters.

Before starting the training process, we cloned the *darknet* directory and placed our files into it. The dataset folder named *plants*, *plant-obj.data*, *plant-obj.names*, *train.txt* and *test.txt* files under *data* folder, the configuration files *plants-yolov3.cfg* and plants-tiny-yolov3.cfg under *cfg* folder. In addition, plant-obj.data and plant-obj.names files are needed to be in cfg folder, so we copied them into it.

Then we downloaded the pre-trained Darknet-53 weights, which contains convolutional weights trained on ImageNet and enables the network to learn faster. After that, we started the training by the command below:

```
./darknet detector train data/plant-obj.data cfg/plants-
tiny-yolov3.cfg darknet53.conv.74 -dont_show -map
```

Training tiny-Yolov3 model was much faster than Yolov3 model so that we could be able to train until 67000 in tiny-Yolov3 whereas we reached only to 6000 with Yolov3. Thus we continued our training with tiny-Yolov3 model. Besides, as we trained our dataset on Google Colaboratory, which ends every session after 12 hours, we did the training partially.

We started with pre-trained Darknet-53 weights and reached about 70000 iterations within our session. Through the training process, a new weights file is saved at every 100-iterations. Thus, after each session, we started a new one and continued the training with the last saved weights file.

The training process can be ended when the average loss value is 0.6, and the smaller the value is good for better results [25]. We trained our model till the maximum iteration number specified in the .cfg file is reached at which the average loss value was around 0.4.

In Yolo training, unlike the traditional methods, there is no training and validation accuracy graph. Instead, there exist average loss and mean average precision charts, which are saved during training as a .png file with -map flag.

As we mentioned in detail in Section 2.5, YOLO uses three loss functions separated as classification, objectness (confidence) and box (localization). The scripts for each function, which reside in the *yolo_layer.c* file in *darknet/src* repository, are shown in the Figures below [26]. The loss values are calculated by taking the sum of squares of delta values.

The script for calculating delta for classification loss is shown in Figure 3.20. and Figure 3.21. respectively.

```
void delta_yolo_class(float *output, float *delta, int index, int class_id, int classes, int stride,
    float *avg_cat,                                          int focal_loss, float
    label_smooth_eps, float *classes_multipliers)
{
    int n;
    if (delta[index + stride*class_id]){
        delta[index + stride*class_id] = (1 - label_smooth_eps) - output[index + stride*class_id];
        if (classes_multipliers) delta[index + stride*class_id] *= classes_multipliers[class_id];
        if(avg_cat) *avg_cat += output[index + stride*class_id];
        return;
    }
    // Focal loss
    if (focal_loss) {
        // Focal Loss
        float alpha = 0.5;    // 0.25 or 0.5
        //float gamma = 2;    // hardcoded in many places of the grad-formula

        int ti = index + stride*class_id;
        float pt = output[ti] + 0.000000000000001F;
        // http://fooplot.com/
            #W3sidHlwZSI6MCwiZXEiOiItKDEteCkqKDIqeCpsb2coeCkreC0xKSIsImNvbG9yIjoiIzAwMDAwMCJ9LHsidHlwZSI6
            MTAwMH1d
        float grad = -(1 - pt) * (2 * pt*logf(pt) + pt - 1);    // http://blog.csdn.net/linmingan/
            article/details/77885832
        //float grad = (1 - pt) * (2 * pt*logf(pt) + pt - 1);    // https://github.com/unsky/focal-loss
```

Figure 3.20. Delta for class loss function of Yolov3 part 1

```
        for (n = 0; n < classes; ++n) {
            delta[index + stride*n] = (((n == class_id) ? 1 : 0) - output[index + stride*n]);

            delta[index + stride*n] *= alpha*grad;

            if (n == class_id) *avg_cat += output[index + stride*n];
        }
    }
    else {
        // default
        for (n = 0; n < classes; ++n) {
            delta[index + stride*n] = ((n == class_id) ? (1 - label_smooth_eps) : (0 + label_smooth_eps/
                classes)) - output[index + stride*n];
            if (classes_multipliers && n == class_id) delta[index + stride*class_id] *=
                classes_multipliers[class_id];
            if (n == class_id && avg_cat) *avg_cat += output[index + stride*n];
        }
    }
}
```

Figure 3.21. Delta for class loss function of Yolov3 part 2

Figure 3.22. shows delta for objectness loss calculation.

```
l.delta[obj_index] = l.cls_normalizer * (0 - l.output[obj_index]);
if (best_match_iou > l.ignore_thresh) {
    l.delta[obj_index] = 0;
}
if (best_iou > l.truth_thresh) {
    l.delta[obj_index] = l.cls_normalizer * (1 - l.output[obj_index]);

    int class_id = state.truth[best_t*(4 + 1) + b*l.truths + 4];
    if (l.map) class_id = l.map[class_id];
    int class_index = entry_index(l, b, n*l.w*l.h + j*l.w + i, 4 + 1);
    delta_yolo_class(l.output, l.delta, class_index, class_id, l.classes, l.w*l.h, 0, l.focal_loss,
        l.label_smooth_eps, l.classes_multipliers);
    box truth = float_to_box_stride(state.truth + best_t*(4 + 1) + b*l.truths, 1);
    const float class_multiplier = (l.classes_multipliers) ? l.classes_multipliers[class_id] : 1.0f;
    delta_yolo_box(truth, l.output, l.biases, l.mask[n], box_index, i, j, l.w, l.h, state.net.w,
        state.net.h, l.delta, (2 - truth.w*truth.h), l.w*l.h, l.iou_normalizer * class_multiplier,
        l.iou_loss, 1);
}
```

Figure 3.22. Delta for objectness loss function of Yolov3

Delta for box loss calculation is shown in Figures 3.23., 3.24. and 3.25. respectively.

```
ious delta_yolo_box(box truth, float *x, float *biases, int n, int index, int i, int j, int lw, int lh,
    int w, int h, float *delta, float scale, int stride, float iou_normalizer, IOU_LOSS iou_loss, int
    accumulate)
{
    ious all_ious = { 0 };
    // i - step in layer width
    // j - step in layer height
    //  Returns a box in absolute coordinates
    box pred = get_yolo_box(x, biases, n, index, i, j, lw, lh, w, h, stride);
    all_ious.iou = box_iou(pred, truth);
    all_ious.giou = box_giou(pred, truth);
    all_ious.diou = box_diou(pred, truth);
    all_ious.ciou = box_ciou(pred, truth);
    // avoid nan in dx_box_iou
    if (pred.w == 0) { pred.w = 1.0; }
    if (pred.h == 0) { pred.h = 1.0; }
    if (iou_loss == MSE)    // old loss
    {
        float tx = (truth.x*lw - i);
        float ty = (truth.y*lh - j);
        float tw = log(truth.w*w / biases[2 * n]);
        float th = log(truth.h*h / biases[2 * n + 1]);
```

Figure 3.23. Delta for box loss function of Yolov3 part 1

```
    // accumulate delta
    delta[index + 0 * stride] += scale * (tx - x[index + 0 * stride]) * iou_normalizer;
    delta[index + 1 * stride] += scale * (ty - x[index + 1 * stride]) * iou_normalizer;
    delta[index + 2 * stride] += scale * (tw - x[index + 2 * stride]) * iou_normalizer;
    delta[index + 3 * stride] += scale * (th - x[index + 3 * stride]) * iou_normalizer;
}
else {
    // https://github.com/generalized-iou/g-darknet
    // https://arxiv.org/abs/1902.09630v2
    // https://giou.stanford.edu/
    all_ious.dx_iou = dx_box_iou(pred, truth, iou_loss);

    // jacobian^t (transpose)
    //float dx = (all_ious.dx_iou.dl + all_ious.dx_iou.dr);
    //float dy = (all_ious.dx_iou.dt + all_ious.dx_iou.db);
    //float dw = ((-0.5 * all_ious.dx_iou.dl) + (0.5 * all_ious.dx_iou.dr));
    //float dh = ((-0.5 * all_ious.dx_iou.dt) + (0.5 * all_ious.dx_iou.db));

    // jacobian^t (transpose)
    float dx = all_ious.dx_iou.dt;
    float dy = all_ious.dx_iou.db;
    float dw = all_ious.dx_iou.dl;
    float dh = all_ious.dx_iou.dr;
```

Figure 3.24. Delta for box loss function of Yolov3 part 2

```
            // predict exponential, apply gradient of e^delta_t ONLY for w,h
            dw *= exp(x[index + 2 * stride]);
            dh *= exp(x[index + 3 * stride]);

            // normalize iou weight
            dx *= iou_normalizer;
            dy *= iou_normalizer;
            dw *= iou_normalizer;
            dh *= iou_normalizer;

            if (!accumulate) {
                delta[index + 0 * stride] = 0;
                delta[index + 1 * stride] = 0;
                delta[index + 2 * stride] = 0;
                delta[index + 3 * stride] = 0;
            }

            // accumulate delta
            delta[index + 0 * stride] += dx;
            delta[index + 1 * stride] += dy;
            delta[index + 2 * stride] += dw;
            delta[index + 3 * stride] += dh;
        }

        return all_ious;
    }
```

Figure 3.25. Delta for box loss function of Yolov3 part 3

Mean average precision is the mean value of average precisions of each class [26]. The average precision is average value of 11 points on precision-recall curve for each probability of detection for the same class [26]. Equations for precision and recall are shown in Figure 3.26.

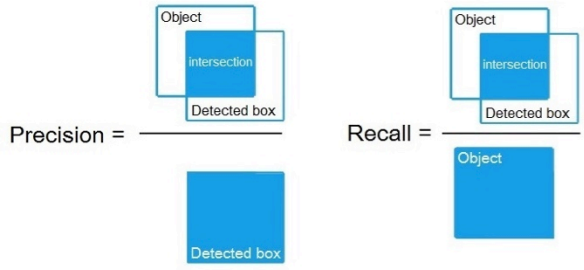


Figure 3.26. Calculation of precision and recall [24]

The loss and mAP-chart of our tiny-Yolov3 model training is in Figure 3.27. in which the average loss value is 0.4361 and the mAP is 46.0% for the last iteration.

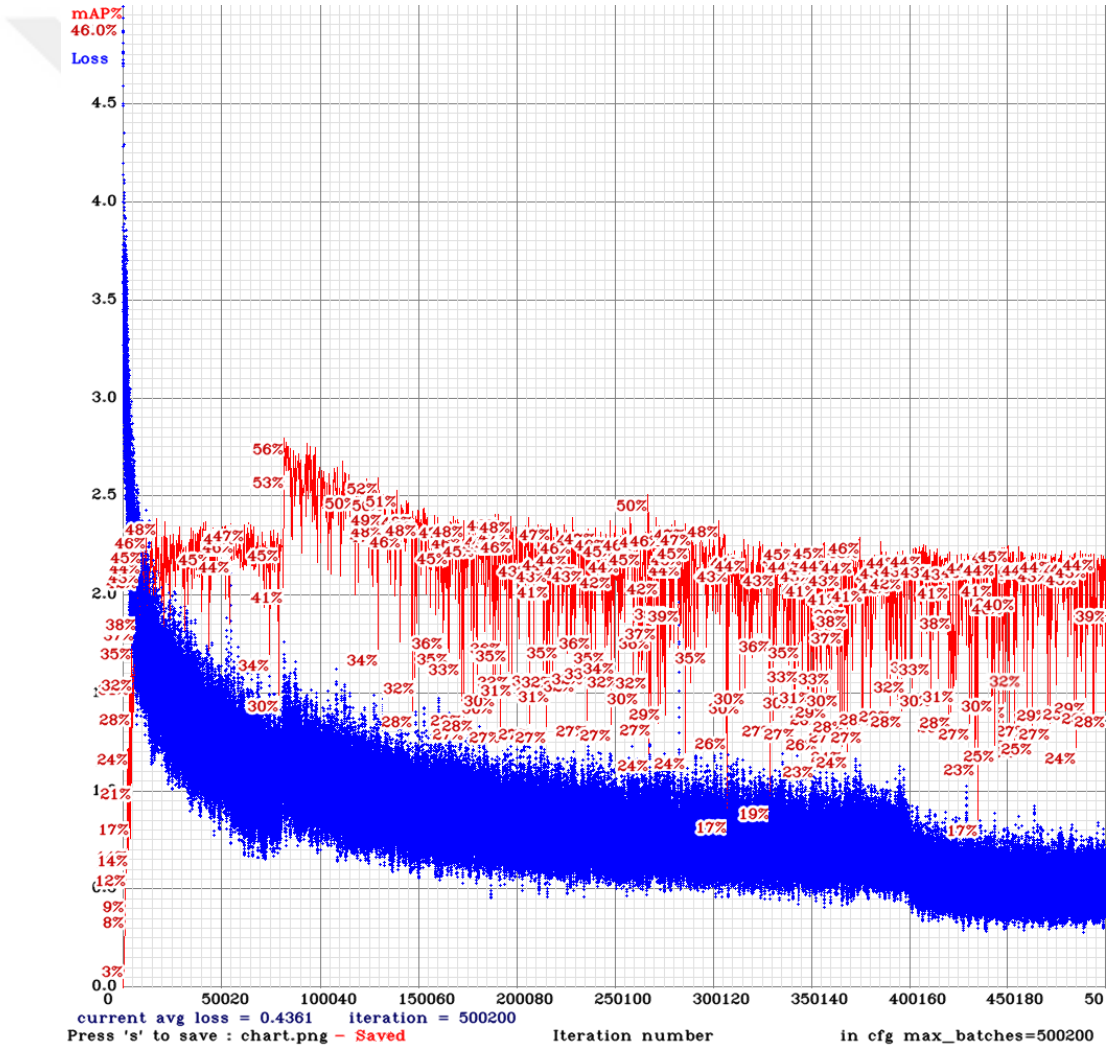


Figure 3.27. Average loss & mean average precision chart

It can be seen from the chart that the average loss value has decreased perceptibly after 400000 iterations, at which the new learning rate is calculated by scales value as we mentioned above.

At the end of our training process, we had a success rate over 80% with tiny-Yolov3 for many of our test images as in Figure 3.28. and Figure 3.29.

```
Total BFLOPS 5.451
Loading weights from plants-yolov3-tiny_67000.weights...
 seen 64
Done! Loaded 24 layers from weights-file
test/IMG_5751.jpg: Predicted in 1268.315000 milli-seconds.
bodur_cennet_bambusu: 82%
bodur_cennet_bambusu: 84%
```

Figure 3.28. Test result of tiny-yolov3 model for IMG_5751.jpg

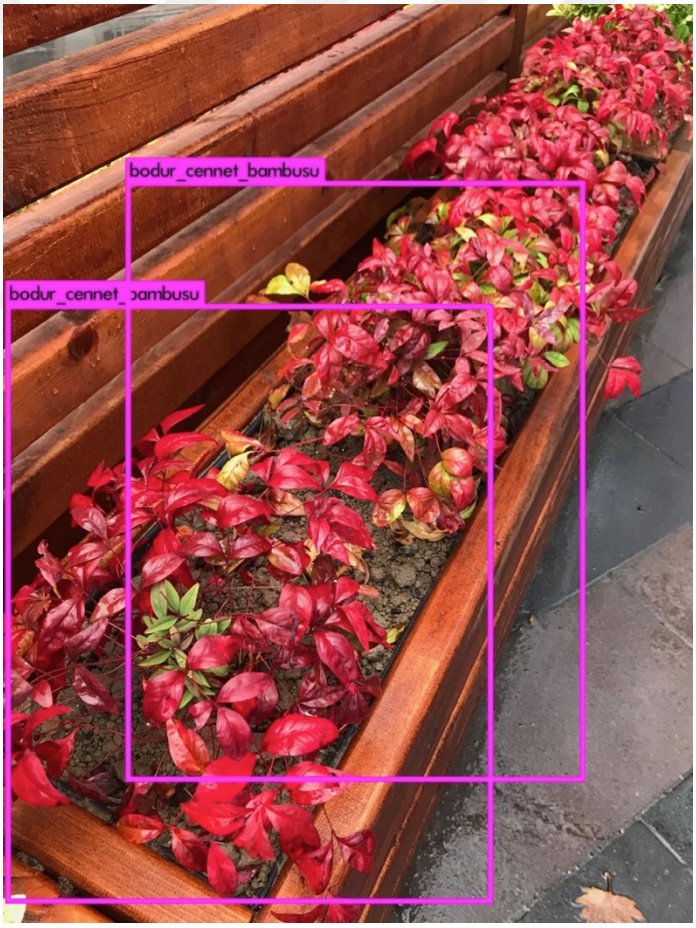


Figure 3.29. Predictions shown on IMG_5751.jpg

Besides, for some test images, although we had correct predictions, the class probabilities were under 70% such as in Figure 3.30. and 3.31.

The reason for this might be a consequence of limited data annotation. Since we had images of flower gardens where determining every single plant is quite hard even by human eyes, we did not draw boxes around all of them. As a result, the model could be able to detect some number of plants in one image.

```
Total BFLOPS 5.451
Loading weights from plants-yolov3-tiny_last.weights...
 seen 64
Done! Loaded 24 layers from weights-file
test/im1.jpg: Predicted in 1690.967000 milli-seconds.
gold_taflan: 54%
gold_taflan: 67%
top_mazi: 33%
gold_taflan: 46%
```

Figure 3.30. Test result of tiny-yolov3 model for im1.jpg

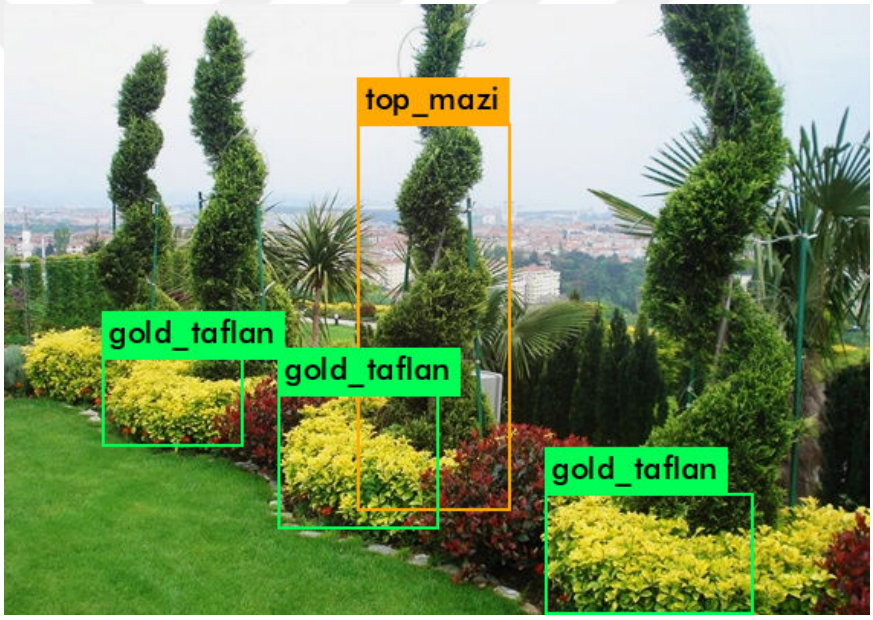


Figure 3.31. Predictions shown on im1.jpg

CHAPTER 4. RESULTS

In this study, we aimed to facilitate specifying ornamental plants in Sakarya parks for everyone by classifying and detecting three ornamental plants with convolutional neural networks. For these processes, we used PyTorch and Yolov3 model respectively.

We trained our 1800-image dataset on a network developed with DenseNet and PyTorch. In consequence of a 30 epochs training, we reached a success rate of 99% in predicting the class of the plant. Figure 4.1. shows a test image of a gold euonymus and its result is indicated in Figure 4.2.

Figure 4.1. Test image img1.jpg of a gold euonymus

```
[0.9998948574066162, 0.00010360587475588545, 1.4973895758885192e-06]
[1, 0, 2]
['gold taflan', 'bodur cennet bambusu', 'top mazi']
```

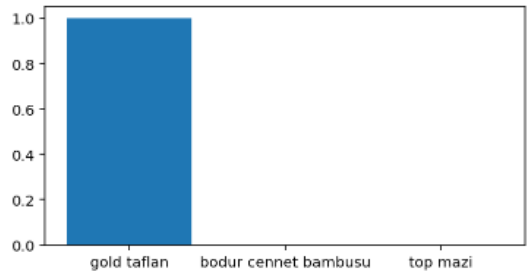


Figure 4.2. PyTorch classification result on img1.jpg

We also tried to detect the three ornamental plants in our dataset with Yolov3 and tiny Yolov3 models.

We succeeded in training the tiny-Yolov3 model and got a success rate over 80% in detecting plants, but we could not be able to get a rate in Yolov3 model due to training on Google Colaboratory. Figure 4.3. shows the test and result of a gold euonymus with tiny-yolov3 model.

```
gold_taflan: 87%
gold_taflan: 97%
top_mazi: 48%
Unable to init server: Could not connect: Connection refused

(predictions:1832): Gtk-WARNING **: 18:09:04.929: cannot open display:
```

Figure 4.3. Tiny-Yolov3 model's detection on a gold euonymus plant image

In Figure 4.4., result of a sacred bamboo test image, which is shown in Figure 4.5., is indicated.

```
Total BFLOPS 5.451
Loading weights from plants-yolov3-tiny_last.weights...
 seen 64
Done! Loaded 24 layers from weights-file
test/iu.jpg: Predicted in 1750.430000 milli-seconds.
bodur_cennet_bambusu: 97%
```

Figure 4.4. Test results of tiny-yolov3 model on iu.jpg

Figure 4.5. Predictions shown on iu.jpg

We believe that the Yolov3 model would also be successful with high accuracy in detecting the plants by a training that is continued until getting average loss value under 0.6, which would have taken weeks on Google Colaboratory.

CHAPTER 5. DISCUSSION AND CONCLUSIONS

Detecting the type of a plant and counting the number of plants in fields are significant topics in artificial intelligence applications that are being studied for years.

In this study, we wanted to draw attention to ornamental plants as we see them everywhere in Sakarya. At the end of our study, we succeeded both in classifying and detecting the plants in our dataset. We got a success rate of 99% in classification with PyTorch and over 80% accuracy in detecting with tiny-yolov3 model.

Creating a large dataset with high resolution images and doing data annotation process more precisely would give better results in detecting the plants, which would also create a way for counting the flowers through images.

A future study about classification and detection of plants can be carried out on counting the flowers in a field through images for the sake of providing automation for plant cultivators' stock tracking. Because we think that counting thousands of plants in a field is an irrational task for humans. As artificial intelligence has reached a level of creating robots that can imitate humans, it should also be creating ways to make people cut free from being responsible for such tasks.

REFERENCES

[1] Hung, C., Bryson, M., Sukkarieh, S., Vision-based Shadow-aided Tree Crown Detection and Classification Algorithm using Imagery from an Unmanned Airborne Vehicle, 34th International Symposium for Remote Sensing of the Environment, 2011.

[2] Yang, L., Wu, X., Praun, E., Ma, X., Tree detection from aerial imagery, 17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, 2009.

[3] Li, W.; Fu, H.; Yu, L.; Cracknell, A. Deep Learning Based Oil Palm Tree Detection and Counting for High-Resolution Remote Sensing Images. *Remote Sens.* **2017**, *9*, 22.

[4] https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606, Access Date: 14.01.2019.

[5] Dyrmann, M., Skovsen, S., Laursen, M.S., Jørgensen, R.N., Using a fully convolutional neural network for detecting locations of weeds in images from cereal fields, 14th International Conference on Precision Agriculture, 2018. Chollet, F. 2018, Deep Learning with Python, Manning Publications Co., Shelter Island, 28-31.

[6] Tian, Y., Yang, G., Wang, Z., Wang, H., Li E., Liang, Z., Apple detection during different growth stages in orchards using the improved YOLO-V3 model, Computers and Electronics in Agriculture, Vol. 157, 2019, 417-426.

[7] https://pytorch.org/hub/pytorch_vision_densenet/, Access Date: 11.11.2019.

[8] Redmon, J., Farhadi, A., YOLOv3: An Incremental Improvement, arXiv: 1804.02767v1, 2018.

[9] Chollet, F. 2018, Deep Learning with Python, Manning Publications Co., Shelter Island, 28-31.

[10] https://medium.com/geoai/swimming-pool-detection-and-classification-using-deep-learning-aaf4a3a5e652, Access Date: 21.09.2019.

[11] Khan, S., Gupta, P. K., 2018, Comparative Study of Tree Counting Algorithms in Dense and Sparse Vegetative Regions, 2018 ISPRS TC V Mid-term Symposium "Geospatial Technology – Pixel to People", 20–23 November 2018, Dehradun, India, Volume XLII-5.

[12] Fuentes-Pacheco, J., Torres-Olivares, J., Roman-Rangel, E., Cervantes, S., Juarez-Lopez, P., Hermosillo-Valadez, J., Rendón-Mancha, J.M., Fig Plant Segmentation from Aerial Images Using a Deep Convolutional Encoder-Decoder Network, Remote Sensing 2019, 11, 1157.

[13] https://hackernoon.com/evolution-of-image-recognition-and-object-detection-from-apes-to-machines-580ed4247f1e, Access Date: 19.11.2019.

[14] Xu, R., Li, C., Paterson, A.H., Jiang, Y., Sun, S., Robertson, J.S., Aerial Images and Convolutional Neural Network for Cotton Bloom Detection, Frontiers in Plant Science, 2018.

[15] Fan, Z., Lu, J., Gong, M., Automatic Tobacco Plant Detection in UAV Images, via Deep Neural Networks, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, Vol. 11, No. 3, 2018.

[16] Redmon, J., Divvala, S., Girshick, R., Farhadi, A., You Only Look Once: Unified, Real-Time Object Detection, arXiv: 1506.02640v5, 2016.

[17] Redmon, J., Farhadi, A., YOLO9000: Better, Faster, Stronger, arXiv: 1612.08242v1, 2016.

[18] https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection, Access Date: 10.10.2019

[19] https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088, Access Date: 15.10.2019

[20] http://www.susbitkileri.org.tr/images/d/library/ea42e662-b5f3-4b88-a02d-219ca8567b80.pdf, Access Date: 20.12.2019.

[21] https://www.freecodecamp.org/news/how-to-build-the-best-image-classifier-3c72010b3d55/, Access Date: 21.09.2019.

[22] https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html, Access Date 25.09.2019.

[23] https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/, Access Date: 01.10.2019.

[24] https://medium.com/@manivannan_data/how-to-train-multiple-objects-in-yolov2-using-your-own-dataset-2b4fee898f17, Access Date: 28.09.2019.

[25] https://medium.com/@manivannan_data/how-to-train-yolov3-to-detect-custom-objects-ccbcafeb13d2, Access Date: 01.10.2019.

[26] https://github.com/AlexeyAB/darknet, Access Date: 03.10.2019

RESUME

Zeynep Bayraktar was born on 11.01.1987 in Sakarya. She completed her primary, secondary and high school education in Sakarya. She graduated from Akyazı Şehit Yüzbaşı Halil İbrahim Sert Lisesi (Y.D.A) in 2005. She started her undergraduate education at Işık University Computer Engineering department in 2005 and graduated in 2011. At the same year, she started working as an IT Specialist at Işık University IT Department. At the end of her one-year contract expiry, she quit her job and returned to Sakarya. She has been working on improving herself in fields she is interested in and computer engineering graduate education is one of them, which she had the opportunity to start at the Institute of Science and Technology in Sakarya University in 2017.