



**REPUBLIC OF TURKEY
AKSARAY UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

**DEPARTMENT OF ELECTRICAL-ELECTRONIC AND COMPUTER
ENGINEERING**

**AN INTELLIGENT SYSTEM FOR DETECTING MEDITERRANEAN
FRUIT FLY [MEDFLY; *CERATITIS CAPITATA* (WIEDEMANN)]**

Ph.D. THESIS

Yusuf UZUN

ADVISOR

Prof. Dr. Mehmet Reşit TOLUN

AKSARAY, 2022

Aksaray University, Graduate School of Natural and Applied Sciences Approval,
Yusuf UZUN, Doctor of Philosophy with thesis, Student No 162353304 successfully
presented the Doctor of Philosophy Thesis titled “**AN INTELLIGENT SYSTEM
FOR DETECTING MEDITERRANEAN FRUIT FLY [MEDFLY; *CERATITIS
CAPITATA* (WIEDEMANN)]**” prepared after satisfying all requirements identified
in the concerned bylaws, before the jury whose signatures are affixed below.

Thesis Advisor: Prof. Dr. Mehmet Reşit TOLUN

Cankaya University

I approve that this thesis is a Doctor of Philosophy in scope and quality

Member: Prof. Dr. Halil Tanyer EYYUBOĞLU

Cankırı Karatekin University

I approve that this thesis is a Doctor of Philosophy in scope and quality

Member: Prof. Dr. Hamit ERDEM

Baskent University

I approve that this thesis is a Doctor of Philosophy in scope and quality

Member: Assist. Prof. Dr. Filiz SARI

Aksaray University

I approve that this thesis is a Doctor of Philosophy in scope and quality

Member: Assist. Prof. Dr. Enes EKEN

Aksaray University

I approve that this thesis is a Doctor of Philosophy in scope and quality

Date of Defence: 22/12/2022

I agree that this thesis which is accepted by the jury, has fulfilled the requirements for
being a Doctor of Philosophy Thesis.

.....

Assoc. Prof. Dr. Mehmet Ali HINIS

Director of Natural and Applied Science

PLAGIARISM PAGE

I hereby declare that all information in this PhD thesis has been obtained and presented in accordance with the academic rules ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

I declare that I will bear all moral and legal consequences that may arise in the event that a situation contrary to this statement I have made about my thesis is detected by the Institute without being bound by a certain time.

Yusuf UZUN

ACKNOWLEDGEMENTS

I would like to extend my thanks to my advisor Prof. Dr. Mehmet Reşit TOLUN for his great leadership and supervision. I would also like to thank committee members Prof. Dr. Halil Tanyer EYYUBOĞLU, Assist. Prof. Dr. Filiz SARI, Prof. Dr. Hamit ERDEM, Assist. Prof. Dr. Enes EKEN, Assist. Prof. Dr. Mehmet KORKMAZ and Assist. Prof. Dr. Ayhan AKBAŞ for their great guidance and contribution.

I extend my thanks to the persons below for their contributions to this study:

Agricultural engineers Ferhat POLAT and Sadık ERTUGRUL who work for the Ministry of Agriculture and Forestry in Turkey. Kubilay DERIN who works for the Research Institute Biological Control in Adana city located south of Turkey on the Mediterranean Sea coast. Halil DANACI who works for the Research Institute Horticulture in Mersin city located south of Turkey on the Mediterranean Sea coast.

Mehmet Fatih TOLGA who works for OYAK Biotechnology Company who provided to this study 107 pictures and 2 videos of his own taking for usage during the training and validation process of the intelligent system. In addition, the left picture in Figure 1.2., the picture in Figure 1.12. and Figure 3.1., the raw pictures of Figure 3.4. and 3.5. (sample case 1 and 2), the raw pictures in Figure 4.1. and 4.3., the raw picture of Figure 4.12. are all supplied by him.

Center for Invasive Species and Ecosystem Health at the University of Georgia provided written authorization to our image usage request number 178344 to use 98 pieces of Medfly pictures during this study (University of Georgia Center for Invasive Species and Ecosystem Health, 2020).

Finally, many thanks to my wife Meltem Sarı UZUN and my children Alya UZUN and Asil UZUN and my brothers Naim UZUN and Ismail UZUN for supporting me at all difficulties.

Yusuf UZUN
AKSARAY, 2022

CONTENTS

ACKNOWLEDGEMENTS.....	i
CONTENTS.....	ii
ÖZET.....	iii
ABSTRACT	iv
LIST OF FIGURES	v
ABBREVIATIONS	vi
1. INTRODUCTION.....	1
1.1 Machine Learning	11
1.2 Deep Learning	13
1.3 Object Detection.....	15
1.3.1 Region based Convolutional Neural Networks (R-CNN).....	17
1.3.2 Fast R-CNN.....	18
1.3.3 Faster R-CNN.....	18
1.3.4 You Only Look Once (YOLO)	20
2. LITERATURE REVIEW.....	22
3. MATERIALS AND METHODS.....	24
3.1 Picture Aquisition.....	24
3.2 Dataset Annotation and Labeling	25
3.3 Deep Learning System Particulars	28
3.4 Image Preprocessing	29
3.5 Training the Intelligent System	29
4. RESEARCH FINDINGS	32
5. DISCUSSION AND CONCLUSIONS.....	42
5.1 Future Work	45
REFERENCES.....	47
APPENDIX LIST.....	52
APPENDIX A. Object-detection-image.py.....	53
APPENDIX B. Visualization-utils.py	58
APPENDIX C.....	99
CIRRICULUM VITAE	100

DOKTORA TEZİ

AKDENİZ MEYVE SİNEĞİNİN (*CERATITIS CAPITATA*) AKILLI SİSTEM İLE TESPİT EDİLMESİ

Yusuf UZUN

Aksaray Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik ve Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Mehmet Reşit TOLUN

ÖZET

Günümüzde tarımla ilgili başlıca sorun, kısaca AMS olarak adlandırılan Akdeniz meyve sineği (*Ceratitis capitata*) başta olmak üzere zararlı haşerelerin meyve, sebze, kabuklu yemiş ve çiçek ürünlerine verdiği zarardır. Bu tehdit ile etkin mücadele edilebilmesi için tarım alanlarındaki AMS varlığının sistematik şekilde izlenmesi gerekir. Bu amaçla sahada özel tuzaklar kullanılarak mevcudiyetleri takip edilerek doğru zamanda pestisit uygulanması yoluyla popülasyonlarının kontrol altında tutulmasına çalışılır. Geleneksel yöntemlerde tuzakların gözlenmesi için sık sık yapılması gereken saha ziyaretleri külfetli ve zor olduğundan günümüzde teknolojik olarak desteklenen otomatik uzaktan izleme sistemlerine ihtiyaç vardır. Bu çalışmada bir resimdeki AMS görüntülerini algılayabilen ve sayısını verebilen bir derin öğrenme sistemi geliştirilmiştir. Günlük olarak tuzağın içindeki yapışkan bandın fotoğrafını çekebilen entegre edilmiş kamera ile donatılmış özel bir tuzak kullanılmaktadır. Çekilen resimler SIM kart içeren bir elektronik devre aracılığıyla nesne algılama algoritmasının çalıştığı merkezi sunucuya iletilir. Bu algoritma fotoğrafta AMS bulunup bulunmadığını, var ise kaç adet olduğu otomatik olarak tespit edilir.

Bu çalışmada tuzağa yakalanmış AMS'leri belirlemede daha hızlı bir bölge tabanlı evrişimsel sinir ağı (faster R-CNN) modeli kullanılmıştır. AMS'ler ve diğer haşereler tuzağın yapışkan bandına yapıştıklarında panik içinde kendilerini kurtarmak amacıyla ölene kadar olağanüstü çaba harcarlar. Bu nedenle, vücutları, kanatları ve bacakları evrilip bükülür, şekilleri kötü bir şekilde bozulur. Buradaki zorluk, derin öğrenme sisteminin bu bozuk şekilli AMS'leri yüksek doğrulukla tanıyabilmesi gerektiğidir. Bu nedenle, akıllı sistem eğitimi ve doğrulama aşamalarında tuzağa yakalanmış bozuk şekillere sahip AMS görüntüleri içeren resimlerin kullanılması çok önemlidir.

Bu çalışmada oluşturulan derin öğrenme sisteminin eğitilmesi sürecinde çok sayıda amaca yönelik özel fotoğrafın veri olarak kullanılması sayesinde bir fotoğraftaki AMS'lerini tanımlamada %94.05 gibi bir başarı oranı yakalanabilmiştir. Bu oran literatürde bulunan güncel çalışmalardaki başarı oranları ile karşılaştırıldığında oldukça başarılı olduğu söylenebilir.

Anahtar Kelimeler: Akdeniz meyve sineği ile mücadele, otomatik haşere izleme, tarımda pestisit optimizasyonu, derin öğrenme, faster R-CNN, haşere e-tuzak

Aralık, 2022; 101 sayfa

Ph.D. THESIS

**AN INTELLIGENT SYSTEM FOR DETECTING MEDITERRANEAN
FRUIT FLY [MEDFLY; *CERATITIS CAPITATA* (WIEDEMANN)]**

Yusuf UZUN

**Aksaray University
Graduate School of Natural and Applied Sciences
Department of Electrical-Electronic and Computer Engineering**

Advisor: Prof. Dr. Mehmet Reşit TOLUN

ABSTRACT

Nowadays, the most critical agriculture-related problem is the harm caused to fruit, vegetable, nut, and flower crops by harmful pests, particularly the Mediterranean fruit fly, *Ceratitis capitata*, named Medfly. Medfly's existence in agricultural fields must be monitored systematically for effective combat against it. Special traps are utilised in the field to catch Medflies which will reveal their presence and applying pesticides at the right time will help reduce their population. A technologically supported automated remote monitoring system should eliminate frequent site visits as a more economical solution. This paper develops a deep learning system that can detect Medfly images on a picture and count their numbers. A particular trap equipped with an integrated camera that can take photos of the sticky band where Medflies are caught daily is utilised. Obtained pictures are then transmitted by an electronic circuit containing a SIM card to the central server where the object detection algorithm runs.

This study employs a faster region-based convolutional neural network (Faster R-CNN) model in identifying trapped Medflies. When Medflies or other insects stick on the trap's sticky band, they spend extraordinary effort trying to release themselves in a panic until they die. Therefore, their shape is badly distorted as their bodies, wings, and legs are buckled. The challenge is that the deep learning system should detect Medflies of distorted shape with high accuracy. Therefore, it is crucial to utilise pictures that contain trapped Medfly images with distorted shapes for training and validation.

In this academical study, the success rate in identifying Medflies when other insects are also present is 94.05%, achieved by the deep learning system training process, owing to the considerable amount of purpose-specific photographic data. This rate may be seen as quite favourable when compared to the success rates provided in the literature.

Keywords: Fight against Mediterranean Fruit Fly, Automatic pest monitoring, pest e-trap, pesticide optimization in agriculture, faster R-CNN, deep learning.

December, 2022; 101 pages

LIST OF FIGURES

Figure 1.1. Appearance of adult male and female Medfly	1
Figure 1.2. Typical views of adult Medfly	2
Figure 1.3. A representative drawing of female and male Medfly	3
Figure 1.4. A fig and a mardarin orange damaged by Medfly larvas	3
Figure 1.5. Medfly larvae inside a persimmon fruit.....	4
Figure 1.6. Medfly pupa on soil	5
Figure 1.7. Medfly pupae	5
Figure 1.8. Emergence of a baby Medfly from puparium.....	6
Figure 1.9. Life cycle of Medfly <i>Ceratitis capitata</i>	7
Figure 1.10. A meeting regarding collective fight against Medfly coordinated by the Ministry of Agriculture and Forestry with participation of Adana Governor	8
Figure 1.11. A Delta trap placed in the field to charm and catch Medfly.....	9
Figure 1.12. Caught Medflies together with some other insects inside the trap	10
Figure 1.13. A Jackson trap.....	10
Figure 1.14. Machine learning vs deep learning	13
Figure 1.15. Face recognition as a deep learning example	14
Figure 1.16. Vehicle detection	15
Figure 1.17. A typical object detection application	16
Figure 1.18. A typical object detection application	17
Figure 1.19. Region Proposal Network of faster R-CNN.....	20
Figure 1.20. Real time object detection with YOLO.....	21
Figure 3.1. Trapped Medflies in a picture from our dataset.....	24
Figure 3.2. Labeling and annotating objects.....	25
Figure 3.3. Labeling and annotating objects.....	26
Figure 3.4. Medfly annotation and labeling for training and validation.....	27
Figure 3.5. Medfly annotation and labeling for training and validation.....	27
Figure 3.6. The architecture of the proposed system.....	28
Figure 3.7. Training the system 100K steps to determine the minimum.....	31
Figure 3.8. Intelligent system optimization and determination of mAP.....	31
Figure 4.1. Sample output of standard Tensorflow.....	33
Figure 4.2. A blue star is placed on every identified object and count is printed....	34
Figure 4.3. A blue star is placed on every identified object and count is printed....	34
Figure 4.4. A blue star is placed on every identified object and count is printed....	35
Figure 4.5. A blue star is placed on every identified object and count is printed....	36
Figure 4.6. A blue star is placed on every identified object and count is printed....	36
Figure 4.7. A blue star is placed on every identified object and count is printed....	37
Figure 4.8. An amazing case: Our intelligent system is precise enough to detect the shadow of a Medfly.....	38
Figure 4.9. The object detector of our intelligent system can be set to print the calculated probability of each detected Medfly.....	39
Figure 4.10. The object detector of our intelligent system can be set to print the calculated probability of each detected Medfly.....	39
Figure 4.11. The object detector of our intelligent system can be set to print the calculated probability of each detected Medfly.....	40
Figure 4.12. The object detector of our intelligent system can be set to print the calculated probability of each detected Medfly.....	41
Figure 5.1. A label map in Phyton for identifying multiple pests simultaneously...46	46

ABBREVIATIONS

AMS	Akdeniz meyve sineği
DL	deep learning
IAEA	International Atomic Energy Agency
ML	machine learning
mAP	mean average precision
Medfly	Mediterranean fruit fly
NN	neural network
R-CNN	region based convolutional neural network
ROI	regions of interest
RPN	region proposal network
SIFT	scale invariant feature transforms
SSD	Single Shot Multibox Detector
YOLO	You Only Look Once



1. INTRODUCTION

The most important agriculture related matter in the world nowadays is by no doubt the damage caused by harmful pests on the crops, the Mediterranean fruit fly, *Ceratitis capitata*, named in short as Medfly being placed on the top. It can damage more than 260 sorts of fruits, no matter they are cultivated or wild. The host list includes apple, apricot, avocado, bell pepper, carambola, coffee, dates, fig, grape, grapefruit, guava, lemon, lime, loquat, lychee, mango, nectarine, orange, papaya, peach, pear, persimmon, plum, pomegranate, pummelo, quince, sapote, tangerine, tomato, and walnut. In California, the combined 2011 gross value of the above hosts was over \$16.5 billion (USDA NASS, 2012). Unfortunately, Medfly is widespread to most tropical and subtropical areas of the sphere. It is not difficult to conclude from here that almost all crops of the world are at risk. Having the advantage to live in cooler climates, Medfly has become more spread than any other fruit fly species globally. So many factors make Medfly among the most influential fly species worldwide in economical terms (Cohen et al., 2007; The University of Arizona, 2021).

Typical views of male and female adult Medfly are presented in Figures 1.1. and 1.2.



Figure 1.1. Appearance of adult male and female Medfly. Left image male Medfly with its club-like mustache, right image female Medfly with its ovipositor at the back (Serantes, 2015).

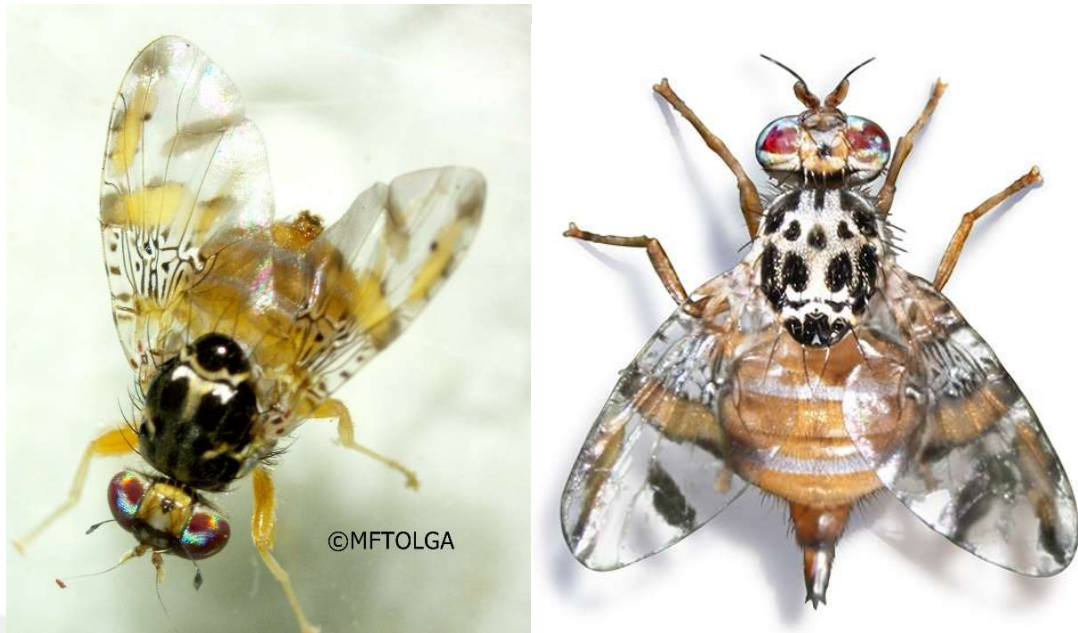


Figure 1.2. Appearance of adult male and female Medfly. Left image male Medfly with its club-like mustache, right image female Medfly with its ovipositor at the back (Serantes, 2015).

Obviously, the population of Medfly must be controlled in order to minimize its harm to the economy. But this is not easy because Medfly could emerge any time depending on instantaneous local weather conditions, that is, the emergence of Medfly is unpredictable. Besides, Medfly grows quite rapidly, females become capable of laying eggs within 5 days after they emerge from pupae (Fruitfly Africa, 2021).

Female Medfly has an ovipositor extended at its back from which it is capable to sink it to inject eggs under the skin of ripening fruits while male Medfly can be differentiated from its clubs as can be seen in the representative drawing provided in Figure 1.3.

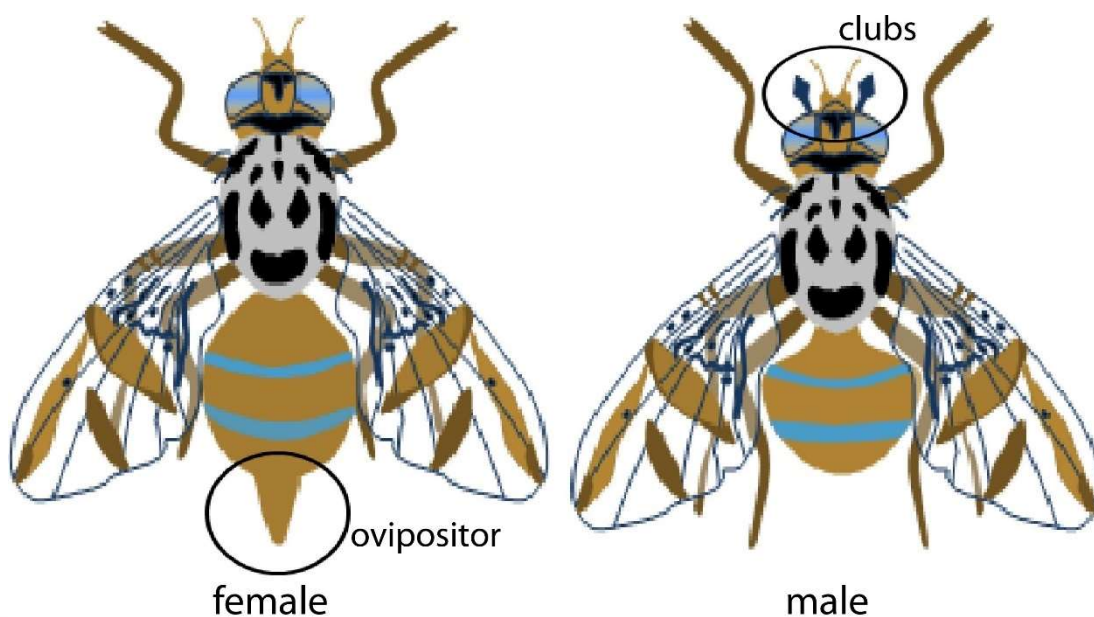


Figure 1.3. A representative drawing of female and male Medfly (State of Western Australia, 2018). With permission of the © owner which is the State of Western Australia, Department of Primary Industries and Regional Development, WA.

Females sink their ovipositor which is like a needle into the shell of ripening fruits to inject up to 10 eggs inside the fruit each time (URL-1). A few days later, Medfly larvae are born there. Because the larvae feed upon the pulp of the fruit and defecate inside it, the whole is reduced at the end to a juicy, inedible mass in the way shown in Figures 1.4. and 1.5. (URL-2).



Figure 1.4. A fig and a mandarin orange damaged by Medfly larvae (Fig: ©Mourad Louadfel, Retired, Bugwood.org; Mandarin: With permission from Florida Division of Plant Industry, Florida Department of Agriculture and Consumer Services, Bugwood.org).



Figure 1.5. Medfly larvae inside a persimmon fruit (Estres vegetal, 2020)

The larvae feed themselves inside the fruit until it's time to pupate, which means to become surrounded by a protective shell to survive under hard conditions. Pupae protect the species during hard conditions, like lack of food, water, and temperature extremes. The pupa is of cylindrical shape, around 4 mm long, of dark reddish Brown color, and looks like a swollen grain of wheat (URL-2). Typical pupa views are given in Figures 1.6. and 1.7.



Figure 1.6. Medfly pupa on soil (with permission of the the State of Western Australia, Department of Primary Industries and Regional Development, WA).



Figure 1.7. Medfly pupae (Florida Department of Agriculture and Consumer Services, Florida Division of Plant Industry, Bugwood.org).

Larvae leave the fruit in large numbers at daybreak or just after it and pupate in the soil or into any other suitable ambient. Then, they keep waiting there until suitable environmental conditions are met, to enable baby Medflies to be born (to emerge from pupae). Emergence is directly dependent on outside weather conditions, on which obviously human being is not able to make exact early estimates. A baby Medfly is photographed while emerging from pupa in Figure 1.8.



Figure 1.8. Emergence of a baby Medfly from puparium the head coming out on the left (with permission, Piero Cravedi, Università Cattolica del Sacro Cuore, Bugwood.org)

Unfortunately, Medfly can breed 7 to 8 puppies per year in the Mediterranean areas (Sorhocam, 2021a). All these facts prove that controlling Medfly population is difficult and complicated so it requires continuous and systematic monitoring of agricultural fields to detect Medfly existence and apply pestciding at the right time. This is vital not only for population control but also for pesticide optimization. Medfly lifecycle is directly dependent on temperature. The lifecycle of Medfly (Fruitfly Africa, 2021) is provided with explanations in Figure 1.9.

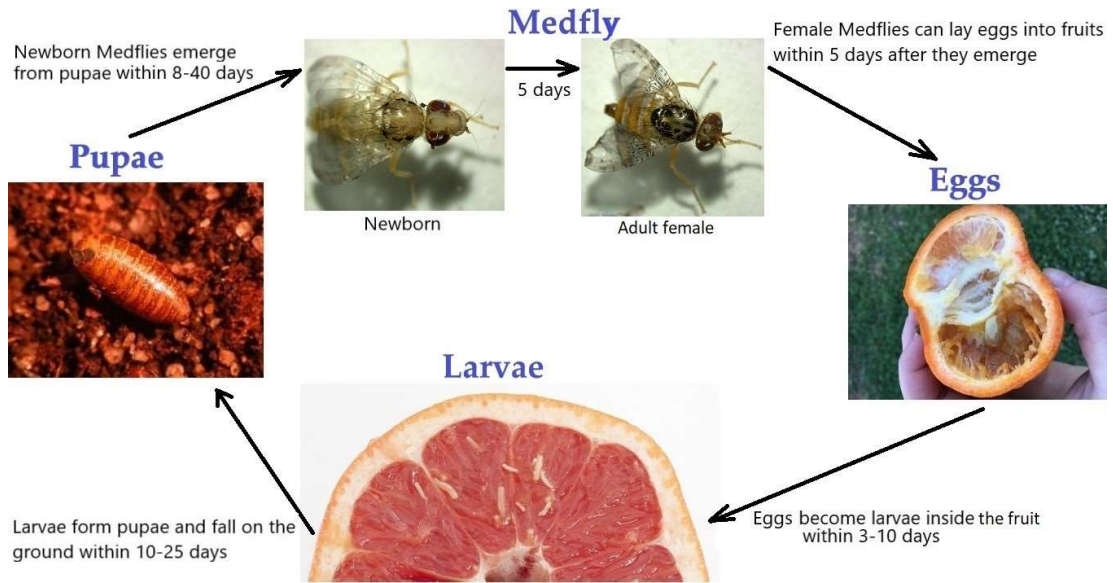


Figure 1.9. Life cycle of Medfly *Ceratitis capitata*. New born and adult female Medfly pictures courtesy of Mehmet Fatih Tolga; other pictures from (State of Western Australia, 2018). Damaged orange flesh picture and Medfly pupae in soil picture used with permission of the the State of Western Australia, Department of Primary Industries and Regional Development, WA; Medfly larvae in grapefruit picture (© owner Pia Scanlon) used with permission of the the State of Western Australia, Department of Primary Industries and Regional Development, WA.

Medfly may virtually damage up to 100% of crops in case no control mechanism is applied (Cohen et al., 2007). Thus, its existence in agricultural fields requires monitoring continuously and systematically. Automated remote monitoring is essential in today's conditions and this can be enabled by means of technological supported systems. Such systems are seldomly existent today but still billions of people who has no other alternative but to survive by cultivating relatively small land pieces cannot afford to buy them.

The Ministry of Agriculture and Forestry of the Republic of Turkey yearly underlines the initiative to combat Medfly and announces its support to scientists, firms and institutions that intend to spend efforts on innovative, economical and effective technological solutions. Within this context, Medfly experts who are authorized at the following ministry branches (Ministry of Agriculture and Forestry, 2019) are consulted regularly during this academical work:

- Republic of Turkey, Ministry of Agriculture and Forestry (T.C. Tarım ve Orman Bakanlığı), Ankara

- Agricultural Protection Research Institute (Zirai Mücadele Araştırma Enstitüsü), Bornova, İzmir, Turkey
- Biological Control Research Institute (Biyolojik Mücadele Araştırma Enstitüsü), Adana, Turkey
- Research Institute Horticulture (Bahçe Kültürleri Araştırma Enstitüsü), Alata, Mersin, Turkey.

Figure 1.10. shows a shot from a meeting regarding collective fight against Medfly coordinated by high level governmental authorities. This clearly proves how much importance is granted to Medfly in Turkey due to the huge risk it imposes on the economy and exports.



Figure 1.10. A meeting regarding collective fight against Medfly coordinated by the Ministry of Agriculture and Forestry with participation of Adana Governor (URL-3).

Within this context, the following important information has been acquired from the Medfly experts mentioned above:

Innovative technological methods are needed desperately to support effective fight against detrimental pests that cause huge harm to crops and reduce the export income.

The top pests are declared to be the Mediterranean fruit fly (*Ceratitidis capitata*), olive fly (*Bactrocera oleae*), and apple butterfly (*Cydia pomonella*). We often come accross the news about the negative effects to the international trading of fruits and vegetables. Ministry experts invite all concerned parties to from of an intelligent system that can detect and monitor harmful pests, Medfly being priority one, to effectively combat them. Besides, the same system could be useful for pesticide optimization as well. Same experts explain that many farmers apply pesticides more often than actually needed because of their Medfly fear since they lack an effective warning system. Please recall as explained in prior sections that Medfly may emerge at varied dates, depending on instantaneous weather conditions. Excessive pesticing causes many critical drawbacks: Additional cost, more harm to people who consume the fruits (due to chemical residues), more harm to the nature because pesticides eliminate beneficial wealth (like ladybugs, bees, etc.) together with harmful ones and causes groundwater pollution (Remboski et al., 2018). In short, pesticide optimization is extremely important in fighting pests in general and of course particularly Medfly.

Various types of commonly used special traps to catch Medfly to monitor their existence, and to pesticide just in time in order to decrease their population can be seen in Figures 1.11., 1.12. and 1.13.



Figure 1.11. A delta trap placed in the field to charm and catch Medfly (Sorhocam, 2021b)



Figure 1.12. Caught Medflies together with some other insects inside the trap (with permission from Mehmet Fatih Tolga)



Figure 1.13. A Jackson trap (State of Western Australia, 2018). ©2013 Sonya Broughton, with permission from the State of Western Australia, Department of Primary Industries and Regional Development, WA.

The Integrated pest management objective is to keep pest population under control to maintain damage to crops at a tolerable level. The most common technique for pest population monitoring is to place specific traps in the field and make regular site visits to visually investigate every trap. However, this is a costly and inefficient method of monitoring as it must be performed by human operators (Remboski et al., 2018; Hong

et al., 2020). For this reason, an innovative solution is needed to automate the following two actions to avoid the need for manned site visits to check and count the number of caught Medflies inside the traps:

- Recognize and count the number of caught Medflies in the trap and send a daily report,
- Roll up the sticky tape in the trap at need (in average once per month) to eliminate the necessity for manned visits to the fields for a full season.

During this academical study, an intelligent system has been formed to identify and count Medflies in trap pictures using ML approaches in general, particularly DL techniques.

1.1 Machine Learning

ML is a branch of artificial intelligence (AI) and computer science that imitates the way that humans learn (IBM Cloud Education, 2020). It explores the study and construction of algorithms that is

- Able to learn from past data
- Able to make future estimations based on learnt data
- Able to improve their performance through experience using training data (Akbas et al., 2019; Espinoza et al., Kashinathan et al., 2020; 2016; Xie et al., 2018; Patricio and Rieder, 2018).

Machine Learning is actually getting computers program themselves. Some tasks are too complicated to be programmed explicitly. Machine learning has been developed to solve complicated problems that are very hard or impossible to program by hand. In such situations where programming is the bottleneck, then a suitable machine learning enables the data to be used to construct a model that would do the work. Instead of writing a program by hand, we collect lots of examples that specify the correct output for a given input. A machine learning algorithm then takes these examples and produces a program that does the job.

ML learns from sample (example) data as the machine accesses the data. This is actually supervised learning using annotated images (URL-4).

Humankind has been using machines in order to support work for a quite long time. In recent years, humankind has started to use machines to support brain power owing to the evolution of machine learning,

In recent years, machine learning has become very effective and widely used in every field. Nowadays it is possible to build systems that can perform tasks on their own.

The following advantages have helped machine learning to be developed in recent years:

- Flood of available big data (very large data sets) owing to the progress in the Internet,
- Increased computation speed,
- Advance in existing algorithms and theory,
- Boosted contribution from the industry.

Common usage in today's life can be listed as follows:

- Pattern recognition
- Speech recognition, Natural language processing
- Web search engines
- Recommendation systems
- Medical Diagnosis
- Computational biology
- Finance & Credit
- Fraud detection
- E-commerce
- Space exploration
- Robotics, unassisted control of a vehicle
- Information extraction
- Social networks.

Unfortunately ML has limitations in solving very complex problems of computer vision tasks that mimic human sight and interpret images from their features. Instead, deep learning (DL) that is explained in next section proves much better performance in computer vision tasks owing to its extremely accurate neural network architecture.

ML requires hundreds or thousands of original or augmented input data to produce acceptable accuracy levels. On the other hand, relatively fewer labeled and annotated images are often sufficient to train a DL algorithm. Without DL, computer vision wouldn't have achieved such high success rates indeed. The drawing in Figure 1.14 illustrates a ML and DL comparison (URL-5).

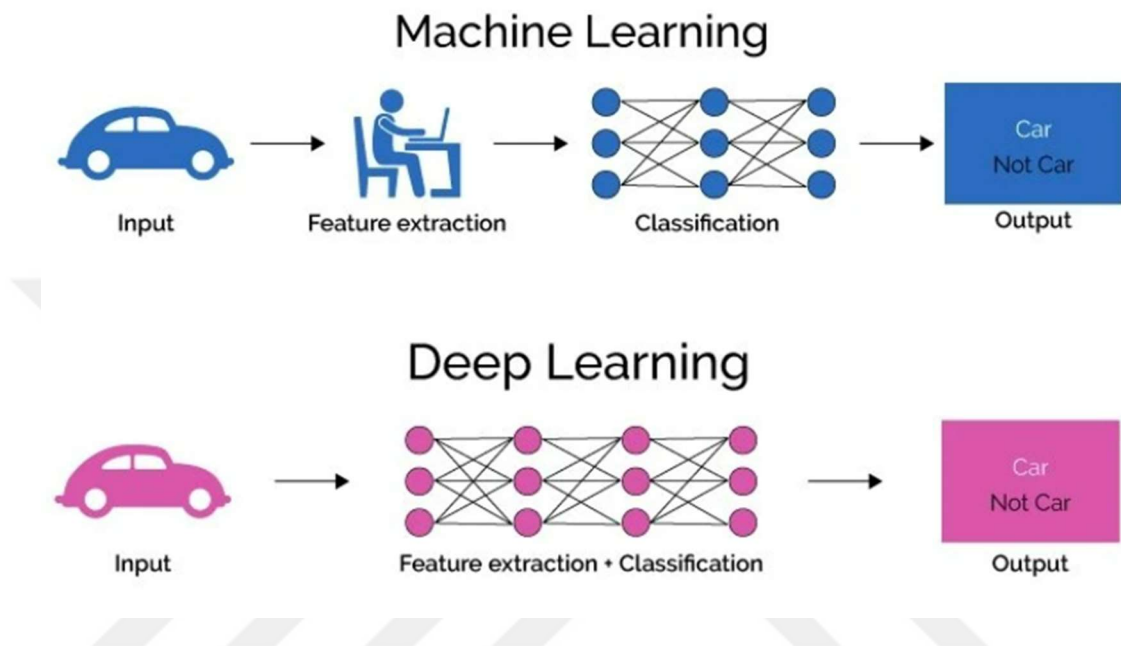


Figure 1.14. Machine learning vs deep learning (URL-5 with permission from viso.aiAG)

1.2 Deep Learning

Deep Learning (DL) is a special type of machine learning as it involves learning in different stages (URL-5). It consists of machine learning models based on deep convolutional neural networks that simulate human decision making. DL trains the computer algorithms to do what the human brain does naturally. DL has a structure of layers. In each layer, knowledge is added to the previous layer. DL is expensive and work intensive since it requires massive datasets and big computing resources (GPU) to train models.

DL methods may use supervised or unsupervised learning but supervised learning has become the standard in computer vision applications. Such tasks require a big

computation power of GPU, thus this power has been increasing in recent years. Extremely big amounts of data are needed for training and validation data so image labelling and annotation process is labor intensive and time consuming. Owing to the development of big data resources, labeled data has become available from many benchmark datasets such as V5, Pascal Voc, Caltech, Kitti, Ms Coco etc. As a result, DL based object detection has found a wide variety of usage among researchers and companies to form commercial products to be used in daily life (URL-4)

A DL algorithm can be taught to detect a specific animal, for instance a dog. This normally costs a massive number of images to differentiate the minor details to distinguish a dog from other animals. Very common commercial applications of DL are visual face recognition and fingerprint recognition used in security fields. A face recognition as a deep learning example can be seen in Figure 1.15.

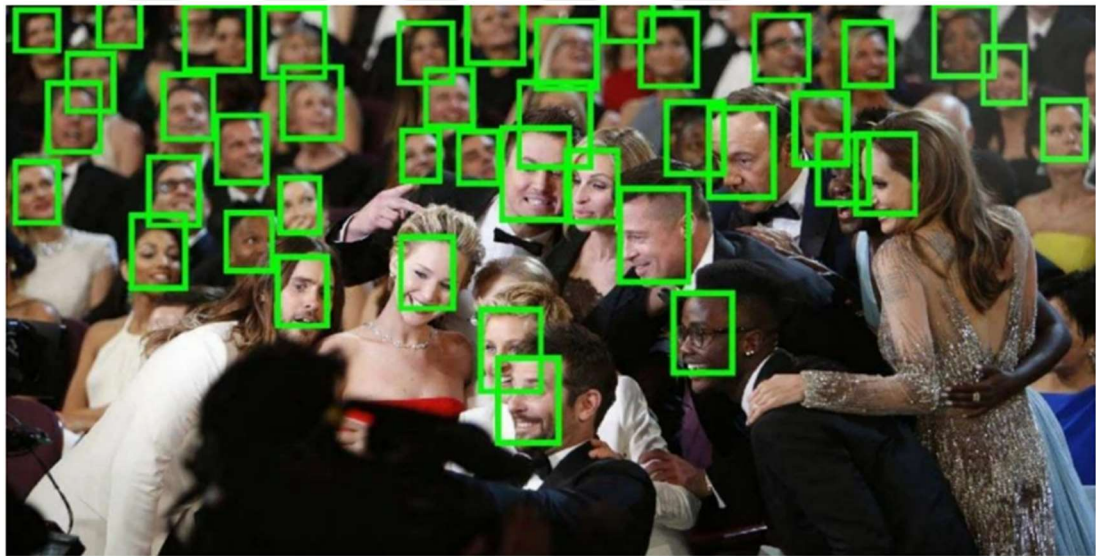


Figure 1.15. Face recognition as a deep learning example (URL-5 with permission from Viso.ai AG)

Some other common applications of DL are text based searching, frame detection, fraud detection, handwriting and pattern recognition, image search, vehicle detection, etc. Evaluating customer preferences for personalized advertising is another application that we often encounter in daily life. A vehicle detection application is illustrated in Figure 1.16.



Figure 1.16. Vehicle detection (with permission from Viso.ai AG)

1.3 Object Detection

Object detection is a recently developed technology of computer vision that utilizes image processing in detecting and identifying meaningful objects in digital images and video frames (URL-4).

Before DL was introduced in 2013, object detection was limited to classical ML techniques like Viola-Jones algorithm, scale invariant feature transforms (SIFT), histogram of oriented gradients etc. These techniques were based on detecting some common features in the image, employing logistic regression, color histograms or random forests to classify their clusters. Fortunately, recently developed DL techniques prove much better performance than these owing to the use of neural network architectures like You Only Look Once (YOLO), RetinaNet, SSD, CenterNet, region proposal feature detection and then identification based R-CNN, fast R-CNN, faster R-CNN and cascade R-CNN (URL-6).

Object detection consists of two consequent tasks: Detection is determining the places of the objects in the image and then classification is identifying what the detected objects are. This process is realized in three stages:

- i. Identify the region of interest
- ii. Perform automatic feature extraction

- iii. Classify the detected objects (Krohn et al., 2019)

Object detection has a large variety of applications today, like detecting pedestrians for autonomous driving, people counting, face detection, finger print detection for security, text detection, car plate detection, identifying abnormalities in medical images, defect detection, etc. (URL-4).

Object detectors draw bounding boxes around detected objects that help to locate them easily. A typical application of object detection is illustrated in Figures 1.17 and Figures 1.18.

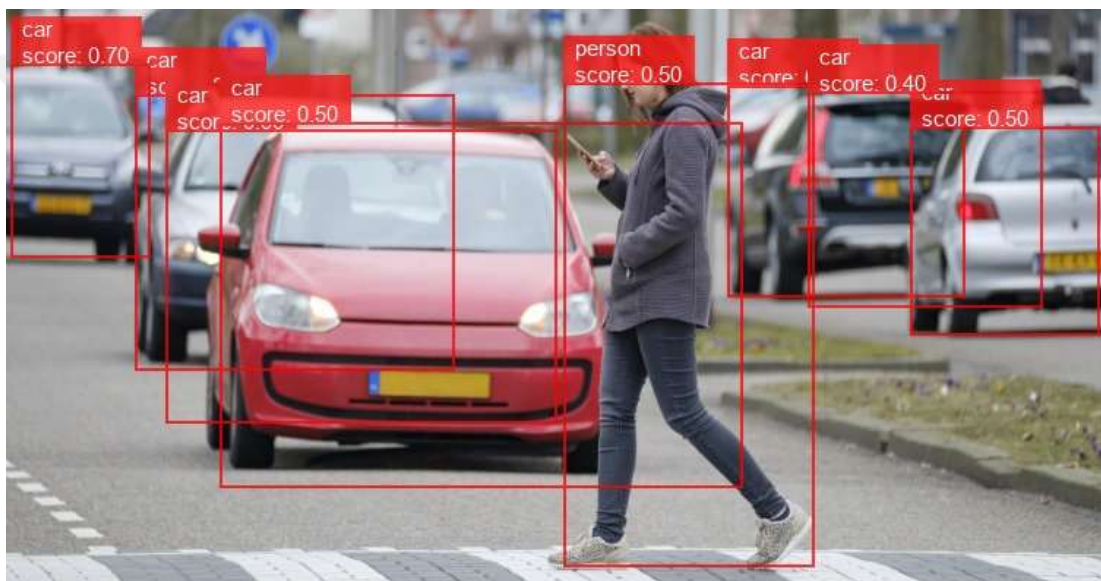


Figure 1.17. A typical object detection application, with permission from Visio.aiAG (URL-7)

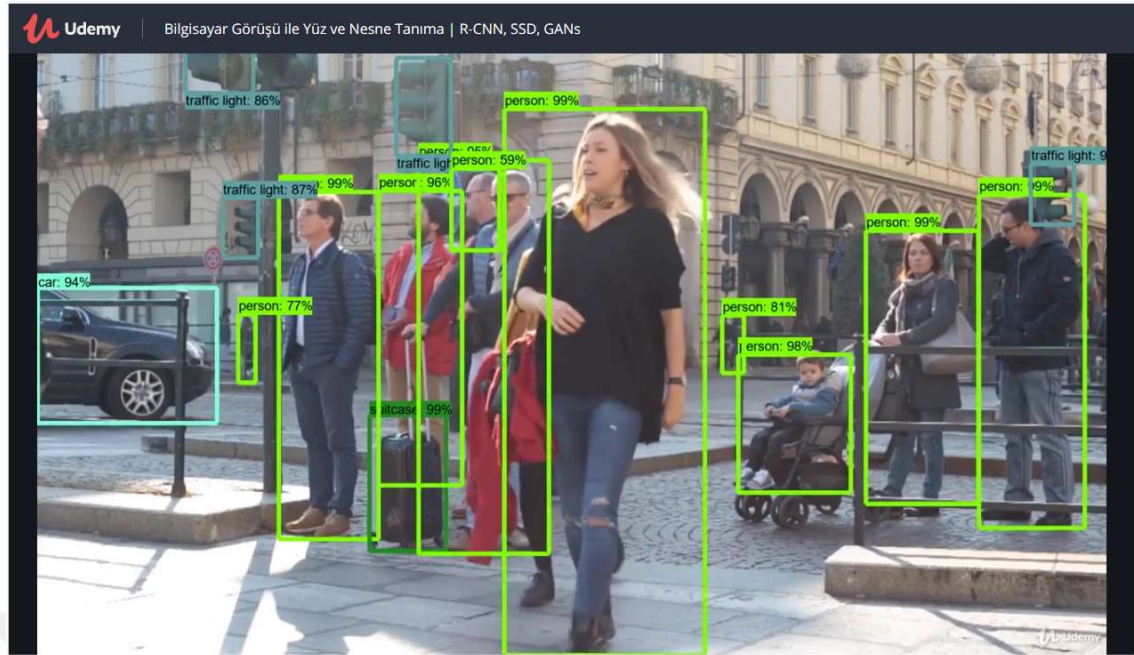


Figure 1.18. A typical object detection application, ©Udeemy instructors Ibrahim Cebeci and Rahmi Tuna (URL-8)

Important models that have helped making progress in object detection field are R-CNN, fast R-CNN, faster R-CNN and YOLO (Krohn et al., 2019).

1.3.1 Region based Convolutional Neural Networks (R-CNN)

Ross Girshick suggested R-CNN in 2013 together with his coworkers who work for UC Berkeley university. They aimed to model an algorithm that behaves like a human brain in such a way that the entire scene is scanned and focus is concentrated on specific regions of interest (ROI).

- i. Do a selective search throughout the image to determine ROIs
- ii. Use a CNN to extract features from all ROIs determined in step i.
- iii. Use ML techniques linear regression and support vector machines to decide the locations of bounding boxes and then classify objects within each box (Krohn et al., 2019).

Region based convolutional Neural Networks other name regions with CNN features are the most important models that apply deep learning in object detection. R-CNN models initially select proposed regions of an image such as anchor boxes. Next, they

label categories and bounding boxes such as offsets based on predefined classes specified by the program. Afterwards, they utilise convolutional neural networks to extract features from each proposed region by forward computation (URL-4).

R-CNN's caused a revolution in object detection because of their big performance gain over previous models. Even though R-CNN has started the new era of DL, it still needed improvement in certain aspects, especially speed:

- Not flexible since the input is fixed to a specific image shape.
- Needs intensive computation, thus it is not fast enough

Due to the massive computing load, it is not feasible to use R-CNN widely in practical applications.

1.3.2 Fast R-CNN

Ross Girshick aimed to solve the speed problem of R-CNN by developing fast R-CNN. The main improvement here is in the step of extracting features from ROI's (step ii above) because the CNN is run many times unnecessarily, once for each ROI. Fast R-CNN provides the opportunity to eliminate this repetition. The CNN makes a global look at the image and then uses extracted features for all ROIs at the same time to reduce unnecessary computing load and increase efficiency. The features extracted by the CNN are placed in a vector and then provided to a dense net along with ROIs and gives two outputs per ROI: Classification with a probability calculation to predict what the object is, and a bounding box to mark its location.

Fast R-CNN is certainly speedier than R-CNN owing to the reduction in computation workload by eliminating repetition of feature extraction for each ROI. However, it still has an important drawback of ROI search because fast R-CNN needs to produce many region proposals in selective search (described in step i above). To reduce excessive workload in region proposal step, faster R-CNN has its own way of solution (Krohn et al., 2019; URL-10).

1.3.3 Faster R-CNN

Faster R-CNN was developed by Shaoqing Ren and his colleagues who worked at Microsoft Research in 2015. It replaces the region proposal method from selective

search to a region proposal network (RPN). As its name suggests, it is speedier than fast R-CNN because the excessive workload that has to be spent in selective search step has been addressed with a bright solution. They decided to utilize the feature activation maps from model's CNN because they contain lots of contextual information of an image. These feature activation maps contain what is in an image and at which location it resides. Faster R-CNN utilizes these valuable information to propose ROI locations to realize all of the three steps at object detection process to form a unified model architecture that is swifter than fast R-CNN (Krohn et al., 2019).

The performance of CNN's on images is much more superior than classic neural networks because they benefit from local spatial coherence of the images (Ren et al., 2016). The meaning is adjacent pixels make sense altogether.

A significant boost in speed is recorded by decreasing the volume of required transactions by means of convolution and pooling. Faster R-CNN, modeled by Microsoft, is a deep convolution network that has found a broad usage in recent years in object detection. It behaves as if it was a unique joint end to end network that is able to estimate standing of different objects fastly and precisely. Because faster R-CNN has the ability to produce region proposals by means of its region proposal network, abbreviated as RPN, its speed of operation is a lot better than other conventional models such as selective search etc. (Gad, 2020).

RPN has been put forward inside the faster R-CNN object detection structure to cope with the operation time congestion of former cutting-edge object detection models (Galdames et al., 2018). Indeed, this is the primary characteristic that shortens the detecting time (Ren et al., 2016). The representation of RPN is presented below in Figure 1.19.

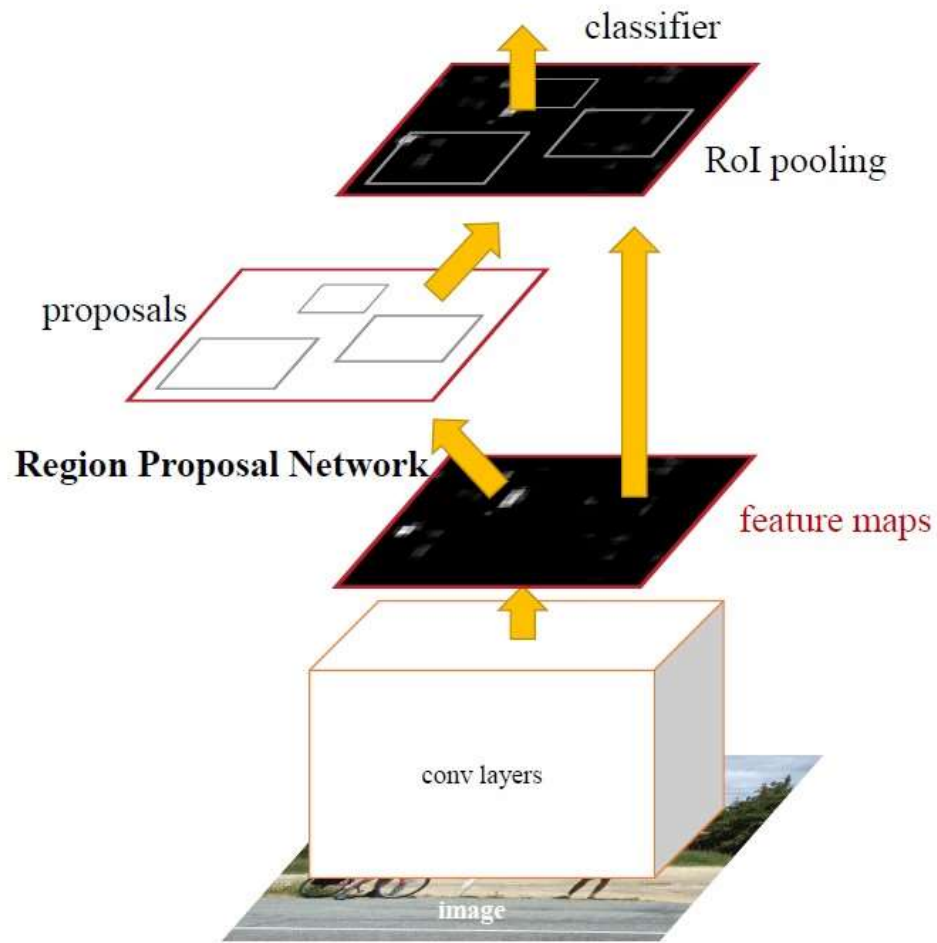


Figure 1.19. Region Proposal Network of faster R-CNN (Ren et al., 2016) with permission from Ross Girshick.

As an image processing tool, Faster R-CNN is used throughout this study in order to identify and enumerate caught Medflies in special purpose traps placed in agricultural fields. We got an encouraging outcome in terms of detection accuracy and economy of practical application of the solution in the field.

1.3.4 You Only Look Once (YOLO)

R-CNN type object detection models explained above have a one shot look at input image to extract features at the beginning and then is treated as a set of ROI proposals each ROI being a part of the input image. Joseph Redmon, Ali Farhadi and their colleagues suggested a different approach in 2015 named You Only Look Once (YOLO) that initiates with a pretrained CNN for feature extraction.

One only NN foresees class probabilities and bounding boxes straight through the whole image in a single assessment (Redmon et al., 2016).

The input image is split into a set of cells, and for each cell, a set of bounding boxes and object classification probabilities are estimated. Then, probabilities that are above a certain threshold value are chosen to be consolidated to locate objects within the image. By this way, only bounding boxes that have a reasonably high probability are allowed to contribute (Krohn et al., 2019).

YOLO algorithm proves a better performance than faster R-CNN in terms of speed, however, it is reported to be having difficulties in accurately detecting small objects. It is known that Joseph Redmon and his coworkers have released further versions in which they state that they improved accuracy and speed (Krohn et al., 2019). A sample YOLO output can be seen in Figure 1.20.

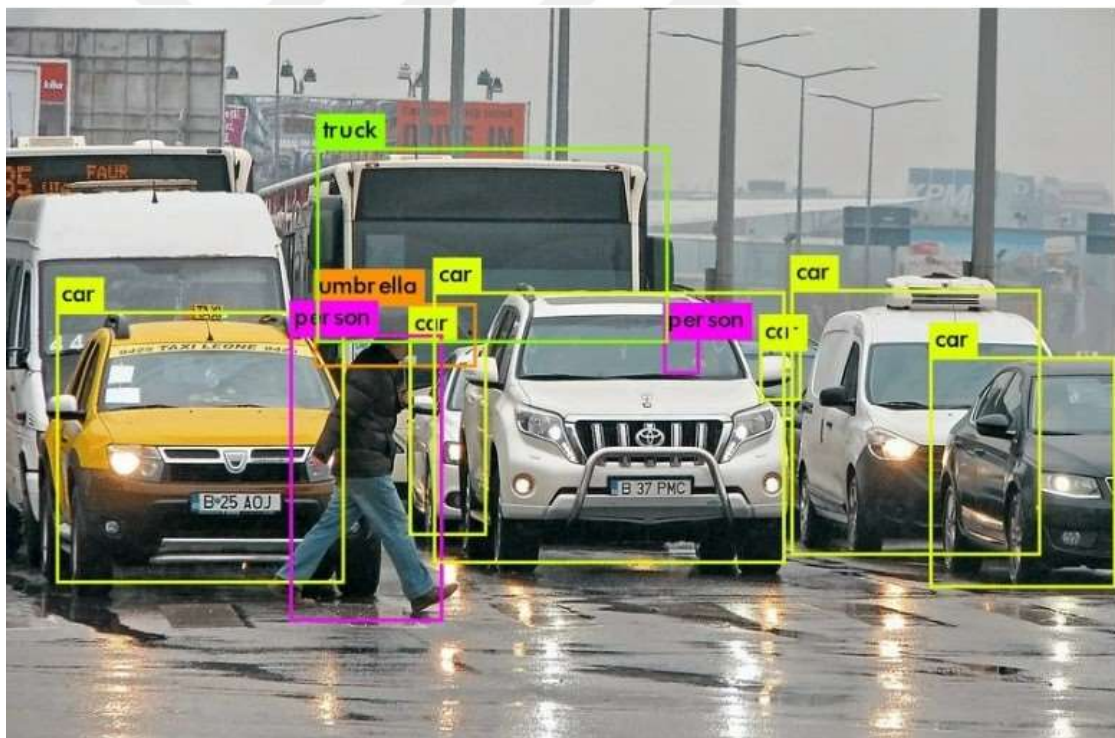


Figure 1.20. Real time object detection with YOLO (URL-9, with permission from Visio.ai AG)

2. LITERATURE REVIEW

Goldstein et al. (2017) note that the disadvantages of traditional monitoring, which relies on weekly site visits to manually count trapped medflies, result in a non-optimal spray frequency in citrus orchards. This drawback on one hand, and the scarcity of insect-tracking sensors on the other, has prompted them to contribute to efforts to develop an automated trap for remote insect monitoring, in particular Medfly. The authors designed a new cylindrical trap and built optical sensors specifically to detect and count trapped Medflies. From this article it becomes clear that scientific studies on automatic Medfly tracking traps have been a matter of recent years as they highlighted that in 2017 they developed the first self-knowledge automated trap for Medfly tracking. They also conducted field testing in commercial citrus orchards during five differing periods from 2013 to 2015 to specify the accuracy of their system based on optical sensors. They mention they concluded their work with an accuracy between 88% and 100%, and they come to the decision that daily monitoring by means of automated traps holds promise in minimizing pesticide application.

Hong et al. (2020) evolved models to identify three types of pest moths in pheromone trap images employing DL object detection methodologies to track the abundance and presence of pests and preserve plants. To this end, they utilised a meta-architectures combination such as region-based fully convolutional network, faster R-CNN, one-shot multi-box detector, and feature extractors such as Alexnet, Mobilenet, Inception, and ResNet. The faster R-CNN ResNet 101 detector produced the top accuracy of 90.25% among the seven combinations used by them.

On the other hand, Bekker et al. (2019) used ML to determine geographic drivers of Medfly trap capture in a farmland. Its ML based modeling performed up to 80% classification accuracy.

Apolo et al. (2020) used faster R-CNN in their model they constructed for fruit detection. They state that during their study, they achieved a mean standard error of only 6.59% between visual counting and fruit detection of their model, that could be considered an encouraging result.

The purpose of this work is to establish an intelligent system capable to detect the presence of Medfly to send timely alert to growers, orchard owners and farmers. It provides the benefits listed below:

- i. Monitor Medfly presence and send daily report (send SMS, email, etc.)
- ii. Assure just in time notification
- iii. Help to minimize insecticiding to decrease harm to human beings and to nature
- iv. Contribute to the economy by raising more food for less cost.

As a second task, the necessity to go to the field for a full agricultural yield season will be eliminated by automating adhesive tape replacement. Medfly experts explain that the adhesive tape begins to lose its viscosity in about one month because of flies, dust etc. So normally it needs to be replaced once per month. Also, if large numbers of flies are caught, the adhesive tape has to be replaced to keep the effectiveness afterwards. An intelligent system is needed that can be a solution for these issues. The particular e-trap designed during this study encloses a sticky band on a rollbar mechanism. It is easy to roll up this tape automatically by giving a command to the system when needed.

3. MATERIALS AND METHODS

3.1 Picture Acquisition

The intelligent system has been trained by 722 pictures and 150 others have been used for validation. Many of them enclose multiple Medfly images. Please beware that their sources are duly defined in Acknowledgements and References section below.

It is very important to emphasize that utilizing normal Medfly pictures widespread in the literature has not proven to be useful in intelligent system training because the deep learning system has to identify buckled Medflies in irregular shapes. Trapped Medflies become wrapped as a result of their fluttering aiming to free themselves so their shape is badly distorted. For this reason, it is essential to train the intelligent system with pictures of trapped Medflies similar to the ones in Figure 3.1. in order to obtain high accuracy rates. Our advantage in achieving a highly qualified training is that we extracted many pictures from videos regarding combatting Medfly. We have been able to find many such videos in the Internet.



Figure 3.1. Trapped Medflies in a picture from our dataset (extracted from a video with permission of Mehmet Fatih Tolga).

3.2 Dataset Annotation and Labeling

Raw data is identified by means of data labeling (images, text, videos) by binding description labels so that an ML modeling can utilize this description for learning (Amazon Web Services, 2021). Basic examples for labelling and annotation in ML are provided in Figures 3.2. and 3.3. Essentially, objects in dataset pictures are surrounded one by one by a rectangle and each object is annotated with a descriptive label in order to teach the intelligent system. As can be seen in Figure 3.2., the bicycle image is annotated by “bicycle”, the dog by “dog” and the car by “truck” identifiers.

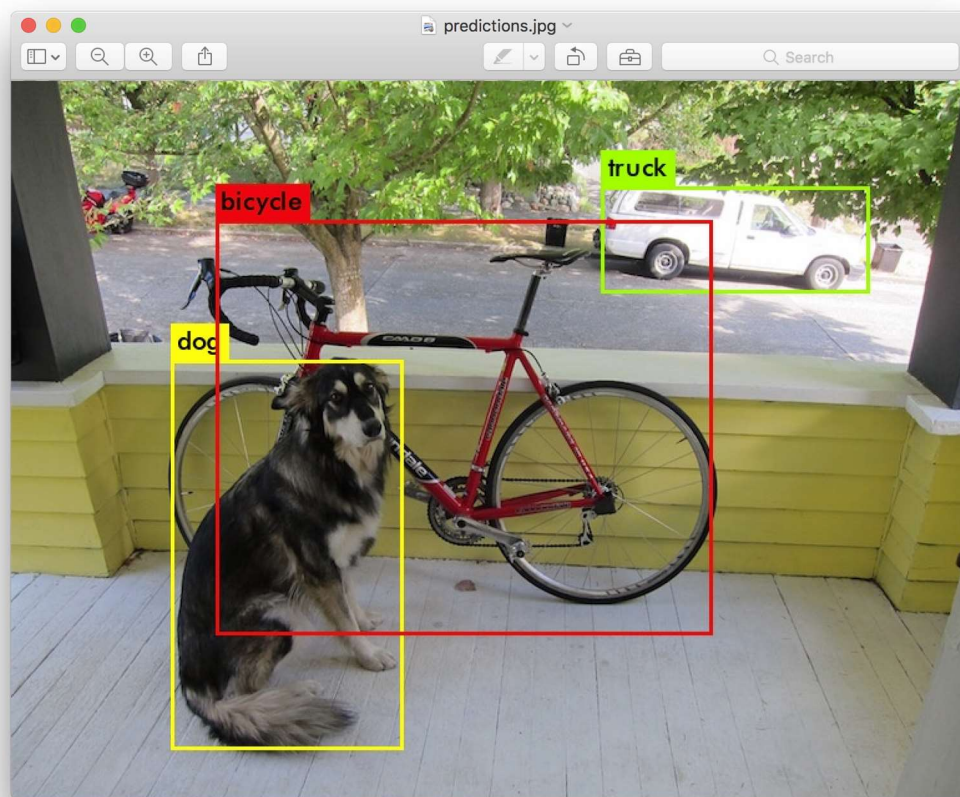


Figure 3.2. Labeling and annotating objects (URL-11) with permission from Joseph Redmon.

On the other hand, in Figure 3.3., all of the horses are annotated as “horse” no matter they have different colors.

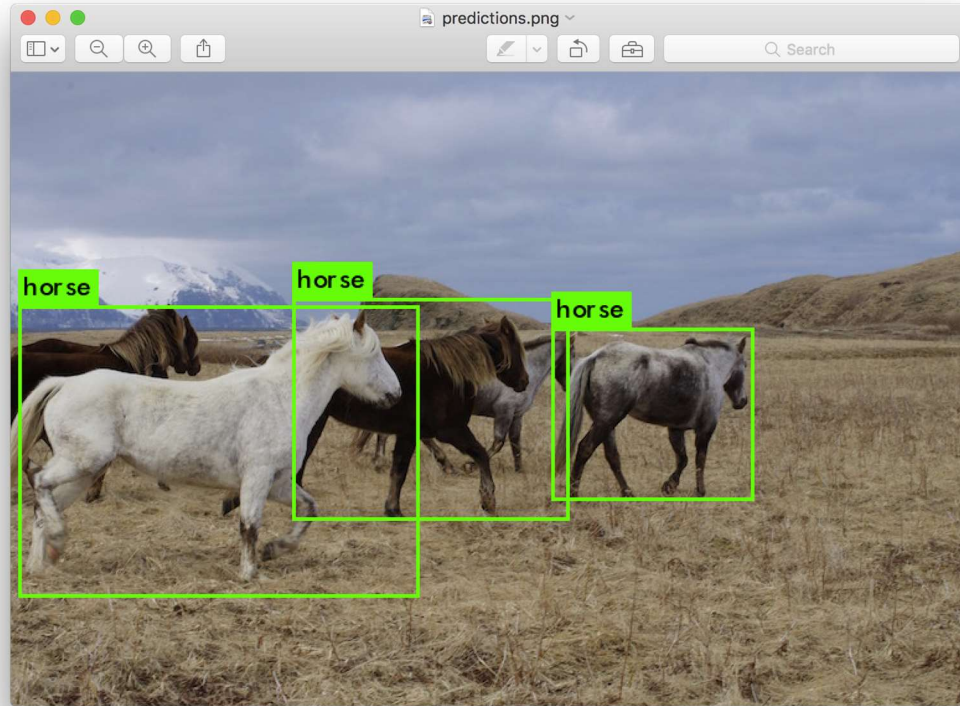


Figure 3.3. Labeling and annotating objects (URL-11) with permission from Joseph Redmon.

We have used the open source “Labellmg” program in order to carry out the labeling process of Medfly images in our dataset. It provides the opportunity to annotate images graphically and to label object bounding boxes (Github Inc, 2021). Medfly images in our dataset pictures are marked manually by the operator one by one by drawing a rectangle around it and are annotated by "Medfly" as can be seen in application cases 1 and 2 belonging to Figures 3.4. and 3.5. Please note that merely Medflies are labeled, i.e., other species are not labeled because our aim is limited to counting the number of Medflies only. We do not aim to count other species. Many pictures we have used as a dataset in this work include more than one images of Medfly.

A labeled and annotated picture from our dataset is presented in Figure 3.4. It can be seen that all Medfly images are surrounded by a rectangle one by one and described by "Medfly" affix. The whole dataset has been processed in this way.

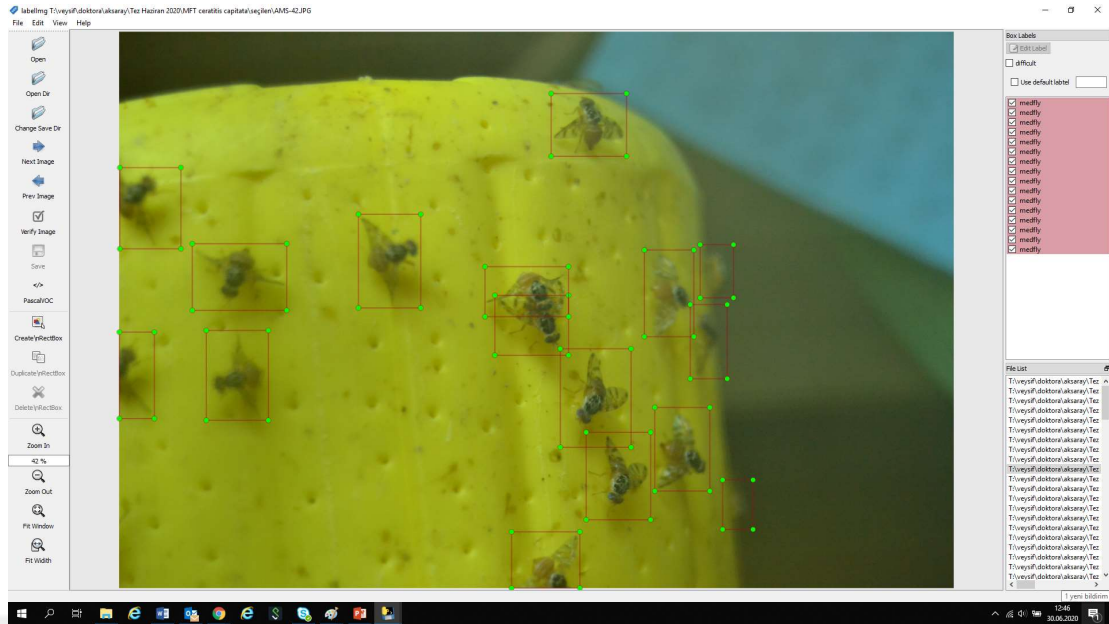


Figure 3.4. Sample case 1. Medfly annotation and labeling for training and validation (raw picture extracted from a video with permission from Mehmet Fatih Tolga).

Another labeled and annotated picture from our dataset is presented in Figure 3.5. All pictures in our training and validation dataset has been processed in this way.

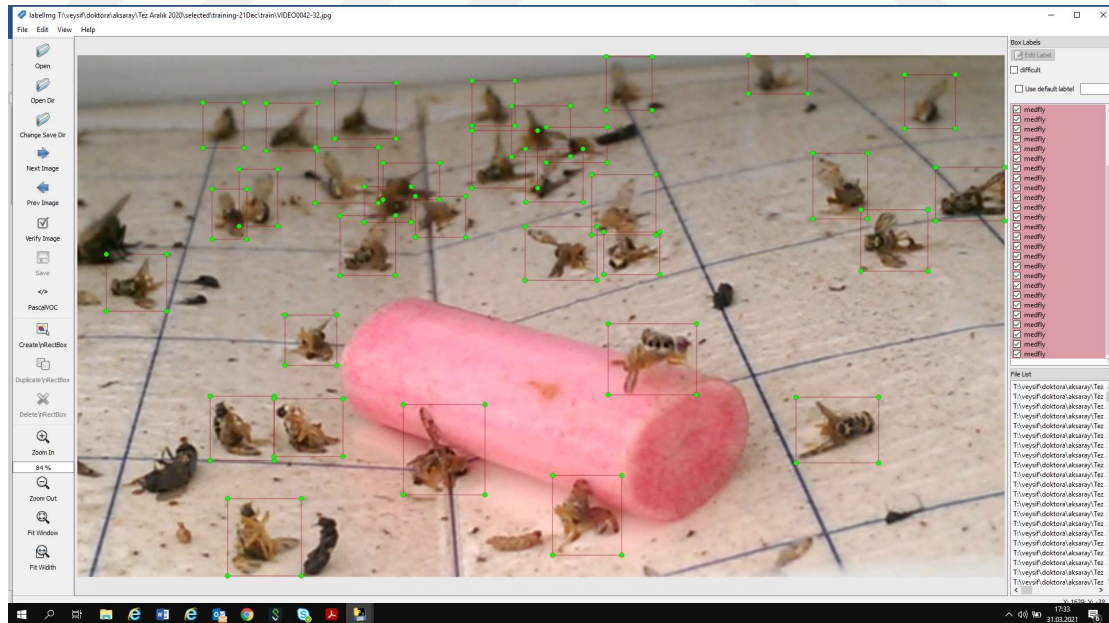


Figure 3.5. Sample case 2. Medfly annotation and labeling for training and alidation (raw picture extracted from a video with permission from Mehmet Fatih Tolga)

3.3 Deep Learning System Particulars

For convenience, Tensorflow software library for machine learning tool has been used throughout this work. Tensorflow is a free source library for ML and deep neural networks research developed by Google Brain Team that belongs to Google's Machine Intelligence research organization. It has a broad usage owing to its flexibility and inclusivity (Umruh A., 2017). Tensorflow among the most commonly used end-to-end and free source ML platforms that has a broad set of tools, resources and libraries. It is really suitable for constructing and deploying computer vision applications powered by ML. It is compatible with multiple programming languages like Python, C++, C, Javascript and Java. Tensorflow is among the simplest computer vision tools quite suitable especially for object detection, DL, facial recognition, image classification, etc. (URL-7).

As an image processing tool, we used faster R-CNN for object detection purposes. Python which is quite compatible to be utilized with Tensorflow has been selected to be the principal language for programming. A basic block diagram of the proposed system is presented in Figure 3.6. Note that each detected Medfly image has been marked by a blue star in the output picture, and their count has been printed as an output.

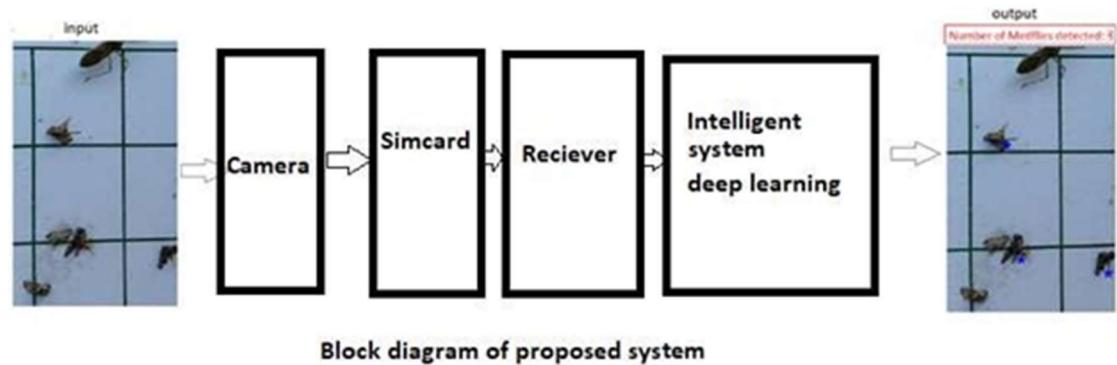


Figure 3.6. The architecture of the proposed system.

In object detection algorithms, mean average precision (mAP) is a broadly utilized accuracy measuring metric including faster R-CNN. It's calculation is simply made by estimation of the area under the curve of the precision recall relationship (Hong et al., 2020; Padilla et al., 2020; Jonathan H., 2018). Tensorflow's object detection API possesses an internally integrated module that can specify the mAP by getting use of

validation data. As can be seen in Appendix A, the mAP in this work has been calculated by the system assuming an intersection over union threshold value of 0.51. The outcome is explained in details in upcoming sections. Pictures in our validation data set mostly involve more than one Medfly images. So, in our data set consisting of 150 pictures in reality enclose hundreds of Medfly images, well enough sufficient for measuring the intelligent system accuracy.

It is possible to configure and train the the intelligent system formed in this study to identify and count multiple pests simultaneously. This point has been clarified better in Future Work section.

3.4 Image Preprocessing

All images in our training dataset have been subject to preliminary processing by the channel mean subtraction method in this study for enhancing the accuracy of detection. The primary initiative here is to make the network less sensitive to varying lightening and background conditions. Initially, the average of the pixels of all images in the training set is taken to calculate the per channel mean. The calculated per channel mean is then subtracted from every image to get the preprocessed images. These preprocessed new images formed to be used for training.

To subtract the dataset mean from each image is done fort he purpose of "centering" the data around zero mean for each channel (R, G, B). It is to decrease the quantity of data to be processed so as to allow faster learning. The network can be made less sensitive to varying background and lightening conditions by determining the "mean image" value of the training set and then subtracting it from validation, training and testing sets (Stackoverflow, 2016 and 2017).

3.5 Training the Intelligent System

In this study, an intelligent system capable to identify trapped Medflies is designed. When Medflies are caught, they flutter in panic aiming to free themselves from the sticky band of the trap. Thus, the shape of trapped Medfly is badly distorted, their wings, legs and their whole body is wrapped. The big challenge in this application is to enable the intelligent system to identify buckled Medflies with superior accuracies. Obviously, this can only be achieved by providing pictures that contain trapped Medfly images in irregular shapes for training and validation. Unfortunately, it is not possible

to acquire a big number of pictures of trapped Medflies in the literature because normally photos of best look are published. We started our trials by using a dataset consisting of regular Medfly images but initial result was not satisfactory in identifying distorted shaped Medfly as it resulted in an accuracy below 80%. Thus, it is understood that the need for trapped Medfly pictures for intelligent system training is inevitable to get satisfactory accuracy levels.

Even though pictures of trapped Medflies of distorted shapes are hardly obtainable in the literature, fortunately we have been able to find many videos that from the Internet regarding the fight against Medfly. These videos often display traps in which Medflies are caught. We have taken this opportunity for granted to extract many trap pictures in which Medfly images with distorted shapes are available and we have utilized them to establish training and validation datasets for this study.

Many pictures we extracted from videos obtained from the Internet had low resolution because most of the videos were quite old recorded when camera technology was not developed enough. Fortunately the result was still encouraging because still we have achieved very high accuracy rates in detecting trapped Medfly images. The influence of this fact has simplified our work in designing the e-trap.

Our training data set consisted of 722 training and 150 validation pictures. They mostly enclose more than one Medfly image so we can say that more than 2000 Medfly images have been used our intelligent system in total.

We prolonged our provisional training up to a 100K steps in order to specify the optimum number of training steps. By this way, we determined the optimum number of steps as 14.73K as illustrated in Figure 3.7. Please note that decreasing loss starts to increase after this point. In reality, the mAP meets optimum value at minimum loss. In the curve, when decreasing loss starts to increase after its minimum can be interpreted that the system is not learning anymore and overfitting has started.



Figure 3.7. Training the system 100K steps to determine the minimum.

Having this information in mind, in order to determine the optimum result, we carried out another training of 14.73K steps only. This time, the mAP has reached 94.05% as shown in Figure 3.8. which is a quite favarouble accuracy value.

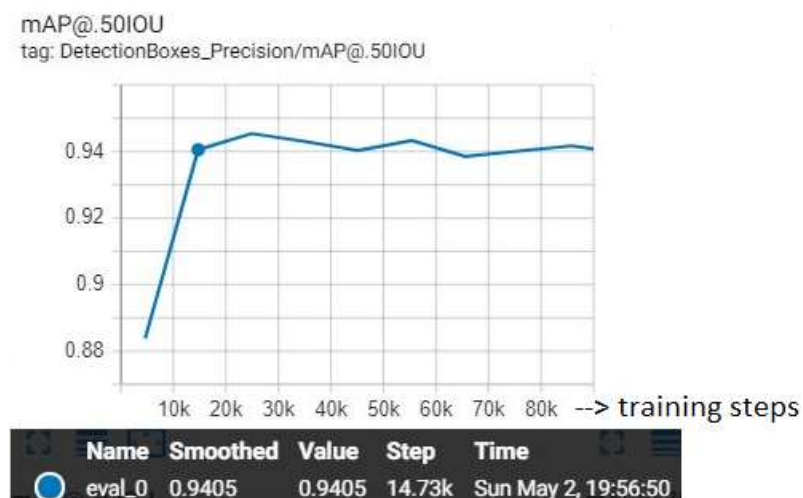


Figure 3.8. Intelligent system training optimization and determination of the optimum mAP.

4. RESEARCH FINDINGS

As shown in Figure 4.1, standard Tensorflow's object detector surrounds each detected object by a rectangle and prints the name of the identified object together with its calculated detection probability above the rectangle. In general, this illustration is fairly appropriate. However, our study has an extraordinary stand because trapped Medfly pictures generally contain many Medfly images intensified inside a small area like the ones in Figures 1.11., 1.12. and 1.13. In case many Medflies exist in a picture, so many rectangles and text will be placed in the output detector of standard Tensorflow. In such cases, so many rectangles will be mixed up and hard to distinguish which rectangle belongs to which image. Normally, flies show existence in extremely big numbers in nature, so mostly this is the case in our application. To solve this problem, we have modified the open source code of the Tensorflow program intentionally. For this purpose, we modified the intelligent system to act as follows: Whenever a picture that includes Medfly images is processed in our intelligent system, the modified Tensorflow object detector places a blue star on every identified Medfly image and prints out the count of identified Medfly images. The modified Tensorflow source code files (`visualization_utils.py` file and `Object_detection_image.py` file) are attached to this document in Appendix A and B below.

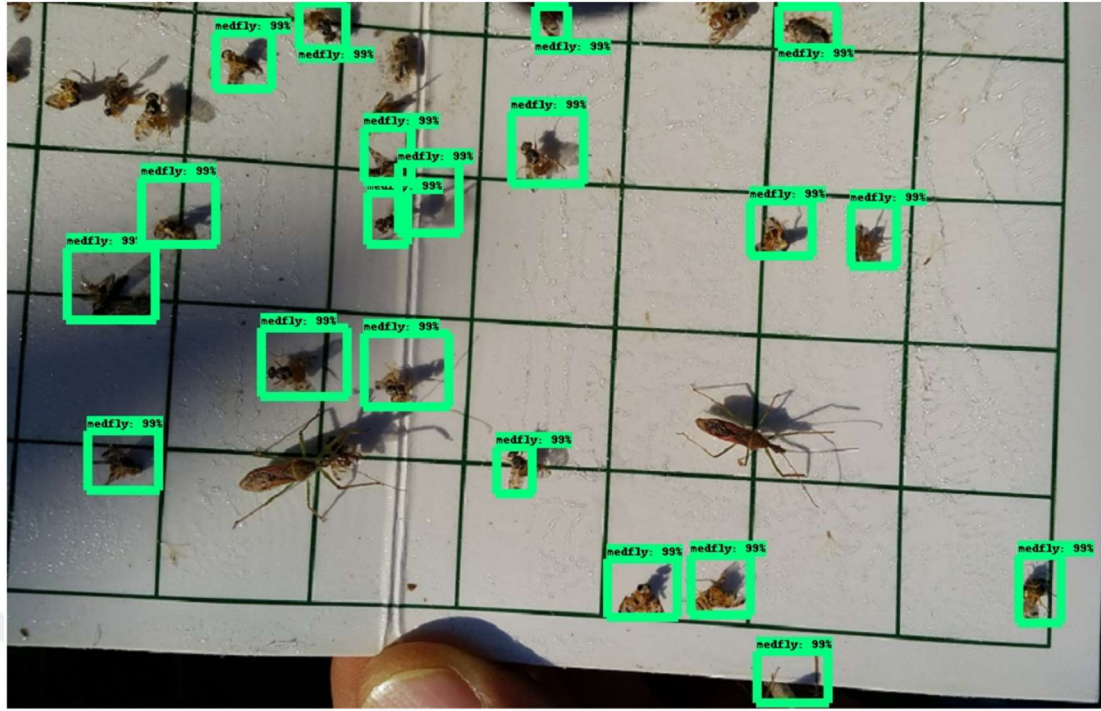


Figure 4.1. A sample output of standard Tensorflow (Raw picture with permission from Mehmet Fatih Tolga)

Figures 4.2., 4.3., 4.4., 4.5., 4.6. and 4.7. display the outputs printed by the object detector of our intelligent system when pictures that contain multiple Medfly images are given as input. These are outcomes of the Tensorflow that has been modified specifically for this study to fit requirements better: Blue stars are placed on each identified Medfly and the total number is printed out.



Figure 4.2. The intelligent system puts a blue star on every identified object and prints the count. Raw picture from (Picbear, 2020).

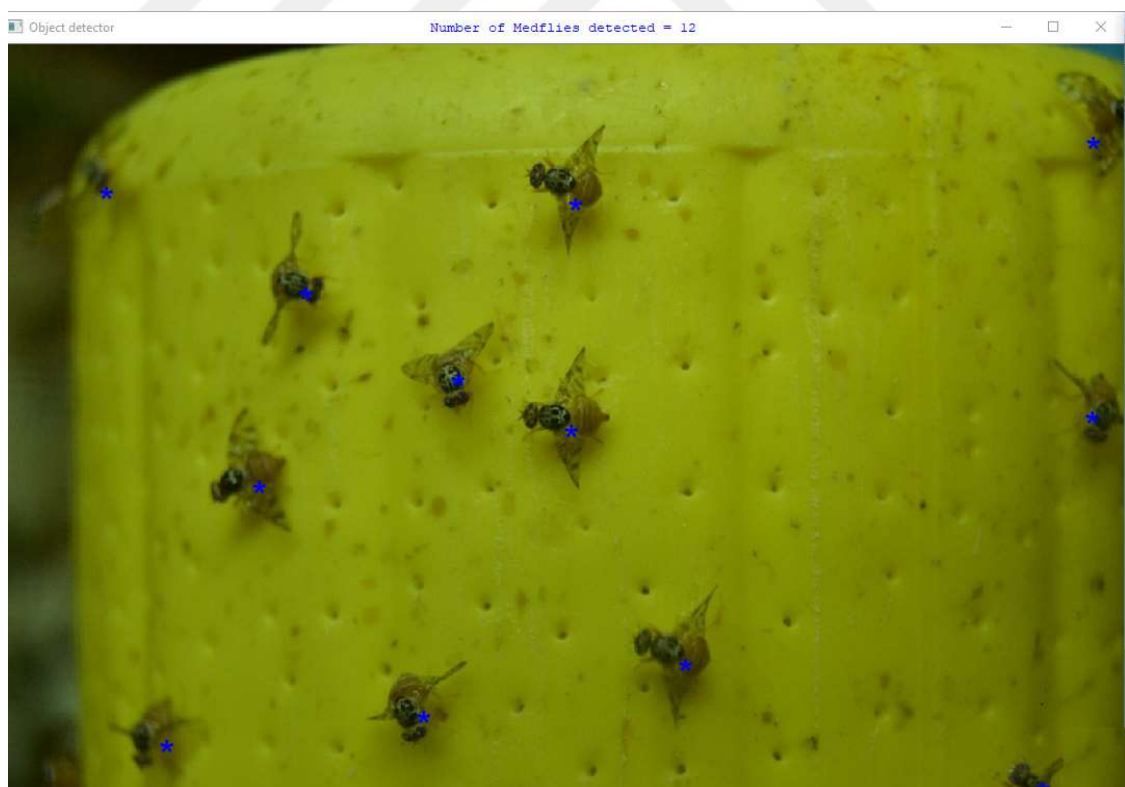


Figure 4.3. The intelligent system puts a blue star on every identified object and prints the count (Raw Picture with permission from Mehmet Fatih Tolga).



Figure 4.4. The intelligent system puts a blue star on every identified object and prints the count. Raw Picture from (ZF, 2010).



Figure 4.5. The intelligent system puts a blue star on every identified object and prints the count. Raw picture from (Eliza W., 2017).



Figure 4.6. The intelligent system puts a blue star on every identified object and prints the count. Raw Picture from (Estres vegetal, 2020).

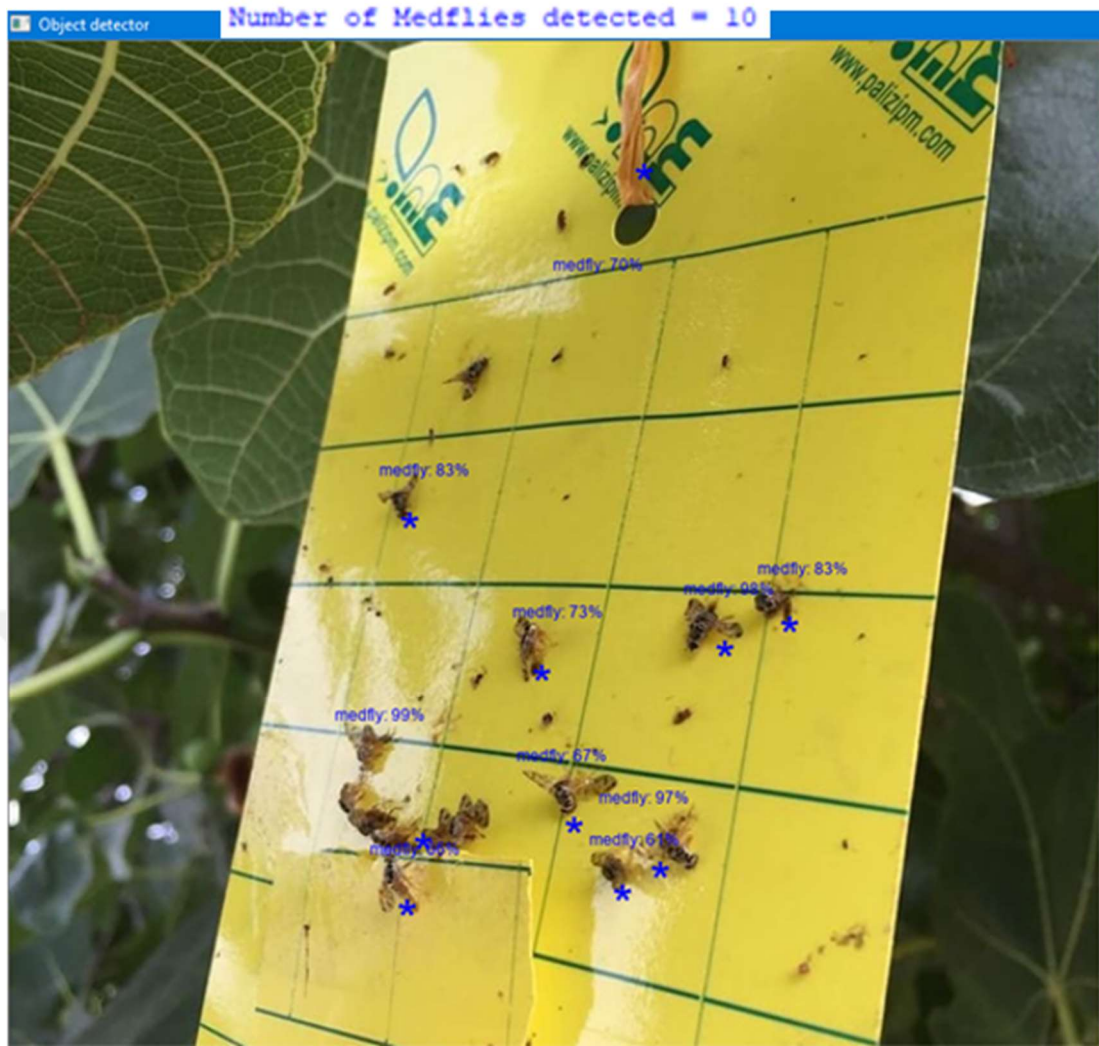


Figure 4.7. The intelligent system puts a blue star on every identified object and prints the count. Raw picture from (Picbear, 2020)

As stated before, our training data set consisted of 722 training and 150 validation pictures most which containing more than one Medfly images. We are very glad to have being achieved an overall detection accuracy of 94,05%. An interesting point worthy to mention is that preprocessing our training images using the channel mean subtraction solely contributed by 1.5% to the overall accuracy. We can justify the reasining by the following sentences:

The channel mean substraction method could not make a significant increase because the accuracy of the intelligent system was already very high anyway. This situation has made it hard for any preprocessing tool to make significant increases because as the accuracy value climbs towards the top point, it becomes harder and harder to incr We could not pass this point without mentioning the following amazing case that we

noticed during analyzing the results of processing our validation set. In Figure 4.8., the black color box has been drawn manually for labeling the existing Medfly image but the other green ones are drawn by the intelligent system during verification. The meaning is that the intelligent system has detected two Medfly images in this Picture but why? Because the intelligent system has detected not only the Medfly itself but also its shadow! This case proves that our intelligent system is so precisely functioning that it is able even to detect the shadow of a fly. Please notice that calculated probability for each case is also printed as 99% and 61% respectively and the count outputed as 2.

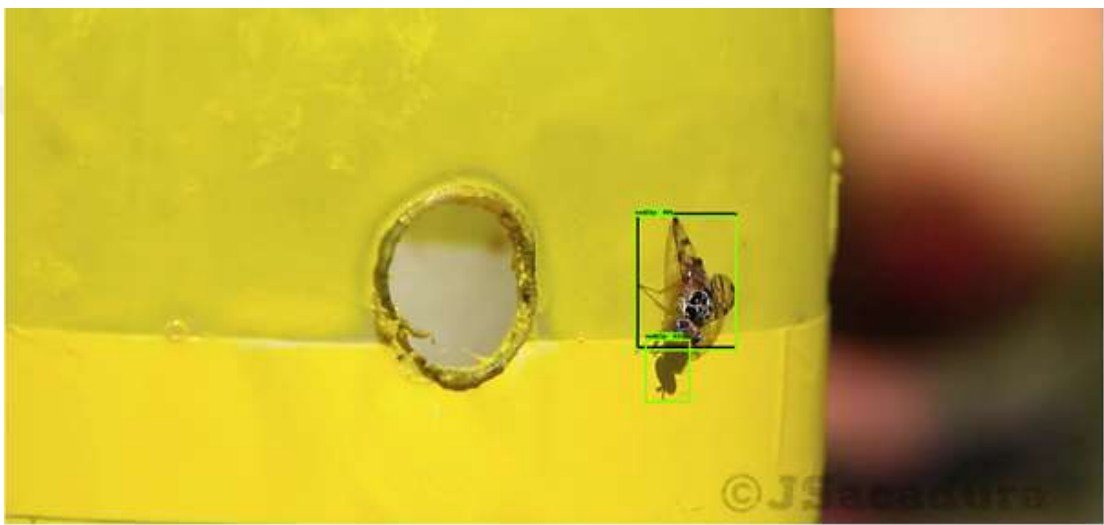


Figure 4.8. An amazing case: Our intelligent system is precise enough to detect the shadow of a Medfly. Raw Picture from (Sacadura J., 2020).

It is worthwhile to state that the object detector of our intelligent system can be set to print the calculated probability of each detected Medfly whenever needed or desired. Such outputs are presented in Figures 4.9., 4.10., 4.11. and 4.12. This is a standard feature of Tensorflow software library for machine learning tool indeed but it has been disabled in our application because it is not useful in cases where the picture to be examined contains a large number of detected objects. If a picture contains more than 20 Medfly images, then there will be a crowd of probability texts in the output and it will not be possible to differentiate which probability text belongs to which image. Such a situation can be seen in Figure 4.10. Even though this picture contain only 4 Medfly images, probability texts overlap so it is hard to differentiate.

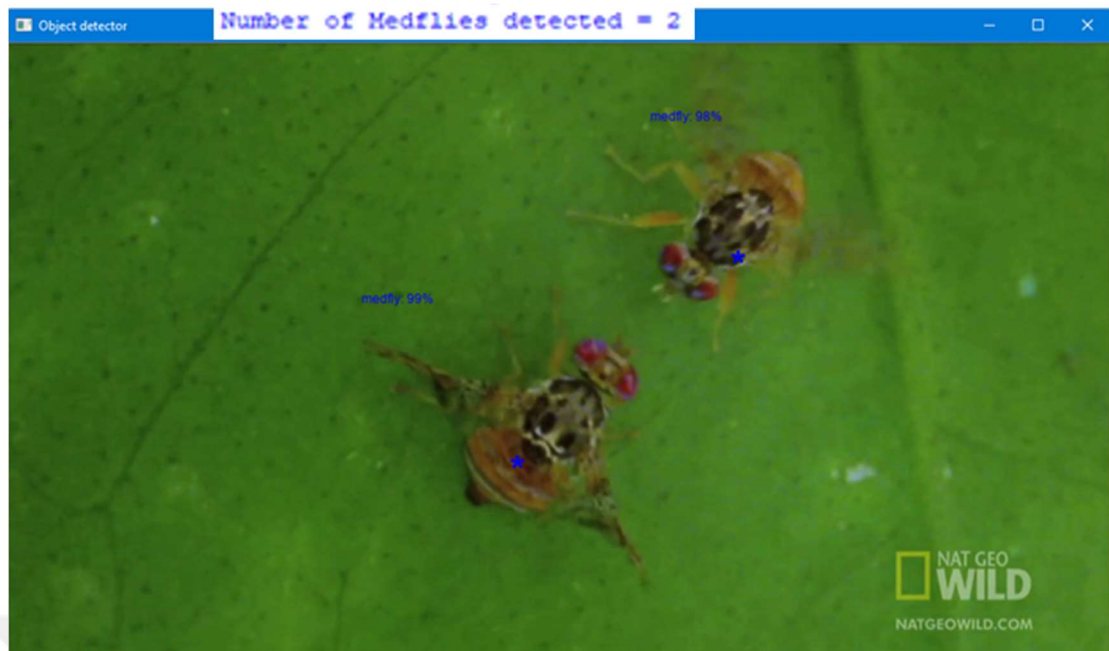


Figure 4.9. The object detector of our intelligent system can be set to print the calculated probability of each detected Medfly (Raw Picture extracted from a video of NAT GEO WILD natgeowild.com)

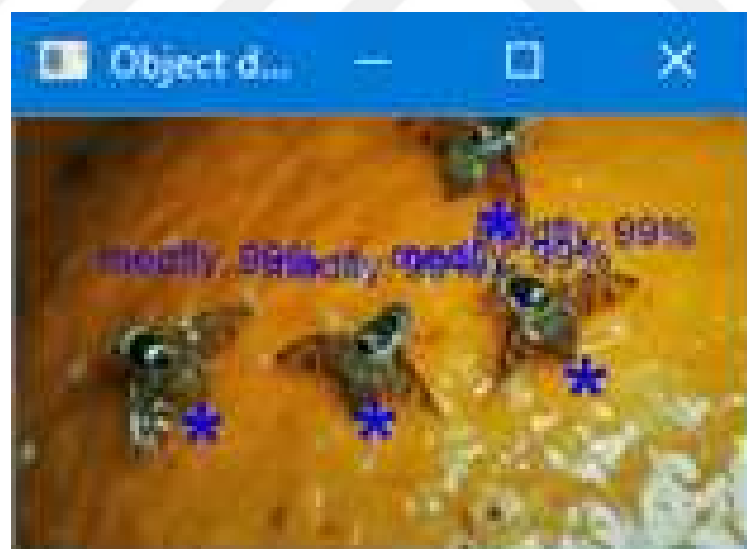


Figure 4.10. The object detector of our intelligent system can be set to print the calculated probability of each detected Medfly. Raw picture from (Eppo Global database, 2020)

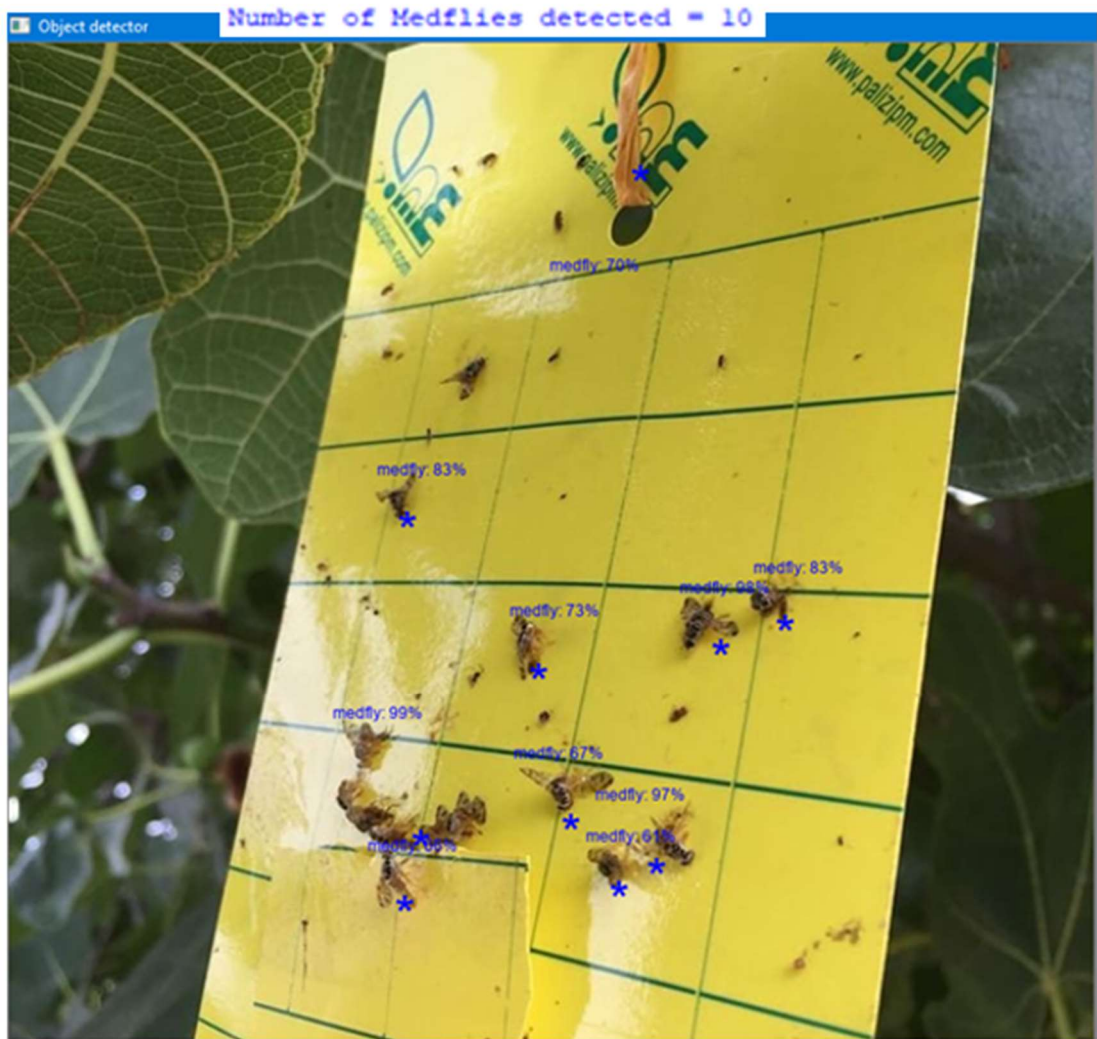


Figure 4.11. The object detector of our intelligent system can be set to print the calculated probability of each detected Medfly (Raw Picture from Mehmet Fatih Tolga)

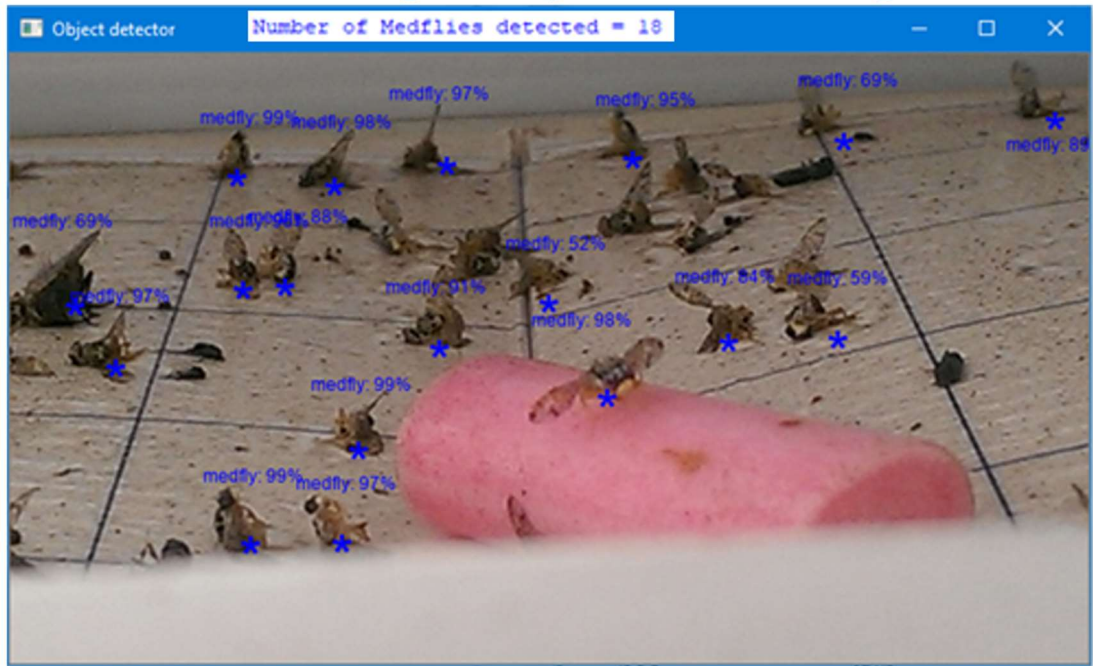


Figure 4.12. The object detector of our intelligent system can be set to print the calculated probability of each detected Medfly (Raw Picture with permission from Mehmet Fatih Tolga).

5. DISCUSSION AND CONCLUSIONS

There are many studies regarding automatic remote insect monitoring in the literature. The reason is obvious that traditional tracking that necessitates human site visits is not only uneconomical but also has serious drawbacks in this age and obstructs large scale field operations (Li et al., 2021; Ding et al., 2016; Sun et al., 2018). For this reason, scientists worldwide have drawn their attention to eliminating this difficulty. Some of these scientists get use of sensors for monitoring insects (Rustia et al, 2020; Goldshtein et al., 2017; Moraes et al., 2019). But the fact that these purpose specific sensors are scarce added to the rapid enhancements in deep learning, has been forwarding the tendency towards DL systems in recent times (Wenyong et al., 2021).

As detailed in the introduction section above, Goldshtein et al. (2017) specify that they used optical sensors to identify and count Medflies caught in traps. Their work resulted in an accuracy between 88% and 100%, and they come to the decision that monitoring daily by means of automated traps holds promise in minimizing pesticide applications.

Hong et al. (2020) utilized DL object detection tools for tracking pests and made a performance comparison among combinations of meta architectures like region-based fully convolutional network, faster R-CNN, single shot multibox detector jointly with several feature extractors. They report that they achieved the top accuracy of 90.25% with faster R-CNN ResNet 101 detector.

In another work, Bekker et al. (2019) employed ML methodologies for determining the geographical drivers of *Ceratitidis capitata* trap catch in agriculture fields resulted in classification accuracy as high as 80%.

Apolo et al. (2020) used faster R-CNN to form a model for fruit identification. They declare that they came up with an average standard error of only 6.59% between their modelling and visual counting.

In this academical study, the accuracy of detecting Medfly in a trap in which other pests as well are present is 94.05%. The highly qualified target oriented photographic dataset we collected from variety of resources we used for DL training and validation process enabled us to achieve such a high accuracy results. Comparing this result with

similar studies existing in the literature, we can conclude that this result is highly satisfactory.

Having made an intelligent system that can identify Medflies within a picture and print their count as an output available, the next step is to construct a particular e-trap that contains a webcam integrated with an electronic controlling circuit that has a mobile SIM card. The webcam is stationed at the top of the trap and oriented to have the best look at the viscous band where pests are caught. As explained earlier, it is needed to take a picture of the sticky band only once per day preferably slightly after noon time to benefit from daylight and to give enough time for Medflies to be caught if they exist at that day. These arrangements are purposely done to minimize the need for electrical energy. Since the webcam will be needed only once per day for a duration of a few seconds, it is not needed to be kept active the whole day. The system is to be programmed to stay passive the whole day except for being activated for a few seconds only for taking the photograph and transferring it over the SIM card to the main server in the center. The viscous band needs to be rolled up only once per month so this action would spend very little energy from the battery. This way, there will be no need for advanced battery systems or solar panels etc., and a simple battery will be enough for an agricultural season. The battery can be replaced or recharged at the beginning of the new agricultural season. The aim is to keep the system cheap so that it can be purchasable by more people. The selection of camera is also important to keep the cost low. The webcams available in the market seem to be quite advanced today. Exploring in the web, it is not difficult to see that cheap low end webcams have the capacity to take pictures of 1280x720 resolution which is quite sufficient for our purpose. Please recall we have explained above that some of the pictures, especially those we extracted from old videos related to fighting techniques against Medfly, used for our intelligent system training and validation had resolution less than this value. In spite of that we were able to achieve an overall accuracy of 94,05%. This fact together with the application case explained in Figure 4.8. simply prove how effectively our intelligent system is functioning even with low resolution pictures. The highly sensitive intelligent system at hand makes a low end cheap webcam to be used in the design of our special e-trap.

One more important point in designing aspects is to run the intelligent system software in a central server to serve all e-traps distributed in the field. This server can be configured to identify Medflies in Daily pictures received from e-traps and send the count to the user by email or SMS. By this way, farmers will need to buy the e-trap only without making any pre-payment for a hardware platform to run the software. A powerful central server can be operated by a company or a governmental organization and this server can serve hundreds of thousands of e-traps by charging a small periodic participation fee per unit e-trap. By this way, millions of farmers and orchard owners can be provided the opportunity to benefit from our automated remote monitoring system at affordable cost.

Thus, it can be concluded that the intelligent system suggested in this study together with its special pest e-trap holds the potential to open new horizons in fighting against Medfly and a variety of pests in general because the system can be configured to identify and count multiple kinds of objects simultaneously.

i. Broad use potential for all kinds of pests - The intelligent system designed and constructed as an outcome of this study can be configured and trained to monitor all sorts of pests or any other objects because it is suitable to identify multiple objects at the same shot.

ii. Economic aspect - Entirely automatic remote pest tracking systems are rarely existent nowadays and rather expensive. Thus, most cultivators can not afford to buy and maintain a fully automatic pest monitoring system. Farmers, Cultivators and fruit orchard owners can be provided with remarkable benefits if this system would be granted a broad use commercially. Estimates show that it can be offered to those who need it by paying only around 10% of the price of comparable remote pest monitoring systems existing in the market nowadays. This constitutes a big support especially to hundreds of millions of people worldwide who must survive by cultivating relatively small pieces of land which is hardly enough for their basic needs only and can not afford to buy other expensive systems.

The cost of our special e-trap equipped with a camera, controller circuit with integrated SIM card, and a suitable simple battery is estimated to be within the frame of several hundred Euros only. On the other hand, the price of a comparable automated pest

detection system produced in developed countries is in the range of several thousand euros, so it is far from being acquirable by most farmers in many poor countries.

As explained earlier, due to the lack of trapped Medfly pictures in the literature, many such pictures have been extracted from videos regarding combatting Medfly in order to be utilized as a dataset. Most of these pictures have really low-resolution values even less than 1280x720 pixels which is the resolution of today's lower-end cheap webcams available in the markets. If pictures with better resolution rates had been used for intelligent system training and validation, accuracy levels could have been improved even further.

5.1 Future Work

As a future work, it is possible to configure and train the intelligent system outcome of this study to identify and count multiple sorts of flies or pests in general at the same time.

Even though the intelligent system in this academical work has been designed to detect Medfly only, it is not difficult to be set and trained to detect multiple pests and if needed other objects at the same time. In such a case, the intelligent system processes a Picture only once and gives the numbers of each object separately in one single output. An example program class in Python is presented in the following label map illustrated in Figure 5.1. where a loop of 6 different harmful flies is written. This loop can be extended to contain as many objects as wished. As should be expected, certainly training and validation datasets have to be enlarged to include all of these objects accordingly.

```
60     #label map
61     def class_text_to_int(row_label):
62         if row_label == 'Medfly':
63             return 1
64         elif row_label == 'OliveFly':
65             return 2
66         elif row_label == 'AppleButterFly':
67             return 3
68         elif row_label == 'CherryFly':
69             return 4
70         elif row_label == 'MexFly':
71             return 5
72         elif row_label == 'PapayaFruitFly':
73             return 6
74         else:
75             None
```

Figure 5.1. A sample label map written in Python for identifying multiple pests simultaneously

REFERENCES

- Akbas, A., Yildiz, H., Ozbayoglu, M. and Tavli, B., 2019. Neural Network Based Instant Parameter Prediction for Wireless Sensor Network Optimization Models. *Wireless Networks*, 25, 3405–3418. 10.1007/s11276-018-1808-y.
- Amazon Web Services, 2021. What is data labeling for Machine Learning? Available from: <https://aws.amazon.com/tr/sagemaker/groundtruth/what-is-data-labeling/>, Accessed: 22 June, 2021.
- Apolo, O.E., Martinez-G., J., Egea, G., Raja, P. and Pérez-R., M., 2020. Deep learning techniques for estimation of the yield and size of citrus fruits using a UAV. *European Journal of Agronomy*, 115, 126030.
- Bekker, G., Addison, P. and Van Niekerk, A., 2019. Using machine learning to identify the geographical drivers of *Ceratitis capitata* trap catch in an agricultural landscape, *Computers and Electronics in Agriculture*, 162, 582–592. 10.1016/j.compag.2019.05.008.
- Cohen, Y., Cohen, A., Hetzroni, A., Alchanatis, V., Broday, D., Gazit, Y. and Timar D., 2007. Spatial decision support system for Medfly control in citrus. Elsevier B.V. *Computers and electronics in agriculture*, 62, 107–117.
- Ding, W. and Taylor, G., 2016. Automatic moth detection from trap images for pest management. *Computers Electronics in Agriculture*, 123, 17-28.
- Eliza, W., 2017. Call to cull falled fruit as Carnarvon mango season fire sup fruit fly pests, ABC News, Available from: <https://www.abc.net.au/news/rural/2017-01-18/carnarvon-mango-season-fires-up-the-fruit-flies/8191276>, Accessed: 26 March, 2020.
- Eppo Global Database, 2020. *Ceratitis capitata* (CERTCA), Agroscope FAW, Waedenswil (CH), Available from: <https://gd.eppo.int/taxon/CERTCA/photos>, Accessed: 22 April, 2020.
- Espinoza, K., Valera, D. L., Torres, J.A., López, A. and Molina-Aiz, F.D., 2016. Combination of image processing and artificial neural networks as a novel approach for the identification of Bemisia tabaci and Frankliniella occidentalis on sticky traps in greenhouse agriculture. *Computers Electronics in Agriculture* 127, 495-505.
- Estres vegetal, 2020. Cera Trap, Available from: <https://www.youtube.com/watch?v=s8HOGAF8vi4>, Accessed: 17 December, 2020.
- Fruitfly Africa, 2021. Life Cycle, Available from: <http://www.fruitfly.co.za/life-cycle-and-identification/life-cycle/>, Accessed: 20 June, 2021.

- Gad, A. F., 2020. Faster R-CNN Explained for Object Detection Tasks, Available from: <https://blog.paperspace.com/faster-r-cnn-explained-object-detection/>, Accessed: 04 April, 2021.
- Galdames, J. P. M., Milhor, C. E. and Becker, M., 2018. Citrus fruit detection using Faster R-CNN algorithm under real outdoor conditions. In Proceedings of the 14th International Conference on Precision Agriculture (unpaginated, online). Monticello, IL: International Society of Precision Agriculture, June 24 – June 2, 2018, Montreal, Quebec, Canada.
- Github Inc, 2021. LabelImg, Available from: <https://github.com/tzutalin/labelImg>, Accessed: 22 June, 2021.
- Goldshtein, E., Cohen, Y., Hetzroni, A., Gazit, Y., Timar, D., Rosenfeld, L., Grinshpon, Y., Hoffman, A. and Mizrach, A., 2017. Development of an automatic monitoring trap for Mediterranean fruit fly (*Ceratitis capitata*) to optimize control applications frequency, Computers and Electronics in Agriculture, 139, 115–125. 10.1016/j.compag.2017.04.022.
- State of Western Australia, 2018. Mediterranean Fruit Fly Identification, Available from: <https://www.agric.wa.gov.au/medfly/mediterranean-fruit-fly-identification>, Accessed: 06 April, 2020.
- IBM Cloud Education, 2020. Machine Learning. Available from: <https://www.ibm.com/cloud/learn/machine-learning>, Accessed: 19 May, 22.
- Hong, S.-J., Kim, S.-Y., Kim, E., Lee, C., Lee, J.-S., Lee, D.-S., Bang, J. and Kim, G., 2020. Moth Detection from Pheromone Trap Images Using Deep Learning Object Detectors. Agriculture, 10, 170. 10.3390/agriculture10050170.
- Jonathan, H., 2018. mAP (mean Average Precision) for Object Detection, Available from: <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>, Accessed: 18 November, 2021.
- Kasinathan, T., Singaraju, D. and Reddy, U. S., 2020. Insect classification and detection in field crops using modern machine learning techniques. Information Processing in Agriculture. 8. 10.1016/j.inpa.2020.09.006.
- Krohn, J., Beyleveld, G. and Bassens, A., 2019. Deep Learning Illustrated: A Visual, Interactive Guide to Artificial Intelligence, 182-186, Addison-Wesley Professional. (ISBN: 9780135116821)
- Li, W., Wang, D., Li, M., Gao, Y., Wu, J. and Yang, X., 2021. Field detection of tiny pests from sticky trap images using deep learning in agricultural greenhouse. Computers and Electronics in Agriculture, 183.
- Ministry of Agriculture and Forestry, 2019. Enstitüler, Available from: <https://www.tarimorman.gov.tr/TAGEM/Link/13/Enstituler>, Accessed: 23 June, 2019.

- Moraes, F., Nava, D., Scheunemann, T. and Rosa, V., 2019. Development of an Optoelectronic Sensor for Detecting and Classifying Fruit Fly (Diptera: Tephritidae) for Use in Real-Time Intelligent Traps. *Sensors*, 19, 1254. 10.3390/s19051254.
- Padilla, R., Netto, S. L. and Da Silva, E. A. B., 2020. A Survey on Performance Metrics for Object-Detection Algorithms. 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), 2020, 237-242, doi: 10.1109/IWSSIP48289.2020.9145130.
- Patrício, D.I. and Rieder, R., 2018. Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review. *Computers and Electronics in Agriculture*, 153, 69-81.
- Picbear, 2020. Trapped Medflies, Available from: <https://www.picbear.org/media/>, Accessed: 26 March, 2020.
- Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788, doi: 10.1109/CVPR.2016.91.
- Remboski, T. B., Souza, W., de Aguiar, M. and Ferreira, J. R., 2018. Identification of Fruit Fly in Intelligent Traps Using Techniques of Digital Image Processing and Machine Learning, SAC 2018: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 260-267. 10.1145/3167132.3167155.
- Ren, S., He K., Girshick, R. and Sun, J., 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 6, 1137-1149. doi: 10.1109/TPAMI.2016.2577031.
- Rustia, Dan J., Lin, Chien E., Chung, Jui-Y., Zhuang, Y., Hsu, Ju-C. and Lin, Ta-T., 2020. Application of an Image and Environmental Sensor Network for Automated Greenhouse Insect Pest Monitoring. *Journal of Asia-Pacific Entomology*, 23, 17-28. 10.1016/j.aspen.2019.11.006.
- Sacadura, J., 2020. An Effective Fruit Fly Trap for the Home Orchard, Reducing Medfly damage in apples and other fruits, Available from: <https://www.youtube.com/watch?v=aDBDdLxw0Hw>> Accessed: 17 December, 2020.
- SEDQ Healthy Crops S.L., 2020. Ceratipack: Use in mass trapping for the management of the Mediterranean fruit fly *Ceratitidis capitata*, Available from: <https://sedq.es/en/producto/ceratipack/>, Accessed: 11 April, 2020.
- Serantes, 2015. Peaceful Uses of Nuclear Energy Technology in Agriculture: The Sterile Insect Technique applied against the medfly *Ceratitidis capitata* in Spain, The momerandum made between IAEA and TRAGSA of Spain, Available from: [https://www.vie-mission.emb-japan.go.jp/3%20%20Serantes%20\(Spain\).pdf](https://www.vie-mission.emb-japan.go.jp/3%20%20Serantes%20(Spain).pdf), Accessed: 02 June, 2017.

- Sorhocam, 2021a. Akdeniz Meyve Sineği (*Ceratitis capitata*), Available from: <https://sorhocam.com/uploads/docs/akdeniz-meyve-sinegi-04547.pdf>, Accessed: 20 June, 2021.
- Sorhocam, 2021b. Akdeniz Meyve Sineği (*Ceratitis capitata*), Available from: <https://sorhocam.com/konu.asp?sid=142&akdeniz-meyve-sinegi-ceratitis-capitata.html>, Accessed: 10 June, 2021.
- Stackoverflow, 2016. Why normalize images by subtracting dataset's image mean, instead of the current image mean in deep learning? Available from: <https://stats.stackexchange.com/questions/211436/why-normalize-images-by-subtracting-datasets-image-mean-instead-of-the-current>, Accessed: 16 November, 2021.
- Stackoverflow, 2017. How does mean image subtraction work? Available from: <https://stackoverflow.com/questions/44788133/how-does-mean-image-subtraction-work>, Accessed: 16 November, 2021.
- Sun, Y., Liu, X., Yuan, M., Ren, L., Wang, J. and Chen, Z., 2018. Automatic in-trap pest detection using deep learning for pheromone-based *Dendroctonus valens* monitoring. *Biosystems Engineering*, 176, 140-150.
- The University of Arizona, 2021. The Mediterranean Fruit Fly *Ceratitis capitata* (Wiedemann), Available from: <https://cals.arizona.edu/crops/insects/fruitfly.pdf>, Accessed: March 07, 2021.
- Umruh A., 2017. What is the Tensorflow machine intelligence platform? Available from: <https://opensource.com/article/17/11/intro-tensorflow>, Accessed: 26 June, 2019.
- University of Georgia Center for Invasive Species and Ecosystem Health, 2020. Invasive Species, Available from: <https://www.bugwood.org/>, Accessed: 4 May, 2020.
- USDA NASS (United States Department of Agriculture, National Agricultural Statistics Service), 2012. Mediterranean Fruit Fly Pest Profile, Host, and Economic Importance, Available from: https://www.cdfa.ca.gov/plant/pdep/target_pest_disease_profiles/mediterranean_ff_profile.html, Accessed: 18 June, 2019.
- Wenyong, L., Tengfei, Z., Zhankui, Y., Ming, L., Chuanheng, S. and Xinting, Y., 2021. Classification and detection of insects from field images using deep learning for smart pest management: A systematic review, *Ecological Informatics*, 66, 101460, ISSN 1574-9541.
- Xie, C., Wang, R., Zhang, J., Chen, P., Dong, W., Li, R., Chen, T. and Chen, H., 2018. Multi-level learning features for automatic classification of field crop pests. *Computers and Electronics in Agriculture*, 152, 233-241.

- ZF, 2010. Mediterranean Fruit Fly (Medfly), Available from: <https://www.youtube.com/watch?v=56x7n1IS8Ns>, Accessed: 17 December, 2020.
- URL-1 <<https://www.agric.wa.gov.au/medfly/fruit-fly-monitoring-commercial-orchards>>, access date: Nov 1, 2022.
- URL-2 <https://entnemdept.ufl.edu/creatures/fruit/mediterranean_fruit_fly.htm>, date of access: Nov 1, 2022.
- URL-3 <<https://adana.tarimorman.gov.tr/Haber/509/Adanada-Akdeniz-Meyvesinegi-Ile-Mucadele-Toplantisi-Yapildi> 26.07.2022>, date of access: Oct 20, 2022.
- URL-4 <<https://viso.ai/deep-learning/object-detection>> date of access: June 20, 2022.
- URL-5 <<https://viso.ai/deep-learning/deep-learning-vs-machine-learning/>> date of access: June 21, 2022.
- URL-6 <<https://www.v7labs.com/blog/object-detection-guide>> date of access: June 21, 2022.
- URL-7 <<https://viso.ai/computer-vision/the-most-popular-computer-vision-tools/>>, date of Access: Oct 26, 2022.
- URL-8 <<https://www.udemy.com/course/bilgisayargorusu/learn/lecture/9729558#overview>>, date of Access: Oct 26, 2022.
- URL-9 <<https://viso.ai/computer-vision/the-most-popular-computer-vision-tools/>>, date of Access: August 16, 2022
- URL-10 <https://d2l.ai/chapter_computer-vision/rcnn.html>, date of Access: Oct 19, 2022.
- URL-11 <<https://pjreddie.com/darknet/yolo/>>, date of access: Aug 14, 2022.

APPENDIX LIST

APPENDIX A

APPENDIX B

APPENDIX C



APPENDIX A. Object-detection-image.py

File - T:\veysif\doktora\aksaray\Tez kitapçığı\ilk teslim sonrası\ekler\visualization_utils\Object_detection_image.py

```
1 ##### Image Object Detection Using Tensorflow-  
  trained Classifier #####  
2 #  
3 # Author: Evan Juras  
4 # Date: 1/15/18  
5 # Description:  
6 # This program uses a TensorFlow-trained classifier  
  to perform object detection.  
7 # It loads the classifier uses it to perform object  
  detection on an image.  
8 # It draws boxes and scores around the objects of  
  interest in the image.  
9  
10 ## Some of the code is copied from Google's example  
   at  
11 ## https://github.com/tensorflow/models/blob/master/  
   research/object\_detection/object\_detection\_tutorial.  
   ipynb  
12  
13 ## and some is copied from Dat Tran's example at  
14 ## https://github.com/datitran/object\_detector\_app/  
   blob/master/object\_detection\_app.py  
15  
16 ## but I changed it to make it more understandable to  
   me.  
17  
18 # Import packages  
19 import os  
20 import cv2  
21 import numpy as np  
22 import tensorflow as tf  
23 import sys  
24  
25 # This is needed since the notebook is stored in the  
   object_detection folder.  
26 sys.path.append("..")  
27  
28 # Import utilites  
29 from utils import label_map_util  
30 from utils import visualization_utils as vis_util  
31
```

```
32 # Name of the directory containing the object
    detection module we're using
33 MODEL_NAME = 'inference_graph'
34 IMAGE_NAME = 'test.jpg'
35
36 # Grab path to current working directory
37 CWD_PATH = os.getcwd()
38
39 # Path to frozen detection graph .pb file, which
    contains the model that is used
40 # for object detection.
41 PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'
    frozen_inference_graph.pb')
42
43 # Path to label map file
44 PATH_TO_LABELS = os.path.join(CWD_PATH,'training','
    labelmap.pbtxt')
45
46 # Path to image
47 PATH_TO_IMAGE = os.path.join(CWD_PATH,IMAGE_NAME)
48
49 # Number of classes the object detector can identify
50 NUM_CLASSES = 1
51
52 # Load the label map.
53 # Label maps map indices to category names, so that
    when our convolution
54 # network predicts `5`, we know that this corresponds
    to `king`.
55 # Here we use internal utility functions, but
    anything that returns a
56 # dictionary mapping integers to appropriate string
    labels would be fine
57 label_map = label_map_util.load_labelmap(
    PATH_TO_LABELS)
58 categories = label_map_util.
    convert_label_map_to_categories(label_map,
    max_num_classes=NUM_CLASSES, use_display_name=True)
59 category_index = label_map_util.create_category_index
    (categories)
60
```

```

61 # Load the Tensorflow model into memory.
62 detection_graph = tf.Graph()
63 with detection_graph.as_default():
64     od_graph_def = tf.GraphDef()
65     with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
66         serialized_graph = fid.read()
67         od_graph_def.ParseFromString(
            serialized_graph)
68         tf.import_graph_def(od_graph_def, name='')
69
70     sess = tf.Session(graph=detection_graph)
71
72 # Define input and output tensors (i.e. data) for
the object detection classifier
73
74 # Input tensor is the image
75 image_tensor = detection_graph.get_tensor_by_name('
    image_tensor:0')
76
77 # Output tensors are the detection boxes, scores,
and classes
78 # Each box represents a part of the image where a
particular object was detected
79 detection_boxes = detection_graph.get_tensor_by_name(
    ('detection_boxes:0'))
80
81 # Each score represents level of confidence for each
of the objects.
82 # The score is shown on the result image, together
with the class label.
83 detection_scores = detection_graph.
    get_tensor_by_name('detection_scores:0')
84 detection_classes = detection_graph.
    get_tensor_by_name('detection_classes:0')
85
86 # Number of objects detected
87 num_detections = detection_graph.get_tensor_by_name(
    'num_detections:0')
88
89 # Load image using OpenCV and
90 # expand image dimensions to have shape: [1, None,

```

```

90 None, 3]
91 # i.e. a single-column array, where each item in the
    column has the pixel RGB value
92 image = cv2.imread(PATH_TO_IMAGE)
93 image_expanded = np.expand_dims(image, axis=0)
94
95 # Perform the actual detection by running the model
    with the image as input
96 (boxes, scores, classes, num) = sess.run(
97     [detection_boxes, detection_scores,
        detection_classes, num_detections],
98     feed_dict={image_tensor: image_expanded})
99
100 # Draw the results of the detection (aka 'visualize
    the results')
101
102 #vis_util.visualize_boxes_and_labels_on_image_array(
103 #    image,
104 #    np.squeeze(boxes),
105 #    np.squeeze(classes).astype(np.int32),
106 #    np.squeeze(scores),
107 #    category_index,
108 #    use_normalized_coordinates=True,
109 #    line_thickness=8,
110 #    min_score_thresh=0.80)
111
112 vis_util.visualize_text_and_labels_on_image_array(
113     image,
114     np.squeeze(boxes),
115     np.squeeze(classes).astype(np.int32),
116     np.squeeze(scores),
117     category_index,
118     use_normalized_coordinates=True,
119     line_thickness=8,
120     min_score_thresh=0.51)
121
122 #added to display the number of detections (boxes)
    on the console - 07.03.2020
123 #print("Number of Medflies detected =", len(boxes))
124 #
125

```

```
126 height = np.size(image, 0)
127 width = np.size(image, 1)
128 cv2.resizeWindow('Object detector', width, height)
129
130 #cv2.namedWindow('Object detector',
    CV_WINDOW_AUTOSIZE)
131
132 # All the results have been drawn on image. Now
    display the image.
133 cv2.imshow('Object detector', image)
134
135 # Press any key to close the image
136 cv2.waitKey(0)
137
138 # Clean up
139 cv2.destroyAllWindows()
140
```

APPENDIX B. Visualization-utils.py

File - T:\veysif\doktora\aksaray\Tez kitapçığı\ilk teslim sonrası\ekler\visualization_utils\visualization_utils_07.03.py

```
1 # Copyright 2017 The TensorFlow Authors. All Rights
  # Reserved.
2 #
3 # Licensed under the Apache License, Version 2.0 (the
  # "License");
4 # you may not use this file except in compliance with
  # the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in
  # writing, software
10 # distributed under the License is distributed on an
  # "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
  # either express or implied.
12 # See the License for the specific language governing
  # permissions and
13 # limitations under the License.
14 # =====
  # =====
15
16 """A set of functions that are used for visualization
  .
17
18 These functions often receive an image, perform some
  visualization on the image.
19 The functions do not return a value, instead they
  modify the image itself.
20
21 """
22 import abc
23 import collections
24 import functools
25 # Set headless-friendly backend.
26 import matplotlib; matplotlib.use('Agg') # pylint:
  disable=multiple-statements
27 import matplotlib.pyplot as plt # pylint: disable=g-
  import-not-at-top
28 import numpy as np
```

```

29 import PIL.Image as Image
30 import PIL.ImageColor as ImageColor
31 import PIL.ImageDraw as ImageDraw
32 import PIL.ImageFont as ImageFont
33 import six
34 import tensorflow as tf
35
36 from object_detection.core import standard_fields as
    fields
37 from object_detection.utils import shape_utils
38
39 _TITLE_LEFT_MARGIN = 10
40 _TITLE_TOP_MARGIN = 10
41 STANDARD_COLORS = [
42     'AliceBlue', 'Chartreuse', 'Aqua', 'Aquamarine',
43     'Azure', 'Beige', 'Bisque',
44     'BlanchedAlmond', 'BlueViolet', 'BurlyWood', '
    CadetBlue', 'AntiqueWhite',
45     'Chocolate', 'Coral', 'CornflowerBlue', 'Cornsilk
    ', 'Crimson', 'Cyan',
46     'DarkCyan', 'DarkGoldenRod', 'DarkGrey', '
    DarkKhaki', 'DarkOrange',
47     'DarkOrchid', 'DarkSalmon', 'DarkSeaGreen', '
    DarkTurquoise', 'DarkViolet',
48     'DeepPink', 'DeepSkyBlue', 'DodgerBlue', '
    FireBrick', 'FloralWhite',
49     'ForestGreen', 'Fuchsia', 'Gainsboro', '
    GhostWhite', 'Gold', 'GoldenRod',
50     'Salmon', 'Tan', 'HoneyDew', 'HotPink', '
    IndianRed', 'Ivory', 'Khaki',
51     'Lavender', 'LavenderBlush', 'LawnGreen', '
    LemonChiffon', 'LightBlue',
52     'LightCoral', 'LightCyan', 'LightGoldenRodYellow'
    , 'LightGray', 'LightGrey',
53     'LightGreen', 'LightPink', 'LightSalmon', '
    LightSeaGreen', 'LightSkyBlue',
54     'LightSlateGray', 'LightSlateGrey', '
    LightSteelBlue', 'LightYellow', 'Lime',
55     'LimeGreen', 'Linen', 'Magenta', '
    MediumAquaMarine', 'MediumOrchid',
56     'MediumPurple', 'MediumSeaGreen', '

```

```

55 MediumSlateBlue', 'MediumSpringGreen',
56     'MediumTurquoise', 'MediumVioletRed', 'MintCream'
    , 'MistyRose', 'Moccasin',
57     'NavajoWhite', 'OldLace', 'Olive', 'OliveDrab', '
Orange', 'OrangeRed',
58     'Orchid', 'PaleGoldenRod', 'PaleGreen', '
PaleTurquoise', 'PaleVioletRed',
59     'PapayaWhip', 'PeachPuff', 'Peru', 'Pink', 'Plum'
    , 'PowderBlue', 'Purple',
60     'Red', 'RosyBrown', 'RoyalBlue', 'SaddleBrown', '
Green', 'SandyBrown',
61     'SeaGreen', 'SeaShell', 'Sienna', 'Silver', '
SkyBlue', 'SlateBlue',
62     'SlateGray', 'SlateGrey', 'Snow', 'SpringGreen',
    'SteelBlue', 'GreenYellow',
63     'Teal', 'Thistle', 'Tomato', 'Turquoise', 'Violet'
    , 'Wheat', 'White',
64     'WhiteSmoke', 'Yellow', 'YellowGreen'
65 ]
66
67
68 def save_image_array_as_png(image, output_path):
69     """Saves an image (represented as a numpy array) to
    PNG.
70
71     Args:
72         image: a numpy array with shape [height, width, 3
    ].
73         output_path: path to which image should be
    written.
74     """
75     image_pil = Image.fromarray(np.uint8(image)).
    convert('RGB')
76     with tf.gfile.Open(output_path, 'w') as fid:
77         image_pil.save(fid, 'PNG')
78
79
80 def encode_image_array_as_png_str(image):
81     """Encodes a numpy array into a PNG string.
82
83     Args:

```

```

84     image: a numpy array with shape [height, width,
      3].
85
86     Returns:
87     PNG encoded image string.
88     """
89     image_pil = Image.fromarray(np.uint8(image))
90     output = six.BytesIO()
91     image_pil.save(output, format='PNG')
92     png_string = output.getvalue()
93     output.close()
94     return png_string
95
96
97 def draw_bounding_box_on_image_array(image,
98                                     ymin,
99                                     xmin,
100                                    ymax,
101                                    xmax,
102                                    color='red',
103                                    thickness=4,
104
105     display_str_list=(),
106     use_normalized_coordinates=True):
107     """Adds a bounding box to an image (numpy array).
108     Bounding box coordinates can be specified in
109     either absolute (pixel) or
110     normalized coordinates by setting the
111     use_normalized_coordinates argument.
112     Args:
113     image: a numpy array with shape [height, width,
114     3].
115     ymin: ymin of bounding box.
116     xmin: xmin of bounding box.
117     ymax: ymax of bounding box.
118     xmax: xmax of bounding box.
119     color: color to draw bounding box. Default is
120     red.

```

```

118     thickness: line thickness. Default value is 4.
119     display_str_list: list of strings to display in
    box
120                     (each to be shown on its own
    line).
121     use_normalized_coordinates: If True (default),
    treat coordinates
122     ymin, xmin, ymax, xmax as relative to the
    image. Otherwise treat
123     coordinates as absolute.
124     """
125     image_pil = Image.fromarray(np.uint8(image)).
    convert('RGB')
126     draw_bounding_box_on_image(image_pil, ymin, xmin,
    ymax, xmax, color,
127                               thickness,
    display_str_list,
128                               use_normalized_coordinates)
129     np.copyto(image, np.array(image_pil))
130
131 def draw_text_on_image_array(image,
132                               ymin,
133                               xmin,
134                               ymax,
135                               xmax,
136                               color='red',
137                               thickness=4,
138
139                               display_str_list=(),
140                               use_normalized_coordinates=True):
141     """Adds a bounding box to an image (numpy array).
142     Bounding box coordinates can be specified in
    either absolute (pixel) or
143     normalized coordinates by setting the
    use_normalized_coordinates argument.
144
145     Args:
146     image: a numpy array with shape [height, width,

```

```

146 3].
147     ymin: ymin of bounding box.
148     xmin: xmin of bounding box.
149     ymax: ymax of bounding box.
150     xmax: xmax of bounding box.
151     color: color to draw bounding box. Default is
        red.
152     thickness: line thickness. Default value is 4.
153     display_str_list: list of strings to display in
        box
154                        (each to be shown on its own
        line).
155     use_normalized_coordinates: If True (default),
        treat coordinates
156     ymin, xmin, ymax, xmax as relative to the
        image. Otherwise treat
157     coordinates as absolute.
158     """
159     image_pil = Image.fromarray(np.uint8(image)).
        convert('RGB')
160     draw_text_on_image(image_pil, ymin, xmin, ymax,
        xmax, color,
161                        thickness,
        display_str_list,
162                        use_normalized_coordinates)
163     np.copyto(image, np.array(image_pil))
164
165 def draw_bounding_box_on_image(image,
166                               ymin,
167                               xmin,
168                               ymax,
169                               xmax,
170                               color='red',
171                               thickness=4,
172                               display_str_list=(),
173                               use_normalized_coordinates=True):
174     """Adds a bounding box to an image.
175
176     Bounding box coordinates can be specified in

```

```

176 either absolute (pixel) or
177     normalized coordinates by setting the
178     use_normalized_coordinates argument.
179
180 Each string in display_str_list is displayed on a
181 separate line above the
182 bounding box in black text on a rectangle filled
183 with the input 'color'.
184 If the top of the bounding box extends to the edge
185 of the image, the strings
186 are displayed below the bounding box.
187
188 Args:
189     image: a PIL.Image object.
190     ymin: ymin of bounding box.
191     xmin: xmin of bounding box.
192     ymax: ymax of bounding box.
193     xmax: xmax of bounding box.
194     color: color to draw bounding box. Default is
195     red.
196     thickness: line thickness. Default value is 4.
197     display_str_list: list of strings to display in
198     box
199                             (each to be shown on its own
200                             line).
201     use_normalized_coordinates: If True (default),
202     treat coordinates
203     ymin, xmin, ymax, xmax as relative to the
204     image. Otherwise treat
205     coordinates as absolute.
206 """
207 draw = ImageDraw.Draw(image)
208 im_width, im_height = image.size
209 if use_normalized_coordinates:
210     (left, right, top, bottom) = (xmin * im_width,
211     xmax * im_width,
212                                     ymin * im_height,
213     ymax * im_height)
214 else:
215     (left, right, top, bottom) = (xmin, xmax, ymin,
216     ymax)

```

```

205     draw.line([(left, top), (left, bottom), (right,
206               bottom),
207               (right, top), (left, top)], width=
208               thickness, fill=color)
209     try:
210         font = ImageFont.truetype('arial.ttf', 24)
211     except IOError:
212         font = ImageFont.load_default()
213
214     # If the total height of the display strings added
215     to the top of the bounding
216     # box exceeds the top of the image, stack the
217     strings below the bounding box
218     # instead of above.
219     display_str_heights = [font.getsize(ds)[1] for ds
220     in display_str_list]
221     # Each display_str has a top and bottom margin of
222     0.05x.
223     total_display_str_height = (1 + 2 * 0.05) * sum(
224     display_str_heights)
225
226     if top > total_display_str_height:
227         text_bottom = top
228     else:
229         text_bottom = bottom + total_display_str_height
230     # Reverse list and print from bottom to top.
231     for display_str in display_str_list[::-1]:
232         text_width, text_height = font.getsize(
233         display_str)
234         margin = np.ceil(0.05 * text_height)
235         draw.rectangle(
236             [(left, text_bottom - text_height - 2 *
237             margin), (left + text_width,
238             text_bottom)],
239             fill=color)
240         draw.text(
241             (left + margin, text_bottom - text_height -
242             margin),
243             display_str,
244             fill='black',

```

```

235         font=font)
236     text_bottom -= text_height - 2 * margin
237
238 def draw_text_on_image(image,
239                         ymin,
240                         xmin,
241                         ymax,
242                         xmax,
243                         color='red',
244                         thickness=4,
245                         display_str_list=(),
246                         use_normalized_coordinates=True):
247     """Adds a text to an image.
248
249     xxx Bounding box coordinates can be specified in
250     either absolute (pixel) or
251     xxx normalized coordinates by setting the
252     use_normalized_coordinates argument.
253
254     xxx Each string in display_str_list is displayed
255     on a separate line above the
256     xxx bounding box in black text on a rectangle
257     filled with the input 'color'.
258
259     xxx If the top of the bounding box extends to the
260     edge of the image, the strings
261     xxx are displayed below the bounding box.
262
263     Args:
264         image: a PIL.Image object.
265         ymin: ymin of bounding box.
266         xmin: xmin of bounding box.
267         ymax: ymax of bounding box.
268         xmax: xmax of bounding box.
269         color: color to draw bounding box. Default is
270         red.
271         thickness: line thickness. Default value is 4.
272         display_str_list: list of strings to display in
273         box
274
275         (each to be shown on its own
276         line).

```

```

267     use_normalized_coordinates: If True (default),
        treat coordinates
268     ymin, xmin, ymax, xmax as relative to the
        image. Otherwise treat
269     coordinates as absolute.
270     """
271     draw = ImageDraw.Draw(image)
272     im_width, im_height = image.size
273     if use_normalized_coordinates:
274         (left, right, top, bottom) = (xmin * im_width,
        xmax * im_width,
275                                     ymin * im_height,
        ymax * im_height)
276     else:
277         (left, right, top, bottom) = (xmin, xmax, ymin,
        ymax)
278     #draw.line([(left, top), (left, bottom), (right,
        bottom),
279               (right, top), (left, top)], width=
        thickness, fill=color)
280     draw.text(
281         ((left + right)/2, (top + bottom)/2),
282         '*',
283         fill='black',
284         font=font)
285
286     try:
287         font = ImageFont.truetype('arial.ttf', 24)
288     except IOError:
289         font = ImageFont.load_default()
290
291     # If the total height of the display strings added
        to the top of the bounding
292     # box exceeds the top of the image, stack the
        strings below the bounding box
293     # instead of above.
294     display_str_heights = [font.getsize(ds)[1] for ds
        in display_str_list]
295     # Each display_str has a top and bottom margin of
        0.05x.
296     total_display_str_height = (1 + 2 * 0.05) * sum(

```

```

296 display_str_heights)
297
298     if top > total_display_str_height:
299         text_bottom = top
300     else:
301         text_bottom = bottom + total_display_str_height
302         # Reverse list and print from bottom to top.
303         for display_str in display_str_list[::-1]:
304             text_width, text_height = font.getsize(
305                 display_str)
306             margin = np.ceil(0.05 * text_height)
307             draw.rectangle(
308                 [(left, text_bottom - text_height - 2 *
309                    margin), (left + text_width,
310                               text_bottom)],
311                 fill=color)
312             draw.text(
313                 (left + margin, text_bottom - text_height -
314                  margin),
315                 display_str,
316                 fill='black',
317                 font=font)
318             text_bottom -= text_height - 2 * margin
319
320 def draw_bounding_boxes_on_image_array(image,
321                                       boxes,
322                                       color='red',
323                                       thickness=4,
324                                       display_str_list_list=()):
325     """Draws bounding boxes on image (numpy array).
326
327     Args:
328     image: a numpy array object.
329     boxes: a 2 dimensional numpy array of [N, 4]: (
330     ymin, xmin, ymax, xmax).
331     The coordinates are in normalized format
332     between [0, 1].
333     color: color to draw bounding box. Default is
334     red.

```

```

329     thickness: line thickness. Default value is 4.
330     display_str_list_list: list of list of strings.
331         a list of strings for
        each bounding box.
332         The reason to pass a list
        of strings for a
333         bounding box is that it
        might contain
334         multiple labels.
335
336     Raises:
337         ValueError: if boxes is not a [N, 4] array
338     """
339     image_pil = Image.fromarray(image)
340     draw_bounding_boxes_on_image(image_pil, boxes,
        color, thickness,
341         display_str_list_list
        )
342     np.copyto(image, np.array(image_pil))
343
344
345 def draw_bounding_boxes_on_image(image,
346     boxes,
347     color='red',
348     thickness=4,
349
        display_str_list_list=()):
350     """Draws bounding boxes on image.
351
352     Args:
353         image: a PIL.Image object.
354         boxes: a 2 dimensional numpy array of [N, 4]: (
            ymin, xmin, ymax, xmax).
355             The coordinates are in normalized format
            between [0, 1].
356         color: color to draw bounding box. Default is
            red.
357         thickness: line thickness. Default value is 4.
358         display_str_list_list: list of list of strings.
359             a list of strings for
            each bounding box.

```

```

360                                     The reason to pass a list
      of strings for a
361                                     bounding box is that it
      might contain
362                                     multiple labels.
363
364     Raises:
365         ValueError: if boxes is not a [N, 4] array
366     """
367     boxes_shape = boxes.shape
368     if not boxes_shape:
369         return
370     if len(boxes_shape) != 2 or boxes_shape[1] != 4:
371         raise ValueError('Input must be of size [N, 4]')
372     for i in range(boxes_shape[0]):
373         display_str_list = ()
374         if display_str_list_list:
375             display_str_list = display_str_list_list[i]
376         draw_bounding_box_on_image(image, boxes[i, 0],
377                                   boxes[i, 1], boxes[i, 2],
378                                   boxes[i, 3], color,
379                                   thickness, display_str_list)
380
381
382 def _visualize_boxes(image, boxes, classes, scores,
383                     category_index, **kwargs):
384     return visualize_boxes_and_labels_on_image_array(
385         image, boxes, classes, scores, category_index=
386         category_index, **kwargs)
387
388
389 def _visualize_boxes_and_masks(image, boxes, classes
390                               , scores, masks,
391                               category_index, **
392                               kwargs):
393     return visualize_boxes_and_labels_on_image_array(
394         image,
395         boxes,
396         classes,
397         scores,
398         category_index=category_index,

```

```

393         instance_masks=masks,
394         **kwargs)
395
396
397 def _visualize_boxes_and_keypoints(image, boxes,
398     classes, scores, keypoints,
399     category_index
400     , **kwargs):
401     return visualize_boxes_and_labels_on_image_array(
402         image,
403         boxes,
404         classes,
405         scores,
406         category_index=category_index,
407         keypoints=keypoints,
408         **kwargs)
409
410 def _visualize_boxes_and_masks_and_keypoints(
411     image, boxes, classes, scores, masks, keypoints
412     , category_index, **kwargs):
413     return visualize_boxes_and_labels_on_image_array(
414         image,
415         boxes,
416         classes,
417         scores,
418         category_index=category_index,
419         instance_masks=masks,
420         keypoints=keypoints,
421         **kwargs)
422
423 def _resize_original_image(image, image_shape):
424     image = tf.expand_dims(image, 0)
425     image = tf.image.resize_images(
426         image,
427         image_shape,
428         method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
429         align_corners=True)
430     return tf.cast(tf.squeeze(image, 0), tf.uint8)

```

```

431
432 def draw_bounding_boxes_on_image_tensors(images,
433                                           boxes,
434                                           classes,
435                                           scores,
436
437     category_index,
438     original_image_spatial_shape=None,
439     true_image_shape=None,
440     instance_masks=None,
441                                           keypoints=
442     None,
443     max_boxes_to_draw=20,
444     min_score_thresh=0.2,
445     use_normalized_coordinates=True):
446     """Draws bounding boxes, masks, and keypoints on
447     batch of image tensors.
448
449     Args:
450         images: A 4D uint8 image tensor of shape [N, H,
451         W, C]. If C > 3, additional
452         channels will be ignored. If C = 1, then we
453         convert the images to RGB
454         images.
455         boxes: [N, max_detections, 4] float32 tensor of
456         detection boxes.
457         classes: [N, max_detections] int tensor of
458         detection classes. Note that
459         classes are 1-indexed.
460         scores: [N, max_detections] float32 tensor of
461         detection scores.
462         category_index: a dict that maps integer ids to
463         category dicts. e.g.
464         {1: {1: 'dog'}, 2: {2: 'cat'}, ...}
465         original_image_spatial_shape: [N, 2] tensor

```

```

456 containing the spatial size of
457     the original image.
458     true_image_shape: [N, 3] tensor containing the
        spatial size of unpadded
459     original_image.
460     instance_masks: A 4D uint8 tensor of shape [N,
        max_detection, H, W] with
461     instance masks.
462     keypoints: A 4D float32 tensor of shape [N,
        max_detection, num_keypoints, 2]
463     with keypoints.
464     max_boxes_to_draw: Maximum number of boxes to
        draw on an image. Default 20.
465     min_score_thresh: Minimum score threshold for
        visualization. Default 0.2.
466     use_normalized_coordinates: Whether to assume
        boxes and keypoints are in
467     normalized coordinates (as opposed to absolute
        coordinates).
468     Default is True.
469
470 Returns:
471     4D image tensor of type uint8, with boxes drawn
        on top.
472     """
473     # Additional channels are being ignored.
474     if images.shape[3] > 3:
475         images = images[:, :, :, 0:3]
476     elif images.shape[3] == 1:
477         images = tf.image.grayscale_to_rgb(images)
478     visualization_keyword_args = {
479         'use_normalized_coordinates':
            use_normalized_coordinates,
480         'max_boxes_to_draw': max_boxes_to_draw,
481         'min_score_thresh': min_score_thresh,
482         'agnostic_mode': False,
483         'line_thickness': 4
484     }
485     if true_image_shape is None:
486         true_shapes = tf.constant(-1, shape=[images.
            shape.as_list()[0], 3])

```

```

487     else:
488         true_shapes = true_image_shape
489         if original_image_spatial_shape is None:
490             original_shapes = tf.constant(-1, shape=[images.
shape.as_list()[0], 2])
491         else:
492             original_shapes = original_image_spatial_shape
493
494         if instance_masks is not None and keypoints is
None:
495             visualize_boxes_fn = functools.partial(
496                 _visualize_boxes_and_masks,
497                 category_index=category_index,
498                 **visualization_keyword_args)
499             elems = [
500                 true_shapes, original_shapes, images, boxes
, classes, scores,
501                 instance_masks
502             ]
503         elif instance_masks is None and keypoints is not
None:
504             visualize_boxes_fn = functools.partial(
505                 _visualize_boxes_and_keypoints,
506                 category_index=category_index,
507                 **visualization_keyword_args)
508             elems = [
509                 true_shapes, original_shapes, images, boxes
, classes, scores, keypoints
510             ]
511         elif instance_masks is not None and keypoints is
not None:
512             visualize_boxes_fn = functools.partial(
513                 _visualize_boxes_and_masks_and_keypoints,
514                 category_index=category_index,
515                 **visualization_keyword_args)
516             elems = [
517                 true_shapes, original_shapes, images, boxes
, classes, scores,
518                 instance_masks, keypoints
519             ]
520     else:

```

```

521     visualize_boxes_fn = functools.partial(
522         _visualize_boxes,
523         category_index=category_index,
524         **visualization_keyword_args)
525     elems = [
526         true_shapes, original_shapes, images, boxes
527     ], classes, scores
528
529     def draw_boxes(image_and_detections):
530         """Draws boxes on image."""
531         true_shape = image_and_detections[0]
532         original_shape = image_and_detections[1]
533         if true_image_shape is not None:
534             image = shape_utils.pad_or_clip_nd(
535                 image_and_detections[2],
536                 [true_shape
537                  [0], true_shape[1], 3])
538         if original_image_spatial_shape is not None:
539             image_and_detections[2] =
540             _resize_original_image(image, original_shape)
541         image_with_boxes = tf.py_func(visualize_boxes_fn
542             , image_and_detections[2:],
543             tf.uint8)
544         return image_with_boxes
545
546     images = tf.map_fn(draw_boxes, elems, dtype=tf.
547         uint8, back_prop=False)
548     return images
549
550     def draw_side_by_side_evaluation_image(eval_dict,
551         category_index,
552         max_boxes_to_draw=20,
553         min_score_thresh=0.2,
554         use_normalized_coordinates=True):

```

```

552     """Creates a side-by-side image with detections
553 and groundtruth.
554     Bounding boxes (and instance masks, if available)
555 are visualized on both
556 subimages.
557     Args:
558         eval_dict: The evaluation dictionary returned by
559             eval_util.result_dict_for_batched_example() or
560             eval_util.result_dict_for_single_example().
561         category_index: A category index (dictionary)
562             produced from a labelmap.
563         max_boxes_to_draw: The maximum number of boxes
564             to draw for detections.
565         min_score_thresh: The minimum score threshold
566             for showing detections.
567         use_normalized_coordinates: Whether to assume
568             boxes and keypoints are in
569             normalized coordinates (as opposed to absolute
570             coordiantes).
571             Default is True.
572     Returns:
573         A list of [1, H, 2 * W, C] uint8 tensor. The
574         subimage on the left
575         corresponds to detections, while the subimage
576         on the right corresponds to
577         groundtruth.
578     """
579     detection_fields = fields.DetectionResultFields()
580     input_data_fields = fields.InputDataFields()
581     images_with_detections_list = []
582     # Add the batch dimension if the eval_dict is for
583     # single example.
584     if len(eval_dict[detection_fields.
585         detection_classes].shape) == 1:
586         for key in eval_dict:
587             if key != input_data_fields.original_image:

```

```

582         eval_dict[key] = tf.expand_dims(eval_dict[
    key], 0)
583
584     for indx in range(eval_dict[input_data_fields.
    original_image].shape[0]):
585         instance_masks = None
586         if detection_fields.detection_masks in eval_dict
    :
587             instance_masks = tf.cast(
588                 tf.expand_dims(
589                     eval_dict[detection_fields.
    detection_masks][indx], axis=0),
590                 tf.uint8)
591         keypoints = None
592         if detection_fields.detection_keypoints in
    eval_dict:
593             keypoints = tf.expand_dims(
594                 eval_dict[detection_fields.
    detection_keypoints][indx], axis=0)
595         groundtruth_instance_masks = None
596         if input_data_fields.groundtruth_instance_masks
    in eval_dict:
597             groundtruth_instance_masks = tf.cast(
598                 tf.expand_dims(
599                     eval_dict[input_data_fields.
    groundtruth_instance_masks][indx],
600                     axis=0), tf.uint8)
601
602     images_with_detections =
    draw_bounding_boxes_on_image_tensors(
603         tf.expand_dims(
604             eval_dict[input_data_fields.
    original_image][indx], axis=0),
605         tf.expand_dims(
606             eval_dict[detection_fields.
    detection_boxes][indx], axis=0),
607         tf.expand_dims(
608             eval_dict[detection_fields.
    detection_classes][indx], axis=0),
609         tf.expand_dims(
610             eval_dict[detection_fields.

```

```

610 detection_scores][indx], axis=0),
611         category_index,
612         original_image_spatial_shape=tf.expand_dims(
613             eval_dict[input_data_fields.
        original_image_spatial_shape][indx],
614             axis=0),
615         true_image_shape=tf.expand_dims(
616             eval_dict[input_data_fields.
        true_image_shape][indx], axis=0),
617         instance_masks=instance_masks,
618         keypoints=keypoints,
619         max_boxes_to_draw=max_boxes_to_draw,
620         min_score_thresh=min_score_thresh,
621         use_normalized_coordinates=
        use_normalized_coordinates)
622     images_with_groundtruth =
        draw_bounding_boxes_on_image_tensors(
623         tf.expand_dims(
624             eval_dict[input_data_fields.
        original_image][indx], axis=0),
625         tf.expand_dims(
626             eval_dict[input_data_fields.
        groundtruth_boxes][indx], axis=0),
627         tf.expand_dims(
628             eval_dict[input_data_fields.
        groundtruth_classes][indx], axis=0),
629         tf.expand_dims(
630             tf.ones_like(
631                 eval_dict[input_data_fields.
        groundtruth_classes][indx],
632                 dtype=tf.float32),
633             axis=0),
634         category_index,
635         original_image_spatial_shape=tf.expand_dims(
636             eval_dict[input_data_fields.
        original_image_spatial_shape][indx],
637             axis=0),
638         true_image_shape=tf.expand_dims(
639             eval_dict[input_data_fields.
        true_image_shape][indx], axis=0),
640         instance_masks=groundtruth_instance_masks,

```

```

641         keypoints=None,
642         max_boxes_to_draw=None,
643         min_score_thresh=0.0,
644         use_normalized_coordinates=
        use_normalized_coordinates)
645     images_with_detections_list.append(
646         tf.concat([images_with_detections,
        images_with_groundtruth], axis=2))
647     return images_with_detections_list
648
649
650 def draw_keypoints_on_image_array(image,
651                                   keypoints,
652                                   color='red',
653                                   radius=2,
654                                   use_normalized_coordinates=True):
655     """Draws keypoints on an image (numpy array).
656
657     Args:
658         image: a numpy array with shape [height, width,
659         3].
660         keypoints: a numpy array with shape [
661         num_keypoints, 2].
662         color: color to draw the keypoints with. Default
663         is red.
664         radius: keypoint radius. Default value is 2.
665         use_normalized_coordinates: if True (default),
666         treat keypoint values as
667         relative to the image. Otherwise treat them
668         as absolute.
669     """
670     image_pil = Image.fromarray(np.uint8(image)).
        convert('RGB')
671     draw_keypoints_on_image(image_pil, keypoints,
        color, radius,
672                             use_normalized_coordinates
        )
673     np.copyto(image, np.array(image_pil))
674
675

```

```

671 def draw_keypoints_on_image(image,
672                             keypoints,
673                             color='red',
674                             radius=2,
675
676                             use_normalized_coordinates=True):
677     """Draws keypoints on an image.
678     Args:
679     image: a PIL.Image object.
680     keypoints: a numpy array with shape [
681     num_keypoints, 2].
682     color: color to draw the keypoints with. Default
683     is red.
684     radius: keypoint radius. Default value is 2.
685     use_normalized_coordinates: if True (default),
686     treat keypoint values as
687     relative to the image. Otherwise treat them
688     as absolute.
689     """
690     draw = ImageDraw.Draw(image)
691     im_width, im_height = image.size
692     keypoints_x = [k[1] for k in keypoints]
693     keypoints_y = [k[0] for k in keypoints]
694     if use_normalized_coordinates:
695         keypoints_x = tuple([im_width * x for x in
696                             keypoints_x])
697         keypoints_y = tuple([im_height * y for y in
698                             keypoints_y])
699     for keypoint_x, keypoint_y in zip(keypoints_x,
700                                     keypoints_y):
701         draw.ellipse([(keypoint_x - radius, keypoint_y
702                         - radius),
703                     (keypoint_x + radius, keypoint_y
704                     + radius)],
705                     outline=color, fill=color)
706
707 def draw_mask_on_image_array(image, mask, color='red',
708                             alpha=0.4):
709     """Draws mask on an image.

```

```

701
702     Args:
703         image: uint8 numpy array with shape (img_height
704             , img_height, 3)
705         mask: a uint8 numpy array of shape (img_height,
706             img_height) with
707             values between either 0 or 1.
708         color: color to draw the keypoints with. Default
709             is red.
710         alpha: transparency value between 0 and 1. (
711             default: 0.4)
712
713     Raises:
714         ValueError: On incorrect data type for image or
715             masks.
716     """
717     if image.dtype != np.uint8:
718         raise ValueError('`image` not of type np.uint8')
719     if mask.dtype != np.uint8:
720         raise ValueError('`mask` not of type np.uint8')
721     if np.any(np.logical_and(mask != 1, mask != 0)):
722         raise ValueError('`mask` elements should be in [
723             0, 1]')
724     if image.shape[:2] != mask.shape:
725         raise ValueError('The image has spatial
726             dimensions %s but the mask has '
727             'dimensions %s' % (image.shape
728             [:2], mask.shape))
729     rgb = ImageColor.getrgb(color)
730     pil_image = Image.fromarray(image)
731
732     solid_color = np.expand_dims(
733         np.ones_like(mask), axis=2) * np.reshape(list(
734             rgb), [1, 1, 3])
735     pil_solid_color = Image.fromarray(np.uint8(
736         solid_color)).convert('RGBA')
737     pil_mask = Image.fromarray(np.uint8(255.0*alpha*
738         mask)).convert('L')
739     pil_image = Image.composite(pil_solid_color,
740         pil_image, pil_mask)
741     np.copyto(image, np.array(pil_image.convert('RGB'

```

```

729 )))
730
731
732 def visualize_boxes_and_labels_on_image_array(
733     image,
734     boxes,
735     classes,
736     scores,
737     category_index,
738     instance_masks=None,
739     instance_boundaries=None,
740     keypoints=None,
741     use_normalized_coordinates=False,
742     max_boxes_to_draw=20,
743     min_score_thresh=.5,
744     agnostic_mode=False,
745     line_thickness=4,
746     groundtruth_box_visualization_color='black',
747     skip_scores=False,
748     skip_labels=False):
749     """Overlay labeled boxes on an image with
750         formatted scores and label names.
751
752         This function groups boxes that correspond to the
753         same location
754         and creates a display string for each detection
755         and overlays these
756         on the image. Note that this function modifies the
757         image in place, and returns
758         that same image.
759
760         Args:
761             image: uint8 numpy array with shape (img_height
762                 , img_width, 3)
763             boxes: a numpy array of shape [N, 4]
764             classes: a numpy array of shape [N]. Note that
765                 class indices are 1-based,
766                 and match the keys in the label map.
767             scores: a numpy array of shape [N] or None. If
768                 scores=None, then
769                 this function assumes that the boxes to be


```

```

762 plotted are groundtruth
763     boxes and plot all boxes as black with no
       classes or scores.
764     category_index: a dict containing category
       dictionaries (each holding
765         category index `id` and category name `name`)
       keyed by category indices.
766     instance_masks: a numpy array of shape [N,
       image_height, image_width] with
767         values ranging between 0 and 1, can be None.
768     instance_boundaries: a numpy array of shape [N,
       image_height, image_width]
769         with values ranging between 0 and 1, can be
       None.
770     keypoints: a numpy array of shape [N,
       num_keypoints, 2], can
771         be None
772     use_normalized_coordinates: whether boxes is to
       be interpreted as
773         normalized coordinates or not.
774     max_boxes_to_draw: maximum number of boxes to
       visualize. If None, draw
775         all boxes.
776     min_score_thresh: minimum score threshold for a
       box to be visualized
777     agnostic_mode: boolean (default: False)
       controlling whether to evaluate in
778         class-agnostic mode or not. This mode will
       display scores but ignore
779         classes.
780     line_thickness: integer (default: 4) controlling
       line width of the boxes.
781     groundtruth_box_visualization_color: box color
       for visualizing groundtruth
782         boxes
783     skip_scores: whether to skip score when drawing
       a single detection
784     skip_labels: whether to skip label when drawing
       a single detection
785
786 Returns:

```

```

787     uint8 numpy array with shape (img_height,
788     img_width, 3) with overlaid boxes.
789     """
790     # Create a display string (and color) for every
791     box location, group any boxes
792     # that correspond to the same location.
793     box_to_display_str_map = collections.defaultdict(
794         list)
795     box_to_color_map = collections.defaultdict(str)
796     box_to_instance_masks_map = {}
797     box_to_instance_boundaries_map = {}
798     box_to_keypoints_map = collections.defaultdict(
799         list)
800     if not max_boxes_to_draw:
801         max_boxes_to_draw = boxes.shape[0]
802     for i in range(min(max_boxes_to_draw, boxes.shape[
803         0])):
804         if scores is None or scores[i] >
805         min_score_thresh:
806             box = tuple(boxes[i].tolist())
807             if instance_masks is not None:
808                 box_to_instance_masks_map[box] =
809                 instance_masks[i]
810             if instance_boundaries is not None:
811                 box_to_instance_boundaries_map[box] =
812                 instance_boundaries[i]
813             if keypoints is not None:
814                 box_to_keypoints_map[box].extend(keypoints[i
815                 ])
816             if scores is None:
817                 box_to_color_map[box] =
818                 groundtruth_box_visualization_color
819             else:
820                 display_str = ''
821                 if not skip_labels:
822                     if not agnostic_mode:
823                         if classes[i] in category_index.keys():
824                             class_name = category_index[classes[i
825                             ]]['name']
826                     else:
827                         class_name = 'N/A'

```

```

817         display_str = str(class_name)
818         if not skip_scores:
819             if not display_str:
820                 display_str = '{}%'.format(int(100*
scores[i]))
821             else:
822                 display_str = '{}: {}'.format(
display_str, int(100*scores[i]))
823                 box_to_display_str_map[box].append(
display_str)
824             if agnostic_mode:
825                 box_to_color_map[box] = 'DarkOrange'
826             else:
827                 box_to_color_map[box] = STANDARD_COLORS[
828                     classes[i] % len(STANDARD_COLORS)]
829
830     # Draw all boxes onto image.
831     for box, color in box_to_color_map.items():
832         ymin, xmin, ymax, xmax = box
833         if instance_masks is not None:
834             draw_mask_on_image_array(
835                 image,
836                 box_to_instance_masks_map[box],
837                 color=color
838             )
839         if instance_boundaries is not None:
840             draw_mask_on_image_array(
841                 image,
842                 box_to_instance_boundaries_map[box],
843                 color='red',
844                 alpha=1.0
845             )
846         draw_bounding_box_on_image_array(
847             image,
848             ymin,
849             xmin,
850             ymax,
851             xmax,
852             color=color,
853             thickness=line_thickness,
854             display_str_list=box_to_display_str_map[box]

```

```

854 ],
855         use_normalized_coordinates=
use_normalized_coordinates)
856     if keypoints is not None:
857         draw_keypoints_on_image_array(
858             image,
859             box_to_keypoints_map[box],
860             color=color,
861             radius=line_thickness / 2,
862             use_normalized_coordinates=
use_normalized_coordinates)
863
864     return image
865
866 def visualize_text_and_labels_on_image_array(
867     image,
868     boxes,
869     classes,
870     scores,
871     category_index,
872     instance_masks=None,
873     instance_boundaries=None,
874     keypoints=None,
875     use_normalized_coordinates=False,
876     max_boxes_to_draw=20,
877     min_score_thresh=.5,
878     agnostic_mode=False,
879     line_thickness=4,
880     groundtruth_box_visualization_color='black',
881     skip_scores=False,
882     skip_labels=False):
883     """Overlay labeled boxes on an image with
formatted scores and label names.
884
885     This function groups boxes that correspond to the
same location
886     and creates a display string for each detection
and overlays these
887     on the image. Note that this function modifies the
image in place, and returns
888     that same image.

```

```

889
890     Args:
891         image: uint8 numpy array with shape (img_height
            , img_width, 3)
892         boxes: a numpy array of shape [N, 4]
893         classes: a numpy array of shape [N]. Note that
            class indices are 1-based,
894             and match the keys in the label map.
895         scores: a numpy array of shape [N] or None. If
            scores=None, then
896             this function assumes that the boxes to be
            plotted are groundtruth
897             boxes and plot all boxes as black with no
            classes or scores.
898         category_index: a dict containing category
            dictionaries (each holding
899             category index `id` and category name `name`)
            keyed by category indices.
900         instance_masks: a numpy array of shape [N,
            image_height, image_width] with
901             values ranging between 0 and 1, can be None.
902         instance_boundaries: a numpy array of shape [N,
            image_height, image_width]
903             with values ranging between 0 and 1, can be
            None.
904         keypoints: a numpy array of shape [N,
            num_keypoints, 2], can
905             be None
906         use_normalized_coordinates: whether boxes is to
            be interpreted as
907             normalized coordinates or not.
908         max_boxes_to_draw: maximum number of boxes to
            visualize. If None, draw
909             all boxes.
910         min_score_thresh: minimum score threshold for a
            box to be visualized
911         agnostic_mode: boolean (default: False)
            controlling whether to evaluate in
912             class-agnostic mode or not. This mode will
            display scores but ignore
913             classes.

```

```

914     line_thickness: integer (default: 4) controlling
        line width of the boxes.
915     groundtruth_box_visualization_color: box color
        for visualizing groundtruth
916     boxes
917     skip_scores: whether to skip score when drawing
        a single detection
918     skip_labels: whether to skip label when drawing
        a single detection
919
920     Returns:
921         uint8 numpy array with shape (img_height,
        img_width, 3) with overlaid boxes.
922     """
923     # Create a display string (and color) for every
        box location, group any boxes
924     # that correspond to the same location.
925
926     # variable is added to count the number of boxes
        drawn on the image - 07.03.2020
927     num_of_detection = 0
928     #
929     box_to_display_str_map = collections.defaultdict(
        list)
930     box_to_color_map = collections.defaultdict(str)
931     box_to_instance_masks_map = {}
932     box_to_instance_boundaries_map = {}
933     box_to_keypoints_map = collections.defaultdict(
        list)
934     if not max_boxes_to_draw:
935         max_boxes_to_draw = boxes.shape[0]
936     for i in range(min(max_boxes_to_draw, boxes.shape[
        0])):
937         if scores is None or scores[i] >
            min_score_thresh:
938             box = tuple(boxes[i].tolist())
939
940             num_of_detection += 1
941             #
942             if instance_masks is not None:
943                 box_to_instance_masks_map[box] =

```

```

943 instance_masks[i]
944     if instance_boundaries is not None:
945         box_to_instance_boundaries_map[box] =
instance_boundaries[i]
946     if keypoints is not None:
947         box_to_keypoints_map[box].extend(keypoints[i
])
948     if scores is None:
949         box_to_color_map[box] =
groundtruth_box_visualization_color
950     else:
951         display_str = ''
952         if not skip_labels:
953             if not agnostic_mode:
954                 if classes[i] in category_index.keys():
955                     class_name = category_index[classes[i
]][['name']]
956             else:
957                 class_name = 'N/A'
958             display_str = str(class_name)
959         if not skip_scores:
960             if not display_str:
961                 display_str = '{}%'.format(int(100*
scores[i]))
962             else:
963                 display_str = '{}: {}'.format(
display_str, int(100*scores[i]))
964         box_to_display_str_map[box].append(
display_str)
965         if agnostic_mode:
966             box_to_color_map[box] = 'DarkOrange'
967         else:
968             box_to_color_map[box] = STANDARD_COLORS[
classes[i] % len(STANDARD_COLORS)]
969
970
971     # Draw all boxes onto image.
972     for box, color in box_to_color_map.items():
973         ymin, xmin, ymax, xmax = box
974         if instance_masks is not None:
975             draw_mask_on_image_array(
976                 image,

```

```

977         box_to_instance_masks_map[box],
978         color=color
979     )
980     if instance_boundaries is not None:
981         draw_mask_on_image_array(
982             image,
983             box_to_instance_boundaries_map[box],
984             color='red',
985             alpha=1.0
986         )
987     draw_text_on_image_array(
988         image,
989         ymin,
990         xmin,
991         ymax,
992         xmax,
993         color=color,
994         thickness=line_thickness,
995         display_str_list=box_to_display_str_map[box
996 ],
997         use_normalized_coordinates=
998 use_normalized_coordinates)
999     if keypoints is not None:
1000         draw_keypoints_on_image_array(
1001             image,
1002             box_to_keypoints_map[box],
1003             color=color,
1004             radius=line_thickness / 2,
1005             use_normalized_coordinates=
1006 use_normalized_coordinates)
1007
1008     # added to count the number of boxes drawn on
1009     the image - 07.03.2020
1010     print(Number of objects detected, "=",
1011           num_of_detection)
1012     #print(num_of_detection)
1013     #
1014     return image
1015
1016 def add_cdf_image_summary(values, name):
1017     """Adds a tf.summary.image for a CDF plot of the

```

```

1012 values.
1013
1014     Normalizes `values` such that they sum to 1,
        plots the cumulative distribution
1015     function and creates a tf image summary.
1016
1017     Args:
1018         values: a 1-D float32 tensor containing the
        values.
1019         name: name for the image summary.
1020     """
1021     def cdf_plot(values):
1022         """Numpy function to plot CDF."""
1023         normalized_values = values / np.sum(values)
1024         sorted_values = np.sort(normalized_values)
1025         cumulative_values = np.cumsum(sorted_values)
1026         fraction_of_examples = (np.arange(
        cumulative_values.size, dtype=np.float32)
1027                                 / cumulative_values.
        size)
1028         fig = plt.figure(frameon=False)
1029         ax = fig.add_subplot('111')
1030         ax.plot(fraction_of_examples, cumulative_values
        )
1031         ax.set_ylabel('cumulative normalized values')
1032         ax.set_xlabel('fraction of examples')
1033         fig.canvas.draw()
1034         width, height = fig.get_size_inches() * fig.
        get_dpi()
1035         image = np.fromstring(fig.canvas.tostring_rgb
        (), dtype='uint8').reshape(
1036             1, int(height), int(width), 3)
1037         return image
1038     cdf_plot = tf.py_func(cdf_plot, [values], tf.
        uint8)
1039     tf.summary.image(name, cdf_plot)
1040
1041
1042     def add_hist_image_summary(values, bins, name):
1043         """Adds a tf.summary.image for a histogram plot
        of the values.

```

```

1044
1045     Plots the histogram of values and creates a tf
       image summary.
1046
1047     Args:
1048         values: a 1-D float32 tensor containing the
       values.
1049         bins: bin edges which will be directly passed
       to np.histogram.
1050         name: name for the image summary.
1051     """
1052
1053     def hist_plot(values, bins):
1054         """Numpy function to plot hist."""
1055         fig = plt.figure(frameon=False)
1056         ax = fig.add_subplot('111')
1057         y, x = np.histogram(values, bins=bins)
1058         ax.plot(x[:-1], y)
1059         ax.set_ylabel('count')
1060         ax.set_xlabel('value')
1061         fig.canvas.draw()
1062         width, height = fig.get_size_inches() * fig.
       get_dpi()
1063         image = np.fromstring(
1064             fig.canvas.tostring_rgb(), dtype='uint8').
       reshape(
1065             1, int(height), int(width), 3)
1066         return image
1067     hist_plot = tf.py_func(hist_plot, [values, bins
       ], tf.uint8)
1068     tf.summary.image(name, hist_plot)
1069
1070
1071     class EvalMetricOpsVisualization(object):
1072         """Abstract base class responsible for
       visualizations during evaluation.
1073
1074         Currently, summary images are not run during
       evaluation. One way to produce
1075         evaluation images in Tensorboard is to provide tf
       .summary.image strings as

```

```

1076     `value_ops` in tf.estimator.EstimatorSpec's `
      eval_metric_ops`. This class is
1077     responsible for accruing images (with overlaid
      detections and groundtruth)
1078     and returning a dictionary that can be passed to
      `eval_metric_ops`.
1079     """
1080     __metaclass__ = abc.ABCMeta
1081
1082     def __init__(self,
1083                  category_index,
1084                  max_examples_to_draw=5,
1085                  max_boxes_to_draw=20,
1086                  min_score_thresh=0.2,
1087                  use_normalized_coordinates=True,
1088                  summary_name_prefix='
evaluation_image'):
1089         """Creates an EvalMetricOpsVisualization.
1090
1091         Args:
1092             category_index: A category index (dictionary
1093             ) produced from a labelmap.
1094             max_examples_to_draw: The maximum number of
1095             example summaries to produce.
1096             max_boxes_to_draw: The maximum number of
1097             boxes to draw for detections.
1098             min_score_thresh: The minimum score threshold
1099             for showing detections.
1100             use_normalized_coordinates: Whether to assume
1101             boxes and keypoints are in
1102             normalized coordinates (as opposed to
1103             absolute coordiantes).
1104             Default is True.
1105             summary_name_prefix: A string prefix for each
1106             image summary.
1107         """
1108
1109         self._category_index = category_index
1110         self._max_examples_to_draw =
max_examples_to_draw
1111         self._max_boxes_to_draw = max_boxes_to_draw

```

```

1105     self._min_score_thresh = min_score_thresh
1106     self._use_normalized_coordinates =
        use_normalized_coordinates
1107     self._summary_name_prefix = summary_name_prefix
1108     self._images = []
1109
1110     def clear(self):
1111         self._images = []
1112
1113     def add_images(self, images):
1114         """Store a list of images, each with shape [1,
        H, W, C]."""
1115         if len(self._images) >= self.
            _max_examples_to_draw:
1116             return
1117
1118         # Store images and clip list if necessary.
1119         self._images.extend(images)
1120         if len(self._images) > self.
            _max_examples_to_draw:
1121             self._images[self._max_examples_to_draw
                : ] = []
1122
1123     def get_estimator_eval_metric_ops(self, eval_dict
        ):
1124         """Returns metric ops for use in tf.estimator.
        EstimatorSpec.
1125
1126         Args:
1127             eval_dict: A dictionary that holds an image,
                groundtruth, and detections
1128                 for a batched example. Note that, we use
                only the first example for
1129                 visualization. See eval_util.
                result_dict_for_batched_example() for a
1130                 convenient method for constructing such a
                dictionary. The dictionary
1131                 contains
1132                 fields.InputDataFields.original_image: [
                batch_size, H, W, 3] image.
1133                 fields.InputDataFields.

```

```

1133 original_image_spatial_shape: [batch_size, 2]
1134         tensor containing the size of the
        original image.
1135         fields.InputDataFields.true_image_shape: [
        batch_size, 3]
1136         tensor containing the spatial size of the
        upadded original image.
1137         fields.InputDataFields.groundtruth_boxes
        - [batch_size, num_boxes, 4]
1138         float32 tensor with groundtruth boxes in
        range [0.0, 1.0].
1139         fields.InputDataFields.groundtruth_classes
        - [batch_size, num_boxes]
1140         int64 tensor with 1-indexed groundtruth
        classes.
1141         fields.InputDataFields.
        groundtruth_instance_masks - (optional)
1142         [batch_size, num_boxes, H, W] int64
        tensor with instance masks.
1143         fields.DetectionResultFields.
        detection_boxes - [batch_size,
1144         max_num_boxes, 4] float32 tensor with
        detection boxes in range [0.0,
1145         1.0].
1146         fields.DetectionResultFields.
        detection_classes - [batch_size,
1147         max_num_boxes] int64 tensor with 1-
        indexed detection classes.
1148         fields.DetectionResultFields.
        detection_scores - [batch_size,
1149         max_num_boxes] float32 tensor with
        detection scores.
1150         fields.DetectionResultFields.
        detection_masks - (optional) [batch_size,
1151         max_num_boxes, H, W] float32 tensor of
        binarized masks.
1152         fields.DetectionResultFields.
        detection_keypoints - (optional)
1153         [batch_size, max_num_boxes, num_keypoints
        , 2] float32 tensor with
1154         keypoints.

```

```

1155
1156     Returns:
1157         A dictionary of image summary names to tuple
1158         of (value_op, update_op). The
1159         `update_op` is the same for all items in the
1160         dictionary, and is
1161         responsible for saving a single side-by-side
1162         image with detections and
1163         groundtruth. Each `value_op` holds the tf.
1164         summary.image string for a given
1165         image.
1166     """
1167     if self._max_examples_to_draw == 0:
1168         return {}
1169     images = self.images_from_evaluation_dict(
1170         eval_dict)
1171
1172     def get_images():
1173         """Returns a list of images, padded to self.
1174         _max_images_to_draw."""
1175         images = self._images
1176         while len(images) < self.
1177         _max_examples_to_draw:
1178             images.append(np.array(0, dtype=np.uint8))
1179         self.clear()
1180         return images
1181
1182     def image_summary_or_default_string(
1183         summary_name, image):
1184         """Returns image summaries for non-padded
1185         elements."""
1186         return tf.cond(
1187             tf.equal(tf.size(tf.shape(image)), 4),
1188             lambda: tf.summary.image(summary_name,
1189                 image),
1190             lambda: tf.constant(''))
1191
1192     update_op = tf.py_func(self.add_images, [[
1193         images[0]]], [])
1194     image_tensors = tf.py_func(
1195         get_images, [], [tf.uint8] * self.

```

```

1184 _max_examples_to_draw)
1185     eval_metric_ops = {}
1186     for i, image in enumerate(image_tensors):
1187         summary_name = self._summary_name_prefix +
1188             '/' + str(i)
1189         value_op = image_summary_or_default_string(
1190             summary_name, image)
1191         eval_metric_ops[summary_name] = (value_op,
1192             update_op)
1193     return eval_metric_ops
1194
1195 @abc.abstractmethod
1196 def images_from_evaluation_dict(self, eval_dict):
1197     """Converts evaluation dictionary into a list
1198     of image tensors.
1199
1200     To be overridden by implementations.
1201
1202     Args:
1203         eval_dict: A dictionary with all the
1204         necessary information for producing
1205         visualizations.
1206
1207     Returns:
1208         A list of [1, H, W, C] uint8 tensors.
1209     """
1210     raise NotImplementedError
1211
1212 class VisualizeSingleFrameDetections(
1213     EvalMetricOpsVisualization):
1214     """Class responsible for single-frame object
1215     detection visualizations."""
1216
1217     def __init__(self,
1218                 category_index,
1219                 max_examples_to_draw=5,
1220                 max_boxes_to_draw=20,
1221                 min_score_thresh=0.2,
1222                 use_normalized_coordinates=True,
1223                 summary_name_prefix='

```

```
1217 Detections_Left_Groundtruth_Right'):
1218     super(VisualizeSingleFrameDetections, self).
1219         __init__(
1220             category_index=category_index,
1221             max_examples_to_draw=max_examples_to_draw,
1222             max_boxes_to_draw=max_boxes_to_draw,
1223             min_score_thresh=min_score_thresh,
1224             use_normalized_coordinates=
1225                 use_normalized_coordinates,
1226             summary_name_prefix=summary_name_prefix)
1227
1228     def images_from_evaluation_dict(self, eval_dict):
1229         return draw_side_by_side_evaluation_image(
1230             eval_dict, self._category_index, self.
1231                 _max_boxes_to_draw,
1232                 self._min_score_thresh, self.
1233                 _use_normalized_coordinates)
```

APPENDIX C



CIRRICULUM VITAE

Name and Surname : Yusuf UZUN

EDUCATION

Undergraduate: Middle East Technical University (ODTÜ), Electrical and Electronic Engineering 1991-1995

Master-1: Başkent University, Master of Business Administration (MBA), 1996-1999

Master-2: Duisburg-Essen University in Germany, M.Sc. in Computer Science and Communications Engineering, 2003-2006

Ph. D.: Aksaray University, Electrical Electronic and Computer Engineering, 2016-2022

PROFESSIONAL EXPERIENCE

1. Türk Telekom General Directorate, Senior Project Manager, since 2013
2. Huawei Telecommunications and Trade Ltd, Project Manager, 2009-2013
3. NEC Australia Pty. Ltd, Ankara, Senior Systems Engineer, 2006-2009
4. Siemens Communications AG in Germany, Database Design & Software Developer 2005-2006
5. Vodafone Holding GmbH in Germany, Project Engineer, 2004-2005
6. Duisburg-Essen University, Campus Duisburg, Germany, IT Specialist, 2003-2004
7. DHL Worldwide Express, IT Department in Istanbul, Application Support Expert, 2001-2002
8. Pelka-Albuj Joint Venture, Resident Engineer in Dubai for Customer Support and Business Development, 1998-2001
9. Başkent University Campus in Ankara, Research and Teaching Assistant, 1996-1998

REWARDS

During my work as a Project Manager for Huawei, I have been rewarded by Huawei's GOLD MEDAL PRIZE made of 30g pure gold. I have been selected to be the best Project Manager of Huawei globally among its all Project Managers working in 140 countries (Appendix C).

PUBLICATIONS DERIVED FROM THIS THESIS

Uzun, Y., Tolun, M. R., Eyyuboglu, H. T. and Sarı, F. (2022). An intelligent system for detecting Mediterranean fruit fly [Medfly; *Ceratitis capitata* (Wiedemann)], Journal of Agricultural Engineering, 53(3).
<https://doi.org/10.4081/jae.2022.1381>