Evaluating Different AI Models to Detect Polyhouses in Satellite Images

by

Arjun Gupta (SNR: 2017474)

A thesis submitted in partial fulfillment of the requirements for the degree of
Master in Business Analytics and Operations Research

Tilburg School of Economics and Management
Tilburg University

Supervised by: Martijn Schoot Uiterkamp

Date: August 16, 2023

Abstract

In this paper, we present a comparative study evaluating three convolutional neural network (CNN) models for detecting polyhouse coverage in the Indian state of Telangana. The ability to accurately map the extent of polyhouse coverage holds significant value for the United Nations Development Programme (UNDP), as it aids in formulating informed agricultural planning strategies. The three CNN models examined in our study include YOLO (YOLO version 5), YOLO-NAS, and Faster R-CNN, each renowned for its prowess in object detection tasks. We assess the suitability of these models for detecting polyhouses, with a particular emphasis on getting the number and coordinates of the polyhouses present. Our findings reveal that the YOLO model outperforms its counterparts in identifying and quantifying the number of polyhouses. Its remarkable performance in this regard enhances its practical utility for the UNDP's long-term agricultural planning initiatives. However, we do highlight some concerns regarding the models' generalization capabilities, particularly when the model is used on a new data set which deviates from the one that was used for training. This suggests that one has to exercise caution when applying the model to new states that were not part of the training data, as polyhouses might look significantly different which has proven to be extremely important to the performance of the model.

Contents

1 Introduction

In a world where man and climate are more at odds with each other than ever before, polyhouses present a way to protect crops and therefore the food chain from ever-changing weather conditions. Polyhouses are plastic or polythene structures that enable controlled growth of plants. Therefore, interest and investment in polyhouse farming has increased substantially over the last decades and continues to do so. Polyhouse farming is already an integral part of the food chain and as such it is important to monitor polyhouse coverage to inform long-term planning and interventions in the agriculture sector. In this study we will focus on getting the coverage of the polyhouses, by getting their number and location, in Telangana state of India, a region where agriculture is very much the backbone of the economy and where government reports already deem living and coping with the impacts of climate change "essential for our survival". This information can then be used by UNDP and local governments to target interventions where there are not enough polyhouses or for other structural reforms. Also, it can serve as a starting point for further studies into the effectiveness of polyhouses and to plan future deployment of polyhouses. This way our research will help the region of Telangana in their efforts to deal with the consequences of climate change and ensure the food security of the people. Therefore, our research question is: What is the best machine learning model to detect the location and quantity of polyhouses in the state of Telangana using satellite imagery?

We will examine the distribution of polyhouses across the region using satellite imagery. We are going to train a few state of the art machine learning models to detect polyhouses in these satellite images. Insights from individual images will then be aggregated into a comprehensive overview of coverage of polyhouses across the whole state. Specifically, we will compare three convolutional neural network architectures to see how they perform. That means that the contribution of our research is twofold, leading to better knowledge about the coverage of polyhouses in Telangana and comparing the performance of three highly advanced models for the task of object detection. As with any data-related project, the availability and quality of the data will be an important factor in our research. There are some open sources where one can retrieve satellite imaging data with varying degrees of quality and coverage, though it remains to be seen how many of them contain data that is useful for this research. It has to be noted that the models that we compile are limited to the range of photos that we can label as training data, meaning that results will not necessarily translate seamlessly to other geographical regions. In this research we are first going to have the models predict the so-called 'bounding boxes' of the polyhouses, indicating where in an image the model thinks it has detected a polyhouse. After assessing the accuracy of these predicted bounding boxes, we will use the model that provides high accuracy and low overfitting to get the number of polyhouses and their coordinates in a distict of Telangana called Ranga Reddy. Note that in this research, we are primarily interested in getting these right and not so much in the size of the polyhouses. Also, as we are using satellite imaging it is important to consider that we will only model the presence and location of polyhouses. The quality of the polyhouse, what crops are grown inside and any other qualitative aspects will not be part of the model and therefore of the mapping exercise. The rest of this thesis will be structured as follows: In chapter 2 we will discuss literature relevant to our topic and what our research adds to the existing body of literature. After that, in chapter 3 we will describe the process of choosing which models to use, going into detail about how the models that we choose relate to the specific task at hand. Chapter 4 discusses how we implemented the models that we are using to model the satellite images as well as any other relevant methodology and chapter 5 summarizes and visualizes the results of the research. Finally, in chapter 6, we will discuss the findings from the results and potential areas for future research.

2 Literature review

Before going into detail about our methodological approach, it is important to assess the research that has already been done and the methods and data that are already available. In this section we will conduct a literature review of various themes that are relevant to our research question. First, in section 2.1 we will examine research that has gone into polyhouse cultivation, examining its pros and cons. Next, in section 2.2 we will assess research about long-term agricultural planning, as this is the eventual use of our mapping exercise. After that we will look at studies related to the use of satellite imagery in section 2.3. Next, in section 2.4 we will have a look at the literature specific to the geographical region where we assess polyhouse coverage to get a better idea of the context. Finally, section 2.5 discusses the concept of detection of objects in images and how we will approach this challenge in our research.

2.1 Polyhouse farming

As discussed, in this research we will examine polyhouse coverage in the Indian state of Telangana. As in many other parts of the world, polyhouse cultivation has grown very fast in Telangana to provide a more efficient and robust alternative to open-field farming. The use of polyhouses is part of a wider trend of using more technological advances in the agriculture sector, that was discussed in more detail by Mirabella and Brischetto 2010. As polyhouses enable farmers to control certain conditions under which their crops grow, they can have various beneficial effects. One of the many advantages of polyhouses, as mentioned by Murthy et al. 2009, is that the crops in a polyhouse are protected against both abiotic stresses such as pests, diseases and weeds and also biotic factors such as temperature and humidity thus, enhancing the quality and productivity of the crops. This helps farmers to grow crops irrespective of the season. A comparative study by Sharma, Gaur, Singh, et al. 2007 shows that use of polyhouses helps crops grow faster and Kumar, Chauhan, and Grover 2016 show that using polyhouses to grow tomatoes in India can contribute significantly to crop yield. Additionally, new techniques are being developed continuously to make the advantages of using polyhouses even bigger. For instance, according to Jonnala and Satrasupalli 2013, the temperature inside a polyhouse can be maintained by minimum hardware and human effort by using LM35 sensor and a polyhouse cooling fan, which sets up the threshold temperature automatically to increase the crop yield. Another such technology is the so called Zigbee technology, as mentioned by Jonnala and Sathyanarayana 2015, which optimizes the vitals for a crop growth such as light intensity, soil moisture and humidity percentage using sensor data. In addition to gaining popularity, these technologies are also getting upgraded, for instance,Mirabella and Brischetto 2010 discuss Zigbee technology becoming wireless, which is making the polyhouses a pick for farmers. One factor to consider when discussing the merits of polyhouse farming is economic viability. Murthy et al. 2009 find that size of a polyhouse can range from 1000m^2 to 10,000m^2 and so does cost, depending on the specific type of polyhouse.

2.2 Agricultural planning

As polyhouse cultivation is an important addition to farming in Telangana, UNDP wants to map coverage of polyhouses across the state as a tool to assist in strategic planning. Using satellite imagery can be a vital part of long-term agricultural planning, as outlined by Balcik, Senel, and Goksel 2020. There are various advantages to having a long-term strategy with respect to farming. For example, as explained by Smith and McDonald 1998, identification of sustainable agricultural practices is important at the planning stage to avoid unnecessary ill uses of land such as deforestation for cultivation, structural reforms on unused land, polluting land and soil quality with irrigation and so forth. Other consideration where long-term planning is involved are using technology to reduce dependency on chemicals (Reganold, Papendick, and Parr 1990), farmland allocation (Boyabatlı, Nasiry, and Zhou 2019) and various others (De Wrachien 2003). Especially as farming has found its way from being an exclusively rural practice into the cities, planning becomes ever more important. Urban agriculture planning can have several benefits, starting from making the area greener, healthier supplies thus making the people healthier, and other such social and economic benefits as pointed out by Lovell 2010. A case study, comparing different time periods of planning in advance by Vermeulen et al. 2013, explains evaluation of uncertainty can help in decision making. Moreover, focusing on the geographical area of interest, that is India, Raja et al. 1997 quantified, using models, that through proper planning there was an increase of about 54% revenue per hectare due to proper planning. Another factor accelerating the need for a more long term outlook on farming is climate change. In their paper Agovino et al. 2019 proved the bidirectional relationship between agriculture practices and climate change, for instance, agricultural activities release gasses in the atmosphere, making temperature go higher and thus affecting the yields. Hence, there is a need adapt practices to make agriculture more resilient to climate changes as discussed by Howden et al. 2007.

2.3 Sattelite imagery

Therefore, we will assess the coverage of polyhouses in Telangana. In this study we will use satellite imaging to do so. The use of satellite images has increased dramatically over recent years, both because of more data becoming available and more advanced algorithms being developed to analyse them. As such, we already have a body of literature with ideas to build on. Satellite images have various characteristics as mentioned by Jacobsen 2005, such as the distance between two adjacent pixels measured on the ground, known as ground sampling distance (which should be generally 0.05mm to 0.1mm) and an image sensor used to capture the image of moving objects called transfer delay and integration sensor. The popularity of satellite images have increased due to the inferences they present. Velásquez et al. 2020 used remote sensing using drone imagery to detect pests in trees. In this study, the task of human visualisation was performed by using remote sensing it was concluded that statistically, there was no difference in human visualisation and via technology integration. Other such studies have been done by Tellman et al. 2021, to estimate and predict the flood extent and how many people would be affected by it, Mathieson et al. 2009 to monitor carbon dioxide movement in Algeria, by Jean et al. 2016 to predict poverty and by Verpoorter et al. 2014 to build an inventory of lakes and use that to calculate contributions to the global carbon cycle. All these studies were done using satellite images. In fact, Sun et al. 2021 did a study in China to detect greenhouses using satellite images. However, the satellite image data often needs processing due to wide range of characteristics, such as different resolutions, scales and presentation(Bonnett and J. Campbell 2002). Moreover, there often exist geometric distortions in a satellite image data, thus pre processing is needed in the raw data for analysis like object detection, as explained by Dave, Joshi, and S. Srivastava 2015. This also applies to polyhouse detection and therefore to this research. It must also be noted that even after processing, different methods yield different results based on the scenario we are applying it to and our selection can greatly influence the results (Asokan et al. 2020).

2.4 Telangana

Although many of the ideas in the previous section definitely help us on our way, the majority of them are either modelling a slightly different thing or take place in different geographical settings. In fact, even the mapping of polyhouses has already been tried in other contexts, for example Hong et al. 2023. In this study, however, we will have to take into account the specific context of India and the Telangana state in order to come up with a solution that best serves UNDP's needs. Around 60 % of India's population is in the agriculture sector, however the share of agriculture has fallen from the GDP and the gap between the rich and the poor has widened (Kurian 2008). Kadiyala et al. 2014 presented agriculture as a tool to generate income, reduce malnutrition and improve diets. Heavy use of chemicals in agriculture has helped India shed a long-standing food deficit. Now it wants to be more ecologically responsible but worries about if the whole country can be nourished this way (Yadav et al. 2013). Polyhouses can be a solution to these problems as they provide a room for organic farming to protect the crops from biotic and abiotic stresses. Production growth in India in the last century was due to expansion of irrigated area. The limits of this expansion have been reached so there's also a necessity for more sustainable agriculture (Kerr et al. 1996). Looking at the specific case of Telangana, irrigated area has increased to increase production but with risks of adverse affects for groundwater levels and on small farmers (Vakulabharanam 2004). Adimalla et al. 2020 also explains how the intensity of irrigation poses a serious danger for the quality of groundwater in Telangana. Polyhouses once again can play a role here by collecting the rainwater on the plastic roof and storing in for later irrigation purposes.

2.5 Detection

As stated, the goal of this research is to detect polyhouses in satellite images of the Telangana region. In general, in object detection we ask the model to estimate not just the presence of an object, but also its location. This is usually done through the concept of bounding boxes. Bounding boxes are a fundamental concept in the field of object recognition, providing a framework for localizing and identifying objects within digital images or videos. A bounding box is essentially a rectangular region that encloses an object of interest. It serves as a visual representation of the object's location and extent within the image, enabling algorithms to precisely locate and classify objects. Bounding boxes are typically defined by four parameters: the coordinates of the box's top-left and bottom-right corners. By extracting these regions of interest, object recognition algorithms can focus their attention on specific areas, facilitating tasks such as object detection, tracking, and segmentation. These bounding boxes that are predicted are then compared to the true location of the objects in the image to assess the correctness of the predictions. In our research, the bounding boxes are then the areas in the image where the models predict the presence of polyhouses. The initial goal of this modelling exercise will be to get those bounding boxes as accurate as possible, so as close as possible to the actual locations of the polyhouses. Once these bounding boxes are as accurate as possible, indicating that the model has learned to map the features in the images to the relevant objects, there can be various angles of interest that require different modelling. One such example is the number of objects that are predicted, see for example Y. Huang et al. 2023. Another priority can be not the number but the size of objects, as discussed by Long, Shelhamer, and Darrell 2015. Yet another example could be the exact location of the objects, something that received a lot of attention in Chen et al. 2017. In our research, once we get the model to predict these bounding boxes as accurately as possible we use the predicted bounding boxes to extract the coordinates and consequently the number of predicted polyhouses in an image.

3 Model selection

In previous sections we have tried to outline some important considerations when modelling the presence of polyhouses in satellite images from Telangana. We are primarily interested in predicting the coordinates and the amount of polyhouses which boils down to predicting the bounding boxes in a satellite image as accurately as possible. The speed of the algorithms is not a key consideration as the mapping exercise will not be done very frequently since the polyhouses do not move every day. We will begin this section by explaining the process of how we got to the models that we use in this research. We will describe for every step what the considerations were for choosing a certain (type of) model and how this choice relates to our specific research. Once we have given motivation for our use of model, we will discuss the models of choice in more detail.

First, it is important to note that various models have different merits when it comes to both accuracy and ability to explain its predictions. The former refers to how well the output of the model matches the phenomenon that is being modelled and the later to what degree it is clear how the model maps input to output. In this research, we are not so interested in what pixels or colours drive the algorithm to predict polyhouse coverage, the important thing is that the predictions are as accurate as they can be. Therefore, we will only consider more 'black box' approaches and not use relatively simple methods such as logistical regression that offer more explainability but generally do not reach the same levels of predictive accuracy. Before diving into the models, we would introduce the concept of Neural networks which is the base of these models and the Convolutional neural networks.

3.1 Neural networks

The combination of the importance of predictive accuracy and the type of data that we will use make neural network an appealing choice of model to use, as neural networks have become the golden standard in regression and classification tasks (Alzubaidi et al. 2021). Neural networks have a number of advantages such as having high processing speed, ability to learn from examples provided and adapt to change with high accuracy when trained with large and suitable data, as shown in Bishop 1994. Neural networks consist of an architecture that are made up of different neurons that represent underlying features in the data and connections between these neurons. A lot of research has gone into these architectures and other features of neural networks to make them learn from large bodies of data even quicker and more efficiently. Some examples are dropping out some observations from different networks and averaging out the predictions to avoid overfitting (N. Srivastava et al. 2014), He et al. 2016 show how to make even deeper neural networks trainable and transfer learning (Yosinski et al. 2014). Using this ever expanding toolkit, neural networks have yielded impressive results in a wide variety of learning tasks, ranging from but not limited to face recognition (Kasar, Bhattacharyya, and T. Kim 2016), economics (W. Huang et al. 2007) and clinical medicine (Baxt 1995). As neural networks are able to learn very quickly with the data provided, they have become the most popular option to analyze imaging data and hence a fit to our case.

3.2 Convolutional neural networks

Another important feature that might influence model selection is that in this research we are dealing with image data, specifically satellite images. In order to quantitatively analyze image data, the general practice is to quantify its color per pixel, usually using so-called red-green-blue notation. This means that every pixel of every image gives a quantitative score of how the color of that image is a mix of the three fundamental colors, red, green and blue. This tends to lead to vast data structures, and although neural networks are known to be able to exploit large volumes of data, the size of the input and the number of parameters to be estimated when taking in this image data can lead to a number of problems such as overfitting. Therefore so-called Convolutional neural networks (CNNs) were designed, pioneered by LeCun et al. 1998 among others. In a CNN, each image is divided into a number of sub-images that are then summarized and used as an input for the neural network. This drastically brings down the dimensionality of the problem without throwing away relevant information. The convolutional and the pooling layers that were alluded to before are important features of convolutional neural networks, as convolutional layers apply a set of filters to detect features in the image that can be optimized during training. After several layers of convolution and pooling, the output of the network is typically fed into a fully connected layer, which functions similarly to the hidden layers in traditional neural networks. This layer uses the features extracted by the earlier layers to make a final prediction about the image. The output of this layer is typically passed through a softmax function to produce a probability distribution over the different possible classes.

3.2.1 Object detection

As stated, we are not just looking to model a binary indicator for the presence of polyhouses but we are looking for bounding boxes that also tell us their location (coordinates in the respective image) and how many. For this reason, a host of neural network designs, such as ResNET (He et al. 2016), were not considered that do not give us the bounding boxes that we are looking for and only output a binary variable if polyhouses are present or not. Our research corresponds to a computer vision task called object detection, where the algorithm looks at a digital image (satellite image in our case) and is asked to distinguish which of a pre-defined set of items is where in the image. The way neural networks interrogate large bodies of data such as images has greatly accelerated momentum of object detection technology and has led to them now being the golden standard when it comes to detecting objects in images. In general, object detection algorithms can be divided in two groups: one-stage vs two-stage algorithms. In order to understand this distinction, we further scrutinize what an object detector actually does. A deep learning object detection algorithm first extracts relevant features from the pixels that are given as input and then performs the following tasks:

1. Find a number of objects(polyhouses in our case) in the image, note that this number can also be zero
2. Classify the objects as one of a pre-determined number of items and estimates the bounding box of the item

3.3 One-step estimators

Two-step estimators do these two steps separately, first proposing object regions using the deep neural network and then classifying the images using bounding box regression. One-step estimators, on the other hand, skip the region proposal stage and instead go straight to predicting bounding boxes over the images. In general, two-step estimators reach higher accuracy but are slower than one-step estimators. This also applies when detecting polyhouses, as demonstrated by Li et al. 2020. While this means that one-step algorithms can even be used to detect images in real time where the extra computational effort for two-step estimators does not allow for this. This could be taken as an indication that we should only include two-step estimators, as predictive accuracy is the main focus of the models. However, although in our research speed is not really an issue, there is mixed evidence on the respective predictive accuracy of one-step and two-step estimators, see for example J.-a. Kim, Sung, and Park 2020 or Li et al. 2020. Therefore we use the most common one-step estimator, the YOLO (You Only Look Once) algorithm, and compare it to a two-step estimator.

3.3.1 YOLO

The YOLO estimator was introduced by Redmon et al. 2016 and is so named because the algorithm requires only a single forward propagation through the algorithm to detect objects, allowing for the algorithm to detect objects in real time. Aside from its subsequent speed and computational efficiency, the YOLO algorithm has reached a very high accuracy in a number of tasks and can predict not just if an object is present in an image but also where in the image it is. This is important, as the accuracy of the predicted bounding boxes is the first step in our detection framework, and the step on which the accuracy of the polyhouse coverage that we predict relies. The YOLO estimator begins by dividing the input image in a S by S grid cells. The cell in which the center of each object falls in responsible for its detection. Each grid cell outputs a given number of bounding boxes, corresponding to possible objects whose center is in that grid cell, along with a confidence score reflecting the likelihood that this bounding box actually is an object. In addition, each bounding box has a prediction to which class the object belongs. Note that each bounding box comes with only one predicted class making it difficult to predict when bounding boxes contain multiple items of different classes. However, as we only predict polyhouses in this research and therefore have only one class this will not be a limiting factor in our research.

Multi-scale Approach

Another important feature of YOLO is its use of a multi-scale approach. This means that the algorithm looks at the image at multiple scales, allowing it to detect objects of different sizes. The algorithm uses a pyramid of feature maps to detect objects at different scales, and then combines these feature maps to generate a final set of object detections. Although scale in itself is not a major factor of interest to us, the ability to detect objects of various sizes is very important. This will allow the model to be more accurate in detecting the correct number of polyhouses in an image, also if they happen to be of different sizes which can very well be the case with polyhouses. YOLO also uses a data augmentation technique called self-adversarial training, which helps to improve the robustness of the algorithm to different types of noise and distortion in the input data. The algorithm generates adversarial examples by perturbing the input data in different ways, and then trains on these examples to make the network more resilient to these types of distortions. This is a helpful feature as data for this research is naturally sparse and any feature that helps the algorithm squeeze the best possible performance from it is helpful.

Architecture

Broadly speaking, the architecture of the YOLO estimator consists of three parts:

1. The backbone: in this part, the raw input (the image) is transformed and aggregated to represent features
2. The neck: in this part, the different features are combined and mixed to be passed forward to prediction
3. The head: takes the features and uses them to predict boxes and classes

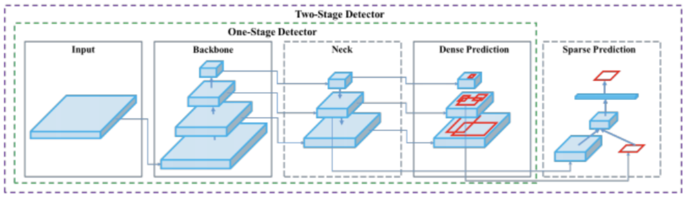

Figure 1: Visual representation showing the outline of the architecture of the YOLO algorithm (source: https://iq.opengenus.org/yolov5/)

The backbone is a network made up of 24 convolutional layers to detect features in the images. 20 of these convolutional layers are typically pre-trained on a dataset such as imageNet and the backbone is trained at a lower resolution than the final detection model as that task requires more granularity. These convolutional layers are followed by an average pooling and a fully-connected layer. The neck consists of fully connected layers that take the features from the convolutional layers and output predictions on bounding boxes and classes of the objects. The head is the final output layer of the network. The neck uses a linear activation function where the layers before it use a ReLU (rectified linear unit) activation function. In order to allow for transfer learning the head can be interchanged with other layers of the same size in the network.

In order to diversify the features learned by the model and prevent overfitting, YOLO uses both data augmentation and dropout. The dropout layer is located between the first and second fully connected layer and uses a dropout rate of 0.5 to discourage them from learning the same thing. This helps as learning more diverse features means that the detection of polyhouses will be made more robust. For data augmentation, the YOLO algorithm uses a technique called 'mosaic data augmentation'. For this, the algorithm takes four different images from the input, resizes them, and combines them into a single mosaic image that is then used as extra input for the model.

The speed of the YOLO algorithm as well as the straightforward architecture that is easy to understand make it a good choice. Also, both the use of grid cells and the fact that the single pass through the network enables the network to consider an object in the context of the whole image mean that the YOLO algorithm does a particularly good job at detecting objects of various sizes.

3.3.2 YOLO-NAS

The YOLO-NAS algorithm is an extension to the standard YOLO algorithm that was described in the previous section. It shares the same foundational concept of one-shot object detection with the regular YOLO algorithm but includes a separate step to optimize the network architecture for the object detection task at hand. This could be an important addition to the regular YOLO algorithm as there is not that much data to learn from so any additional hyperparameters that could help the model improve would be very welcome. We do, however, invite some risk of overfitting.

Neural architecture search

In this so-called architecture search step, first a search space is defined. This space consists of the set of architectural decisions that the algorithm can make when optimizing the architecture. This search space typically includes various architectural components, such as layer sizes, types of layers (e.g., convolutional, pooling), connectivity patterns, and operations (e.g., skip connections, residual blocks). Once the search space is defined, the algorithm has to use an exploration algorithm to iterate through the search space looking for the best architecture configuration. Common choices for the exploration algorithm are reinforcement learning, Bayesian optimization or an evolutionary algorithm. The exploration algorithm begins by suggesting a number of candidate architectures from the search space, each representing a different combination from the hyperparameters that are defined in the search space. Each of these architectures is then trained and evaluated on pre-set training and validation sets of the data. They are compared based on predictive accuracy, guiding the algorithm towards more promising architectures from the search space. Note that when needed the algorithm can take into account other measures as well, such as runtime or memory usage. The algorithm then starts a new iteration by proposing new candidate architectures based on the performance of the candidates in the last round. It will then evaluate those new candidates and keep going until a stopping criterion is met. Once the algorithm has stopped, the best of all the considered candidate architectures is chosen to be the final architecture. This architecture represents the optimal configuration of the YOLO-NAS algorithm. With this architecture, the model is trained on a larger dataset as one would with the original YOLO architecture. The added flexibility gives the advantage that the model has more ways to tune itself to predict the bounding boxes of the polyhouses as accurately as possible. Although it has some added flexibility, the YOLO-NAS is still rooted in the original YOLO algorithm in a number of ways. As with the original YOLO algorithm, it is a one-shot estimator that detects objects with a single pass through the network. YOLO-NAS also follows the general object detection pipeline that is used in YOLO. It involves preprocessing the input image, passing it through the network architecture, generating bounding box predictions, applying non-maximum suppression to remove redundant detections, and outputting the final object detection results. It also uses the same loss function and training procedure as the original YOLO algorithm.

By allowing the architecture to adapt, YOLO-NAS can capture object details more effectively and learn more efficient feature representations, enhancing the model's detection capabilities. Furthermore, YOLO-NAS offers flexibility and adaptability, as the discovered architecture can be easily applied to different object detection scenarios and datasets. This adaptability makes YOLO-NAS a versatile choice that can generalize well across diverse object detection tasks. However, the extra architecture search step comes at a computational cost, and the extra flexibility that the model is granted in finding the optimal architecture configuration can cause the YOLO-NAS model to suffer from overfitting by tuning the architecture too specifically to the training data. It is also less commonly used than the original YOLO algorithm which means that there are less resources available.

3.4 Two-step estimators

The two-step algorithm that we consider is the faster region based CNN (F-RCNN) algorithm. As its name suggests this is an extension of the earlier region based CNN. The main feature of this RCNN is that it's a combination of algorithms that execute multiple steps in the object detection process.

3.4.1 Faster RCNN

The Faster R-CNN (Region-based Convolutional Neural Network) is a deep learning-based object detection algorithm that was introduced by Ren et al. 2015. It is a two-stage procedure for detecting objects in an image, which has proven to be a powerful approach for object detection tasks.

First stage

In the first stage of the Faster R-CNN, the region proposal network (RPN) generates potential object locations, or regions of interest (RoIs), in an image. The RPN is a fully convolutional network that takes an image as input and outputs a set of rectangular object proposals, each of which is associated with a confidence score. The RPN scans the entire image with a set of predefined anchor boxes at different scales and aspect ratios, and predicts the likelihood of an object being present in each of these regions. This first stage helps the algorithm identify all potential polyhouses at an early stage, and is very tuned to this task as it already looks for rectangular bounding boxes, which tends to be the shape of polyhouses.

Second stage

In the second stage, the RoIs generated by the RPN are processed by a second network, typically a Fast R-CNN network (see Girshick 2015), to classify the objects and refine their locations. This network takes each RoI as input and applies a sequence of fully connected layers to extract features from it. The resulting feature vector is then fed into two separate fully connected layers: one that outputs a softmax distribution over the object classes, and another that predicts the offset values for the bounding box coordinates of the object within the RoI. The RPN and Fast R-CNN networks are both based on CNN architectures and are trained jointly in an end-to-end fashion.

3.5 Comparison

In summary, algorithms like YOLO, YOLO-NAS, and Faster R-CNN offer distinct advantages that make them suitable choices, listed in table 1. YOLO stands out for its simplicity and real-time performance, allowing for efficient deployment even in non-real-time scenarios. Its single-shot detection approach and multi-scale feature fusion contribute to accurate detection of small objects. YOLO-NAS, with its architecture search process, can adapt and optimize the network specifically for the task at hand, potentially achieving improved detection performance and generalization. Faster R-CNN, on the other hand, excels in accurate object localization and high detection accuracy, making it a robust choice for tasks where precision is paramount. With its widely adopted and extensively researched nature, Faster R-CNN offers a solid foundation with ample resources available for implementation and further development.

	Pro	**Con**
YOLO	Fast Good performance on small objects Detection of objects of various sizes	Localization accuracy Limited flexibility in architecture
YOLO-NAS	Improved performance Generalization Flexibility	Increased complexity Risk of overfitting Limited resources
Faster R-CNN	Accurate localization High accuracy	Slow Complexity Needs lot of data

Table 1: Relative weaknesses and strengths of the algorithms

4 Methodology

Given the problem that we set out to solve and the data available to us to solve it, there are a number of methodological steps to take. In the last section we have assessed the available methods in the literature and given motivation for the ones that we will be using. In this section, we will provide more detail about these methods and our specific implementation. First we will describe the steps we took in collecting data. Then we describe the steps taken for processing and manipulating the data that we have to make it ready for analysis. After that we will go into more detail about how we implemented each of the models that we compare. Then we will explain what we did to assess the quality of the models and how to rate their performance. This is especially important as we use several different methods and we need to be able to compare them and assess their relative performance.

4.1 Data Collection

Unfortunately, there is no existing database containing the presence or images of polyhouses in Telangana so we had to build the database that the model learns from ourselves. There is a vast range in terms of the quality of the satellite images available from different sources. It must be noted that the initial task would be to label the images to train the model on what is a polyhouse. Hence, we need the images wherein we can detect the polyhouses first with the naked eye in order to label them. We tried to go for open sources like Google Earth Engine and the best quality available there was Sentinel 2 images from Telangana which were not suitable for our research as labelling was not possible due to poor quality of the images. Other sources had high quality images such as Planet Labs, but were not open, thus we decided to go for the google earth data which had far better quality and polyhouses could be easily labelled. We manually proceeded with getting the coordinates of the polyhouses by hovering over each part of Telangana, checking the areas where we could find polyhouses and collecting the respective coordinates. Then we used the Google Earth Pro software (https://www.google.com/earth/versions/) to get the images having these coordinates. These images were of much better quality and gave a decent idea on where are polyhouses. Also, it must be noted, that it is most likely better if the quality of the images is higher, giving us more accuracy to label and hence train the model.

4.1.1 Positive Examples

First of all we manually searched for polyhouses in Telangana, to make sure we had positive examples to feed the models. In order to increase the size of the dataset, we looked for polyhouses in the states of Uttrakhand and Maharashtra also as they had polyhouses that looked similar to the ones in Telangana. Once we found them, we checked the land piece by piece and collected their coordinates and collected the images as mentioned in the previous section. We gathered all these images and labelled them using the Edge Impulse software (https://edgeimpulse.com/) that yielded a text file with coordinates of all bounding boxes of polyhouses that we found to be used as input by the model. The images are used by the model as the input to learn from and the text file with the coordinates of the bounding boxes is what it maps the images to.

4.1.2 Negative Examples

Next, we repeat the process but this time for images that have no polyhouses present. It is important for the model to be trained using both images that contain polyhouses and images that do not. If we only feed it images that contain polyhouses, the model will overestimate the likelihood of polyhouses appearing, leading to a bias when we later use this algorithm to assess polyhouse density across the whole state of Telangana. Note that in this final mapping exercise, the model will also encounter images without any polyhouses, so it is important that it also learns to recognize this in the training stage. As it will be infeasible to build a database with many thousands of images, we try and make sure that the images that we do manually collect are as informative as possible. Therefore, for the negative examples (images without polyhouses), we look for images that look similar to polyhouses but are not, to make sure the model will learn from particularly difficult examples to recognize. If we were to feed it only negative examples of mountains or rainforest (images where it is immediately obvious that there are no polyhouses), then the model might be inclined to predict the presence of polyhouses in any case that looks somewhat similar, as it would look more like the training data of polyhouses than the training data without polyhouses. However, having the model learn from examples that look somewhat like polyhouses but are not, such as houses, it will tune its parameter to learn the more granular specific qualities of a polyhouses, which is what we are looking for when later applying it across the whole state of Telangana. We therefore look for buildings that look similar to polyhouses and save the images in the same manner as before using Google Earth Pro. Note that this time, we do not need to add any coordinates, as in these examples there are no correct bounding boxes, as there are not polyhouses in the images.

Feeding the model all the images and the coordinates of the correct bounding boxes, we ensure that it will do as good a job as possible at predicting the presence of polyhouses with limited data.

4.1.3 Ranga Reddy images

After learning from the labelled data that we use to train and assess the model, there is a subsequent step where we apply the best model to a new body of data in Telangana to map the polyhouse coverage in the region. This data will not be labelled as finding labels for it is one of the outputs of this research. We will use the district of Ranga Reddy, a district that borders the capital of Telangana, as the area to map polyhouses. It is well-populated, meaning there are a lot of buildings to make the task sufficiently complex for the model, and there is a significant number of polyhouses in the region. Ranga Reddy has an area of 5,031 square kms. In order to make the marking of polyhouses visible to us, we need to cut down the image of the full district into smaller images. Moreover, we take the part of a district which has some polyhouses to give ourselves a better insight. Thus we collect images covering about 69 square km area and divide it into 276 images covering about about 0.25 square kms each and put the images into one folder, which can be directly fed to the model. To do this, we have used the same software, Google Earth Pro.

4.2 Data augmentation

We were able to gather only limited data. When dealing with a limited volume of data one potential solution can be to augment the data by applying transformations to the existing body of images and feed those transformed images to the model as well as the original. In our research, we will apply various methods to transform the images. Using many different methods rather than just one or a few has the advantage that it will not allow the model to tweak itself especially to capture the different features of one transformation (for example color), but it has to really be alert for all sorts of features that might change about the data. Specifically, we transformed the data in the following ways:

- Flip - the image is flipped over either the horizontal or the vertical axis to generate a mirror image.
- Rotate - the image is rotated to end up with a different angle of the same image.
- Shear - involves applying a distortion to an image along a specific axis, causing a slanting or tilting effect. In the context of images, this can simulate the effect of viewing an object from a different angle or perspective.
- Brightness - the brightness of the image is adjusted to generate the same image with a different levels of light
- Saturation - the level of saturation is randomly changed, affecting the vividness of the colours in the image
- Blur - the images are randomly blurred, lowering the focus of the image. This will generally make it more difficult to detect objects but is nevertheless useful to learn from.

All transformations were done using the roboflow tool (https://roboflow.com/, Inc. 2023).

4.3 Train-test split

In order to both compile the model and assess performance, the labelled data that we do have has to be split in a train, a validation and a test set. This is good practice to make sure we assess model performance over a dataset that has not been seen by the model in the training process in order not to artificially inflate performance. The train set is used as input for the model to learn from and tune its parameters. The validation set is then used to optimize the hyperparameters of the neural networks. Once a final optimal model has been established, the performance of that model is evaluated over the test set to allow for comparison between the different models. We use various ways of splitting the data that is available to us between the train, validation and test set to assess if and how they affect performance. Note that from here on out, every such split will be referred to as a 'dataset'. Specifically, we will split the data in the following five ways:

1. Dataset 1: The training and validation sets are the polyhouse images from Telangana and Uttrakhand. The final models are then tested on the images from Maharashtra. This way we not only assess the performance of the models but also how well they perform when presented with data from another state than the one where the training data came from and when the training and test data is not so similar.

2. Dataset 2: For the second split, we randomly take 50 images from the set of buildings that look like polyhouses but are not and, using various data augmentation techniques, we create 90 new images from them. Of these 90 new images, 72 are added to the train set and 18 to the validation set of dataset 1 and thus dataset 2 is created. By adding the non polyhouse images to the train and validation set we help the model learn about what is not a polyhouse.

3. Dataset 3: For the third split, we also randomly take 50 of the images where polyhouses are present and, using various data augmentation techniques, create new ones. Note that these images are added on top of the images that were already added in split 2. This will increase the size of the dataset.

4. Dataset 4: Next, we take the 370 original images that we had before and instead of taking images from certain states together, we randomly divide them between the train, validation and test set to compare what happens if the distribution between the different sets is more similar.

5. Dataset 5: For the last split, we do the same as in split 4, but this time we add 437 augmented images that are added to the train and the validation set, to compare what happens if the size of the dataset is increased.

In summary, dataset 1 contains only polyhouse images and is tested on a new area to check the robustness of the model. Dataset 2 contains combined images of polyhouses and non polyhouses, and dataset 3 is made up by adding more polyhouse images to dataset 2. In these 3 datasets, we will do polyhouse and non polyhouse testing separately in order to compare the accuracy in both the cases. Considering shortage of data, to increase the size of dataset, we combine this data and use it for dataset 4 and 5 and only do a combined testing of images of which some contain polyhouse and some do not. In table 2, we show an overview of the number of images of each of the datasets under each of the splits.

	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5
Train	136	208	360	236	586
Validation	34	52	90	59	146
Test	200	200	200	74	74

Table 2: Sizes of the different sets that are used under different splits

4.4 Modelling

All models were programmed in Python. The Faster R-CNN model was compiled using the TensorFlow library, for the YOLO models we used PyTorch. All of them are openly available on GitHub (Ultralytics and Jocher 2023, LLC 2023, Tran 2023) and other sources (Gopani 2020, Techzizou 2022, Skalski 2023, Rath 2023). We did manually tune some parameters to make the specific model more connected to our research as will be described in section 4.5. Also, we have manually transformed the model output to the output that is presented in this research, as well as plots to understand the role of the tuned parameters and we extracted the number and location of polyhouses from the standard output, all with the aim to make the output more actionable for UNDP.

4.5 Model Tuning

Though the weights, or parameters, of the links between different neurons are optimized by the model, there are so-called 'hyperparameters' that the user needs to define when training a neural network. Hyperparameters are key settings that are specified prior to training a neural network and have a significant impact on the model's performance and training time. Note that these hyperparameters are the reason why having a validation set is useful. Once the parameters have been estimated, we use another set of data over which we optimize the hyperparameters in order to avoid overfitting.

4.5.1 Batches and Epochs

Two important hyperparameters are batches and epochs. Batch size refers to the number of training examples used in each iteration of gradient descent. Epochs refer to the number of times the model is trained on the entire training dataset. During each epoch, the model iteratively processes all training examples in the dataset and updates its weights and biases accordingly. In our research we have tried a number of different values in order to optimize the performance of the model. In the end, there was no more improvement after 100 epochs which was thus the optimal number of epochs and the optimal batch size was 16 images.

4.5.2 Confidence Level

Other than the previously mentioned hyperparameters that have to do specifically with the neural network structure, important factors to be decided has to do with the actual output of the models. All of the models output a number of bounding boxes with a confidence level, between 0 and 1, that indicates how likely the model thinks it is that there is an object of interest, in this case a polyhouse, in the bounding box. Those bounding boxes are then turned into a prediction if the predicted confidence level is over a certain cutoff and discarded if the predicted confidence level is under that cutoff. This cutoff is an important factor as it determines how sure a model needs to be for it to make a prediction.

If we set the cutoff really low we will end up with a lot of predicted polyhouses, leading us to probably capture a lot of the actual polyhouses but also with more buildings that are not polyhouses. If we set the cutoff to be high, on the other hand, we will make fewer predictions and be very sure about the predictions that we do make, but there might be some more situations where polyhouses are not predicted as the confidence of the corresponding bounding box does not meet the cutoff.

As we are interested in the coverage of polyhouses in general of the whole state, both systematic overprediction and underprediction are problematic. Therefore, we try to choose the cutoff confidence value in such a way as to balance between these and we do so by trying over a few confidence values and capturing the accuracy over those points and then finally choosing the best. Note that this optimal cutoff confidence value can be different for each model, as they might have different optimal balances between positive and negative predictions. Also, we optimize this cutoff confidence value over the validation sets to make sure that the performance over the test set is not wrongly inflated.

4.6 Performance measures

After running the various methods that we described earlier in section 3, we need a way to compare their respective performances to decide which is better. We will use some very familiar performance measures here, though we note that in the context of object detection they have a slightly different interpretation than usual. As the models are outputting not just whether or not an object was detected but also the bounding boxes, we cannot say that the detection of an object was either correct or incorrect. Therefore we make use of the concept of Intersection over Union (IoU, Rezatofighi et al. 2019). Intersection over Union compares the areas of a bounding box of the prediction of the model, called p and the ground truth where we know that the object is, t. IoU can then be calculated by $\frac{t \cap p}{t \cup p}$, so the overlap between t and p divided by the area that's part of either t, p or both.

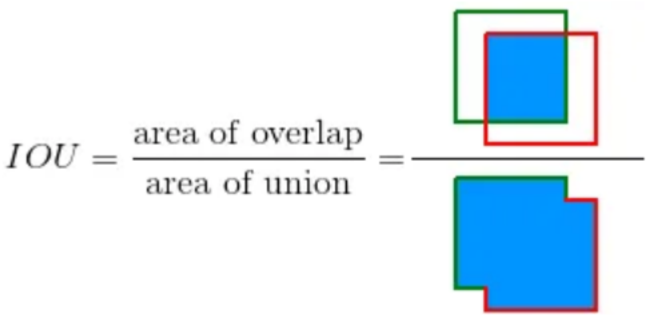
$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$

Figure 2: A visual depiction of the concept of Intercept over Union (IoU) (source: https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e)

This IoU can be calculated for every combination of prediction and ground truth and ranges between 0 and 1 where 0 indicate that there's no overlap between the two and 1 indicates perfect overlap. We have to define a value α that we use as cutoff for when we consider that a prediction is "correct". Anytime a prediction and a ground truth object have an IoU higher than this α we say that it has made a correct prediction and when it has a lower α we say that the prediction was incorrect. This choice of α has important implications for the final model that we use. This α basically controls how close a prediction has to match a true polyhouse for it to be considered correct. If it is set higher the model will need to be very sure about the location of a polyhouse for it to be considered correct, if it is lower then the model will accept a prediction even if it is not exactly correctly located.

To get the best possible predictions for each model, we would allow different values for IoU threshold and check at which value are the results high. We can define for each prediction whether it is correct or incorrect, we consider the following terms:

- True Positive (TP) - model correctly detects an item
- False Positive (FP) - model detects an item where there is none
- False Negative (FN) - model does not detect an item where there is one
- True Negative (TN) - this refers to the model correctly not detecting an item. As the model does not explicitly detect regions where there is no item this metric is not used, but it is mentioned here for completeness. Using these concepts, we will present the performance measures that we employ in this study, as outlined in more detail by Carion et al. 2020.

4.6.1 Precision

Precision assesses what fraction of the total number of predictions that the model makes actually detect an item. It can therefore be calculated by $\frac{TP}{TP+FP}$. This means that a high precision indicates that out of the times that the model indicates that there is a polyhouse, it is likely that there actually is one and that the model has a low probability of falsely predicting that there's a polyhouse. Precision is of particular interest when we want to be very sure of the correctness of any prediction we make before taking action. Note that precision will typically improve when the confidence threshold is set higher, as that will cause the models to make less predictions and only keep the ones it is really sure about.

4.6.2 Recall

Recall assesses what fraction of ground truths are recovered by the model. It can therefore be calculated by $\frac{TP}{TP+FN}$. This means that a high recall indicates that out of the times that there is a polyhouse, a lot of the time the model will be able to find it. Recall is of particular importance when we want to be very sure that we will detect all objects of interest. Note that recall will typically improve when the confidence threshold is set lower, as that will cause the model to make more predictions, making it more likely that actual objects of interest will be predicted.

4.6.3 F1 score

A model is considered good when it has both high precision and high recall. Precision can be artificially inflated by making a very low number of predictions, and recall by making a very high number of predictions. It is therefore important to consider both to make sure that the model recovers as many of the true items in the images as possible, but as little other things as possible. In order to allow for a trade-off between precision and recall, we also consider the so-called f1 score. The traditional f1 score is the harmonic mean of precision and recall. It can therefore be calculated by $\frac{2TP}{2TP+FP+FN} = 2 * \frac{precision * recall}{precision + recall}$. The traditional f1-score assigns the same weight to precision and recall, implying that predicting an item where there is none is exactly as bad as not predicting an item where there is one. There is also an adjusted f1-score, weighting the two to give one more relative importance than the other. In our research, however, we do not consider one worse than the other as they both contribute the same to the error in our mapping exercise for the coverage of polyhouses. As such, we will use the traditional f1-score

4.6.4 Mean average precision

Another way of coming up with a trade-off between recall and precision is the so-called mean average precision (mAP). For each of the earlier performance measures a cutoff values for the IoU is set and based on that the predictions are classified as FP, TP or FN. Based on that, the respective performance measures are calculated. When calculating the mAP, on the other hand, we deliberately vary this cutoff point to come up with a set of different values for the corresponding precision and recall. Note that as we increase the cutoff value, therefore requiring more certainty before predicting an object, precision will generally go up (as the higher scrutiny for predicting an object will lead to less predictions and less false positives) and recall will generally go down (as the higher scrutiny for predicting an object will lead to less predictions and more false negatives). Plotting these values of precision (or interpolated precision to avoid inconsistencies) and recall against each other, one obtains a graph.

The final mAP score is then the average precision across all values of recall, or the area under the curve in the graph. The 'mean average' in the name, which might seem excessive, refers to the fact that first one takes the average of precision across values of recall and then one takes the mean of these average precisions across different classes. However, we will only be predicting one class and therefore our average precision and mean average precision are the same.

4.7 Model Evaluation

Our goal is to get the model that gives the best bounding box using which we can get the number and coordinates of the polyhouses in the state of Telangana. Thus, this evaluation is three folds. First, to get the parameters in which the model works the best, then compare the three models we chose with their tuned parameters and finally use the model with the best performance to get the number and coordinates of the polyhouses in Ranga Reddy district.

Initially, we need to have an evaluation over different hyperparameters of each model and assign the IoU and confidence level thresholds to the model that gives us the highest performance. With performance we mean the F1 score, since that gives a balance between the percentage of polyhouses correctly identified and the percentage of polyhouses identified. Also, while tuning we need to make sure the optimal level of epochs and batch sizes has been chosen.

Next, we will compare how the three models performs when presented with the different train-test splits and their respective best threshold for IoU and confidence to get the model that works the best for our scenario. This would tell us what kind of model and data suits our scenarios the best.

Finally, we will use the chosen model to map the coverage of polyhouses across the Ranga Reddy district of Telangana. Thereafter, we will use the above mentioned performance measures: Precision, Recall and F1 score to calculate the performance of the chosen model in this case. This would be a rough representation of its performance in Telangana. However, it must be noted, this cannot tell the exact accuracy of the model in Telangana, for that, the model should be applied to the full state. For the ease of UNDP and their usage, we will output an excel file which tells the number of polyhouses and their coordinates per image and the images would have bounding boxes marked over what model thinks is a polyhouse.

It must be noted that first we get the model with highest bounding box accuracy and then use that to get the number of polyhouses and their coordinates. Thus, there can be models or techniques that directly give the number or coordinates with a higher accuracy but our research involves getting the bounding boxes first and then go for the coverage of polyhouses. The advantage of our approach is that we can use this bounding box data for multiple purposes such as getting the size of the polyhouses, delineate these from other buildings, identifying what are not polyhouses and other such tasks.

5 Results

In this chapter we will discuss a summary of the most important results. As mentioned before, our evaluation is three folds. In section 5.1, 5.2 and 5.3 we explain the performance of the YOLO, YOLO NAS and Faster RCNN models respectively and their inter comparison. In section 5.4 we choose the best model and in section 5.5 we apply this best model to get the number of polyhouses and their coordinates over the Ranga Reddy district of Telangana. Finally, we will discuss some relevant limitations in section 5.6.

However, before diving into the results it is very important to note the metrics relevance for certain cases. For datasets 1, 2 and 3 for all the 3 models, we do testing on polyhouse images and non polyhouse images separately as mentioned before. In non polyhouse testing case, precission is always 0 as there are no true positives with respect to polyhouse detection, and thus is the F1 score. Therefore, these metrics would not be regarded for datasets 1, 2 and 3 for non polyhouse testing within and across the models and are taken as 0. Moreover, recall is also not mathematically defined for these cases as there are no false negatives. But for comparison across different datasets and models, we will take recall as the percentage of incorrect predictions out of the total images and subtract it from 100%. For instance, if there are 100 non polyhouse images and model predicts 20 polyhouses, then the ”recall” will be 80%.

5.1 YOLO

In figures 5, 6, 7, 8 and 9 in the appendix, we see how the YOLO algorithm performs for different values of the confidence and IoU cutoff hyperparameters. Although the pattern might be difficult to interpret, the most important conclusion is probably that if we look at the y-axis the differences in F1 score when changing these values is extremely small. It is encouraging that the model seems to be robust and not very prone to changes in these cutoff values, and in the case of dataset 5 we even see no effect whatsoever.

5.1.1 Dataset 1-3

The results for the first three datasets can be found in the appendix in table 3, 4 and 5. When looking at the results over the first dataset, we see that the model does very well at detecting polyhouses in the training set and almost perfectly in the validation set. When we focus on performance in the test set, so the images from a different area, the performance is much worse. The model only detects about one fifth of all actual polyhouses correctly and predicts many polyhouses that are not there. It seems, therefore, that even though the model is tuned to accurately predict the data that it was fed to learn from, these findings do not generalize well to the new data, a clear case of overfitting. Moreover, not providing non polyhouse images has not allowed the model to learn on what is not a polyhouse, that is why in about 90% of the predictions model says there is a polyhouse, but there is not.

When looking at the second dataset, we see that it appears that the added images of the objects that look like polyhouses have affected the model. The performance over train and validation set are still very good, but we see that over the test set the model hardly predicts any polyhouses. This is the case both for images where there are polyhouses and where there are none. This suggests that from the transformed images of objects that look like polyhouses, the model has learned to be very selective in its decisions, labelling something as not a polyhouse when in doubt as it has learned form deceptive examples that even when something looks like a polyhouse it might not be one.

When looking at the third dataset, we see that performance get much better for the majority of performance measures that we consider. Performance over the train and validation set gets a little bit worse, probably reflecting the increased diversity in those sets. However, over the test set the model performs much better, almost as good as over the training and validation set. This highlights the importance of the similarity of data between the training, validation and the test set for the performance of the YOLO algorithm.

5.1.2 Dataset 4-5

The results of dataset 4 and 5 can be found in the appendix in table 6 and 7. Looking at datasets 4 and 5, we see the same pattern as we did for dataset 3. The mixing of images between the training, validation and test set means that the model is very adept at detecting polyhouses in all of them. We also see a good balance between precision and recall, indicating that the model neither under- nor overpredicts systematically. What is interesting is that the results between dataset 4 and 5 are extremely similar. That suggests that when the images are already randomly mixed between the train, validation and test set, and there are no systematic differences anymore, the impact of adding transformed images is limited for the YOLO algorithm.

5.1.3 Summary

In general, we find that the YOLO model is able to do a good job at detecting polyhouses in images for most values of confidence and IoU cutoffs, but it is extremely sensitive to the degree of similarity between the test and the train set. When they are similar, either by adding augmented images to the train and validation set or by randomly mixing the images between all sets, it performs very well with almost 90% over all performance measures. This suggests that if we are confident that the images that we want to map are similar to the images that were used to train the model, the YOLO algorithm is an excellent choice. It is good to be aware, though, that performance will get worse fast if the level of similarity decreases.

5.2 YOLO-NAS

In the appendix in figures 10, 11, 12, 13 and 14, we see the F1 score when we use different confidence and IoU cutoffs for the YOLO-NAS model. In this case, we see somewhat bigger differences between different settings than we did for the YOLO models. Specifically, we see that as the confidence cutoff increases, the F1 score goes up, suggesting that the model might overpredict if the cutoff is not set sufficiently high. The IoU threshold seemed to have much less influence over results.

5.2.1 Dataset 1-3

The results for the first three datasets can be found in the appendix in table 8, 9 and 10. Looking at the first dataset, we see that the in-sample performance over the train and validation set are again very good, indicating that the model does manage to map the given input to the output. However, the performance over the test set is significantly worse, showing that the difference in distribution between the data that was fed to the model and the new data that was used to test it caused the model real difficulties, indicating at least some degree of overfitting. The model has identified really less polyhouses compared to the regular YOLO algorithm. Thus,the accuracy of polyhouses detection has fallen but the model also did not make too many false predictions of polyhouses in non polyhouse images like the YOLO algorithm. Interestingly, the YOLO-NAS model does seem to systematically predict less polyhouses than the ordinary YOLO model, suggesting that the extra freedom in setting the hyperparameters has had a regularizing effect on the model.

When looking at the second dataset, we see that the performance over non-polyhouses has gotten better, though not as much as we saw for the regular YOLO model. Again, this makes sense as we have added augmented images of the non-polyhouses to the train and validation set. We see that the model again systematically underpredicts, though for this datasaet that was more to be expected as we explained before. The fact that results improve so much after adding some transformations indicates that perhaps the transformations and their original versions in the test set are very similar, possibly making it too easy for the model. In any case, the results of the first and second dataset combined, both for YOLO and YOLO-NAS, indicate that the similarity in distribution between the train/validation and the test set is very important with regards to the performance of the model. This suggests that if we were to use this model, it would be wise to use it on data that is similar to the data that was fed to the model in order to train it.

When looking at dataset 3, the results for both polyhouses and non-polyhouses in the test set are excellent, again underlining the importance of the similarity between the images and their augmented versions. This shows that when we can be sure that the distribution of the train/validation and the test data is similar this model can be expected to perform very well at detecting polyhouses.

5.2.2 Dataset 4-5

The results for dataset 4 and 5 can be found in full in the appendix in table 11 and 12. The results over the fourth dataset show that without using augmented images but simply mixing all available images before distributing them over the train, validation and test set also yields excellent results. The results are just a little bit worse then for dataset 3, which might be because in dataset 4 we test on combined polyhouse and non polyhouse scenario which is a more realistic scenario. Interestingly, the results over dataset 5 are again similar to those over dataset 4, indicating that when they're randomly distributed the augmented images do not really help the model to predict better. Also, over dataset 4 and 5 the model no longer seems to systematically underpredict nor overpredict.

5.2.3 Summary

All in all, the YOLO-NAS algorithm performs well, but is very dependent on how similar the train and test data are. However, we see that this dependence is probably just a little bit less than it was for the YOLO model, suggesting that the model has used its extra flexibility to be more prudent in its predictions when predicting something that looks new. This does seem dependent on the choice of the confidence cutoff parameter.

5.3 Faster-RCNN

As we saw before that the confidence and IoU cutoff do not really affect the results, we set the cutoffs at their optimal value for Faster R-CNN to see if it can match the YOLO models' performance.

5.3.1 Dataset 1-3

The results for dataset one to three can be found in tables 13, 14 and 15 in the appendix. For the faster R-CNN model, we also see that for the first dataset the model performs very well on the train set and very poor on the test set. It does, however, perform a bit worse on the validation set, indicating potential overfitting. Also we note that it does not seem very shy in predicting polyhouses in new images that it has not seen yet, overpredicting heavily on the images with polyhouses and predicting a relatively large number of polyhouses from images where there were none.

In the second dataset, the model does not seem to predict less polyhouses in the test set, even though it has learned from a lot of extra examples that seem deceptively like polyhouses but are not, information that the model is given to learn from. This is interesting and markedly different from what we saw for the YOLO models. Also interestingly, the model seems to underpredict the train and validation set. In the third dataset the performance over the test set where there were polyhouses gets really good, but it still predicts a lot of polyhouses when there are none.

5.3.2 Dataset 4-5

Table 16 and 17 in the appendix show the results for dataset four and five. Curiously, for dataset 4 and 5 the Faster R-CNN model systematically underpredicts over the test set. It also does so over the validation set, where performance does pick up in dataset 5 but is still not at the levels of the YOLO and the YOLO NAS. Another interesting feature is that between dataset 4 and 5, the results over the training set get significantly worse, suggesting that for the Faster R-CNN model adding the augmented images actually seemed to confuse the model.

5.3.3 Summary

In summary, the conclusion we can draw from the Faster R-CNN model is that there are several downsides to its performance. We have seen it both under and over predict depending on the configuration of the dataset and we have seen both situations where it does not do very well on the training set, indicating it has trouble mapping input to output, and over the validation and test sets, indicating that the model might be prone to some overfitting in certain situations.

5.4 Comparison

When comparing the three models, the first conclusion is that the Faster R-CNN model is probably not the best choice. Although all three models have their caveats, we see the Faster R-CNN model make an array of different types of errors that make it difficult to know how to interpret its predictions. We also find that over the whole, its performance in terms of all of our performance measures is not on the same level as the YOLO and the YOLO NAS, especially when the data in the train and test set are similar. That being said, the performance of both the YOLO and the YOLO NAS seem very dependent on this. When the train and test set are different, the YOLO models perform very poorly and have a very hard time detecting any polyhouses. The regular YOLO algorithm finds a few more than the YOLO-NAS algorithm, but also comes up with more wrong predictions. As the level of similarity increases, the performance of both models goes up spectacularly. This effect is stronger for the regular YOLO model, probably because less regularization goes on here. In summary, we prefer the regular YOLO model, as its performance over datasets where the train and test set are similar is simply better than YOLO-NAS. Additionally it detects about 90% of the polyhouses in the images that were provided to it when the size of dataset increases and is balanced for polyhouses and other buildings. Moreover, similarity of images is not a problem for us to create as our goal is to map the coverage of polyhouses in the Telangana state only.

5.5 Mapping

In the previous section, we saw that YOLO performed better than the rest and showed a decently high accuracy. Thus we choose the YOLO model for the mapping exercise. This mapping involves, as mentioned before, getting the number of polyhouses and the coordinates of the image where the polyhouses are according to the YOLO model and then calculating the performance metrics. We applied the yolo model to the images from the Ranga Reddy district and received the images that have polyhouses marked in them (if any). Each image contains the number of polyhouses detected and an excel file which tells us the coordinates and number of the polyhouse in each image and the total number of polyhouses in the area. The following are 4 examples of the excel output and corresponding images:

image Name	Number of Polyhouses	Coordinates
split_10_1.jpg	1	(152, 224)
split_10_11.jpg	0	
split_15_9.jpg	4	(21, 48), (196, 177), (123, 150), (75, 137)
split_11_6.jpg.jpg	2	(195, 256), (100, 234)

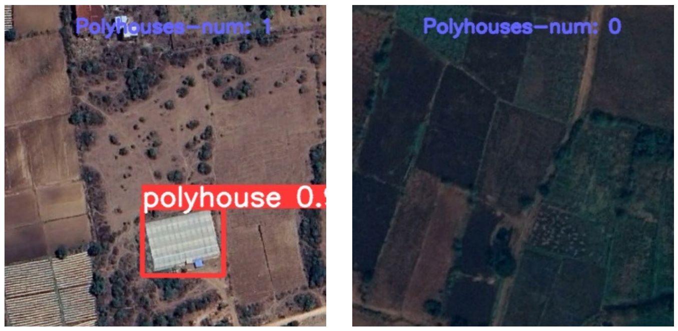

Figure 3: image named "split_10_1.jpg" on the left and "split_10_11.jpg"(right)

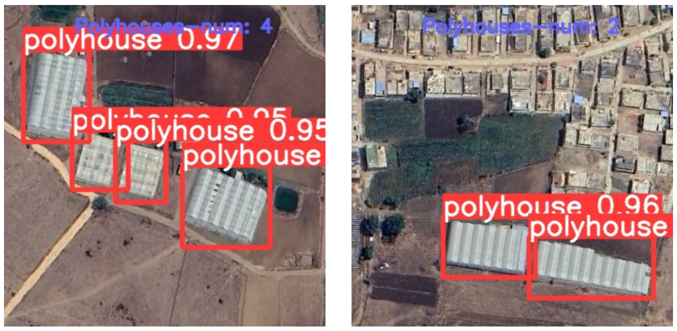

Figure 4: image named "split_15_9.jpg" (left) and "split_11_6.jpg" (right)

In the chosen region there were 164 polyhouses in total. YOLO model gave 172 bounding boxes, where 7 polyhouses were identified less(false negative) and model provided 15 extra polyhouses where there were there wasn't one (false positives), thus achieved a precision of 90.85%, recall of 95.73% and F1 score of 93.23%. This is in line with the score that we expect of the model when it is tested on data that looks similar to the data that it was trained on. This is encouraging, as it shows that when applied to new data from Telangana the model can be able to perform as it does in the best examples that we have considered before.

5.6 Limitations

In this chapter, we have compared the methods that we selected for the object detection task and chosen a method that has our preference. Although we believe that the method that we chose has a good performance, there are some caveats to it, most obviously how the data that was used to train the model has been collected and how similar the new task is that the model is going to be used for. However, we have not been able to find a model that credibly resolves these issues, so we encourage anyone using these models to be very aware of them. In practical terms, we recommend that anyone wanting to use our models to map polyhouse presence in a certain area first gathers a body of images from that area that are labelled for the presence of polyhouses. Training the model on this subsample of the area of interest is the easiest way to ensure that the data that the model is trained on is similar to the data that the model will then be applied to. Another limitation is that the images were labelled individually and represents random splits of the full surface of a state. That means that it is theoretically possible that a single polyhouse is split across two different images, recognized as a polyhouse in both and therefore counted twice. In this research we have taken manually identifying steps to eradicate this while data collection, and this is a step that could be taken in the future to fine tune the model.

Another limitation has to do with the volume of available data. As there is currently no labelled dataset of polyhouses freely available online that we can use to train the model, we had to manually engineer one. Though we think the results show this was a worthwhile exercise, having a large dataset ready would have obviously been better as the volume of the data that we used was naturally limited. Also the quality of the data is a concern. As we want to train a model to learn to detect polyhouses in the images we feed it, it is essential that the quality of the images permits the polyhouses to be clearly distinguished. In an earlier pass, we tried to use sentinel 2 data which clearly was not of sufficient quality (10m spatial resolution) for our models to work. For the results presented in section 5, we used Google Earth data, which seems to work much better because of higher quality satellite images (1m spatial resolution). For anyone using these models to detect polyhouses in a new context, it is therefore of utmost importance to feed the model high-quality images, as we have seen that the degree to which the models that we presented here learn from images of bad quality is very limited. There are two exceptions when the YOLO-NAS model might be preferred. If the training and the test are very different the performance of YOLO-NAS suffers a bit less than the ordinary YOLO algorithm and as such when one suspects that this might be the case the YOLO-NAS algorithm might be preferred. However, one has to take into account that in this situation also the YOLO-NAS algorithm performs worse than otherwise so its results have to be taken with care and it is still highly advisable to find more similar data to feed the model. The other scenario is when the cost of incorrectly predicting the absence or presence of a polyhouse is different. In some scenarios, we see that the regular YOLO algorithm predicts substantially more polyhouses than the YOLO-NAS algorithm. If for some specific reason one is looking for an algorithm that predicts only when it's very sure and does not want to risk ending up with a lot of predictions where there are no actual polyhouses, then in some situations the YOLO-NAS algorithm might be preferred.

6 Discussion and Conclusion

In this section we will go into some more detail about the meaning of the results and what they implicate for the research question and practical applications of our research. We begin in section 6.1 with a conclusion summarizing the main takeaways of this research. In section 6.2, we will discuss other possible angles to approach object recognition than the one that we employed in our research and in which cases these might be more suitable. Then, in section 6.3 we discuss some practical recommendations for UNDP when using this research and in section 6.4 we discuss some interesting avenues for further research.

6.1 Conclusion

We have assessed three different models to detect polyhouse coverage in the Indian state of Telangana. Being able to map the coverage of polyhouses in this area would be useful for UNDP as it would help with long-term agricultural planning. Specifically, we have compared three convolutional neural networks: YOLO, YOLO-NAS and Faster R-CNN. We believe that of the models we tested, the YOLO model is the best one to detect polyhouses. It does a pretty good job at distinguishing specifically the number of polyhouses that there are, which is an important focus for this research. However, there are some concerns about the ability of each of the models to deal with data that is not sufficiently similar to the data that it was trained on. That is, the results show evidence that when this data is not sufficiently similar, model performance suffers considerably. This should definitely be taken into account when applying the model we presented, and we hope that further research will shed more light on the relationship between data similarity and model performance.

6.2 Other applications

In this section we will briefly outline what other choices could be made that have not been applied in this research but that could easily be examined using the same type of models, in order to make our models applicable to other areas.

6.2.1 Size of polyhouses

In our research we have chosen to focus primarily on the amount of polyhouses in any given image. One could also, however, want to consider the size of the predicted polyhouses. In this case, it would be better to set a higher IoU threshold, indicating that the bounding boxes need to be a closer match for the prediction to be considered correct. This way, the required overlap for a prediction to be counted as such has to be higher, meaning the model favours predictions that have more area in common with an actual polyhouse and is therefore close to the true size. Also, when looking at size one could use different performance measures than the ones we have used in this research as the ones we have used do not particularly focus on the size of the polyhouses. One could, for example, calculate the performance measure that we already use but weight them by the size of the actual polyhouse when averaging to be sure to favour models that do particularly well at larger items. That means that in order to calculate each performance measure, prediction of bigger polyhouses will count heavier towards the average measure that is reported, for example counting polyhouses with a size above a certain threshold twice. One could also make the IoU threshold dependent on the size of the object to reflect the relative importance of small and large items. These extra performance measures could then be used to set hyperparameters and compare models in such a way that the model will be more specifically geared towards correctly predicting the size of polyhouses.

6.2.2 Exact Location of polyhouses

Another closely related priority could be not just to take the coordinates of the polyhouses in the respective image as a priority, but rather about their coordinates on the map of Telangana (or any other corresponding region). In this case one can use the exact models presented, but additionally convert the coordinates in the image to the coordinates of the corresponding other regions by using the gdal or tifffile libraries from python.

6.2.3 Detecting other objects

Moreover, the applicability of our models extends beyond the domain of polyhouse detection, encompassing diverse agricultural and infrastructural scenarios. By adjusting the input data, the same convolutional neural networks can be repurposed to identify a wide range of objects, thereby aiding various agricultural planning initiatives. To achieve this, our data collection methodology, detailed in section 4.1, can be adapted for different objects or regions. One would have to follow the same steps, but instead of looking for polyhouses any other object could be selected. With the model trained on altered data, it becomes proficient in recognizing agricultural infrastructure or relevant features using the same framework. Furthermore, fine-tuning the algorithm to distinguish between object categories entails expanding the data to encompass not only bounding boxes but also object types. This way, the model will learn not just the location but also the type of each object. While minimal hyperparameter adjustments are necessary, a larger dataset would be imperative. This prospective application may require more data acquisition efforts or automated datasets, but the model's ability to differentiate between diverse object categories underscores its versatility and utility.

6.3 Recommendations to the UNDP

The initial goal of this research was to provide UNDP with a tool that can be used to map the coverage of polyhouses in the state of Telangana, India. We have compared three models that, for various reasons, seemed good candidates for this object detection task. Although we have seen some concerns around the data, we feel that the YOLO model that we considered to be the best of the three does a good job at detecting polyhouses. The main takeaway for UNDP that has been reiterated at various points in this report, is the importance of training the model on data that looks similar to the data where polyhouses are to be detected. It is therefore not advised to use the existing model with all its parameters that are trained on the data that we had so far and blindly apply it to a new context, as it is a real danger that polyhouses look different in different geographical areas and the model will not do a good job. If UNDP decides it wants to move forward its efforts to map polyhouse coverage this way then we highly advice it to invest some effort in setting up a structural way to collect data on polyhouse coverage in areas of interest, that can then be used to train the model which will subsequently predict the rest of the state.

6.4 Further research

Although we have exposed some important limitations of the methods that should be taken into account, further research is still required to shed more light on their exact workings. Specifically, we have only used five configurations of the train, validation and test data. The results clearly indicate that some setups yield very bad results and some of them very good ones, but it is probably in between the two extremes where we can gather more meaningful insights. From our results it is clear that some scenario's are very difficult for all models and others are relatively easy for all models, but the situations where performance went up or down were similar for all three models. It would be interesting to think of more setups of the data that could be used to not just tell when the models perform well or poorly but also where performance of the respective models will differ more.

Additionally, we have already given some indications as to what type of detection one could also be interested in, aside from what we have done in our research. All of them rely in principle on the model having to predict accurate bounding boxes, but it would be interesting to see how subsequent priorities and steps to be taken accordingly affect the results. Finally, the models to be used and context to use them, both in terms of which objects to detect and where to detect them, are almost infinite. We have only examined a very limited part of India using only the polyhouse images that we could find, but off course the question of how to plan the agricultural future of India is far from complete there and is probably a topic that will need to go through many more iterations of thinking. Distinguishing polyhouses by type or by performance, predicting not just where polyhouses are now but where they could potentially be added and doing so for a geographical space far beyond what we have looked at all seem like potential avenues that could help organizations plan for the future more effectively. It is our hope that our research will have given a first impulse to show what is possible, but that it will be only the beginning of a far more comprehensive body of work to ensure the food security of the people of India for years to come.

7 Appendix

Results

YOLO

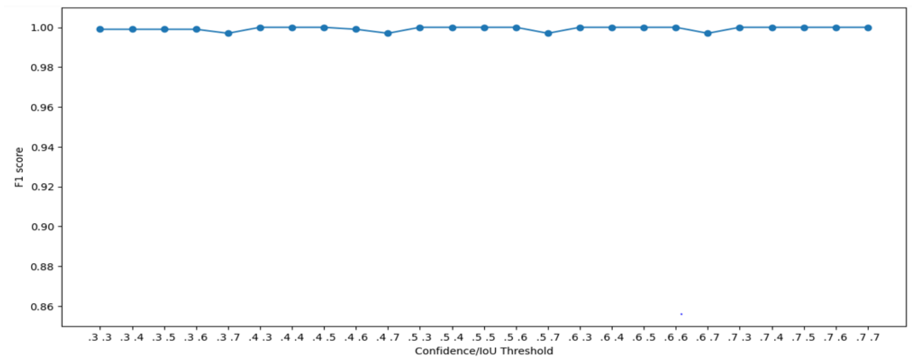

Figure 5: Performance of YOLO over dataset 1 with different confidence and IoU cutoffs

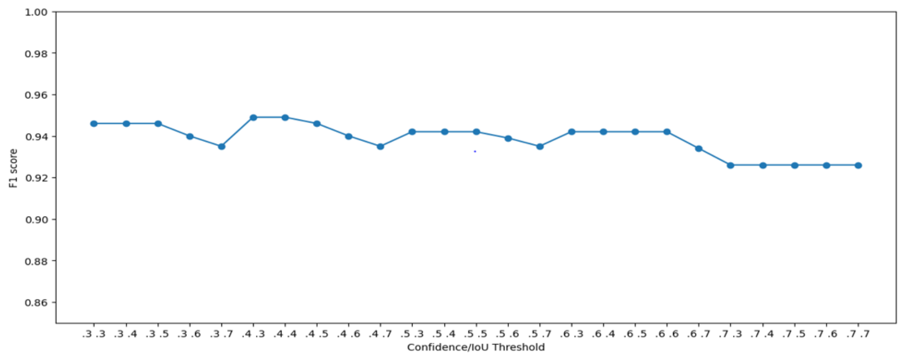

Figure 6: Performance of YOLO over dataset 2 with different confidence and IoU cutoffs

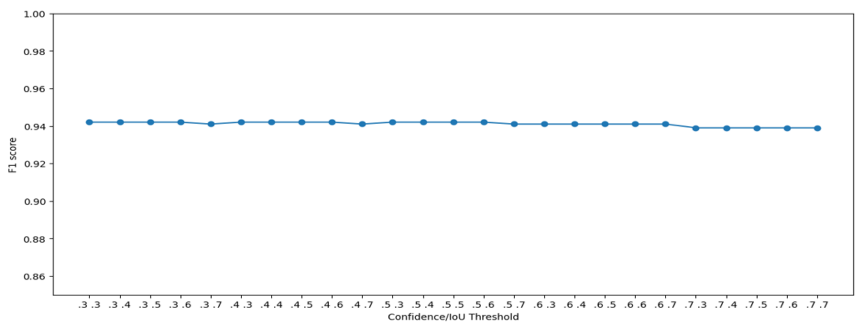

Figure 7: Performance of YOLO over dataset 3 with different confidence and IoU cutoffs

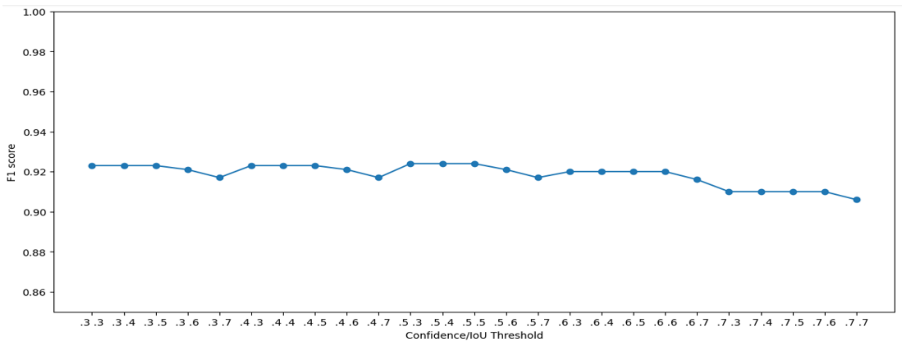

Figure 8: Performance of YOLO over dataset 4 with different confidence and IoU cutoffs

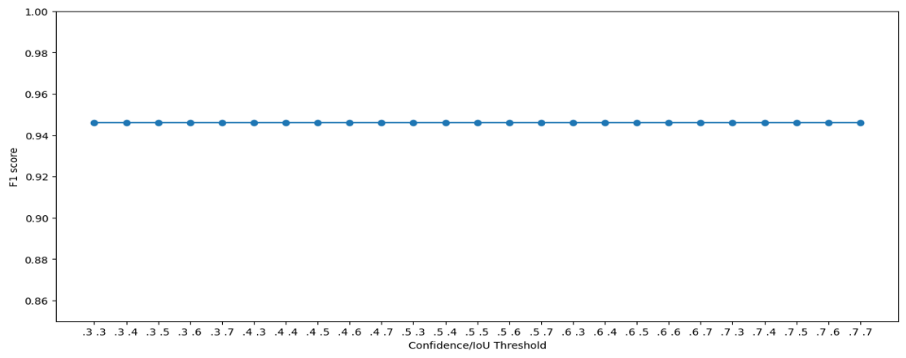

Figure 9: Performance of YOLO over dataset 5 with different confidence and IoU cutoffs

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.992	0.9855	0.983	0.988	153	480	482
Validation	0.995	1	1	1	17	44	44
Testing (polyhouses)	0.08925	0.14425	0.1185	0.1885	100	216	357
Testing (non-polyhouses)		0	0	0.19	100	0	81

Table 3: Results YOLO dataset 1

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.995	0.998	0.996	1	208	425	427
Validation	0.957	0.949	0.959	0.939	52	99	97
Testing (polyhouses)	0.084	0.0084	0.1665	0.00431	100	216	3
Testing (non-polyhouses)		0	0	0.98	100	0	2

Table 4: Results YOLO dataset 2

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.995	0.997	0.994	1	360	624	628
Validation	0.954	0.94	0.954	0.93	90	178	177
Testing (polyhouses)	0.943	0.934	0.948	0.921	100	216	210
Testing (non-polyhouses)		0	0	0.88	100	0	12

Table 5: Results YOLO dataset 3

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.995	0.9965	0.995	0.998	236	447	448
Validation	0.952	0.9215	0.92	0.923	59	112	113
Testing	0.942	0.917	0.926	0.908	74	178	175

Table 6: Results YOLO dataset 4

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.995	0.998	0.998	0.998	586	811	812
Validation	0.97	0.946	0.95	0.942	146	191	189
Testing	0.931	0.906	0.913	0.899	74	178	174

Table 7: Results YOLO dataset 5

YOLO-NAS

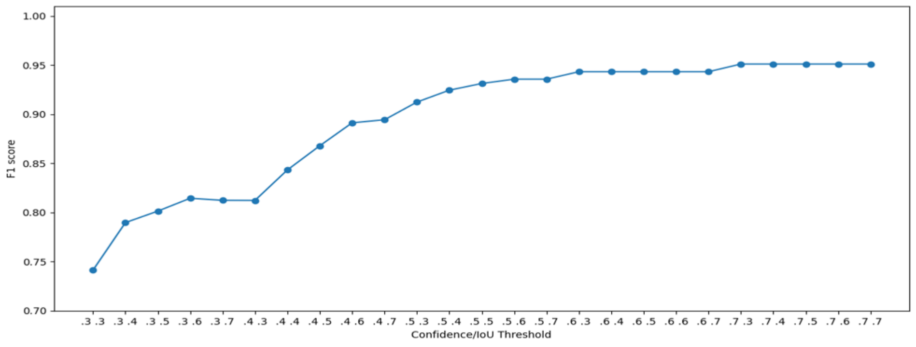

Figure 10: Performance of YOLO-NAS over dataset 1 with different confidence and IoU cutoffs

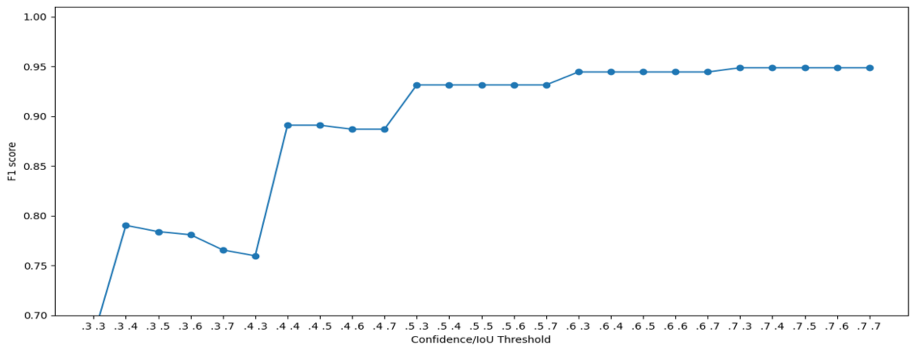

Figure 11: Performance of YOLO-NAS over dataset 2 with different confidence and IoU cutoffs

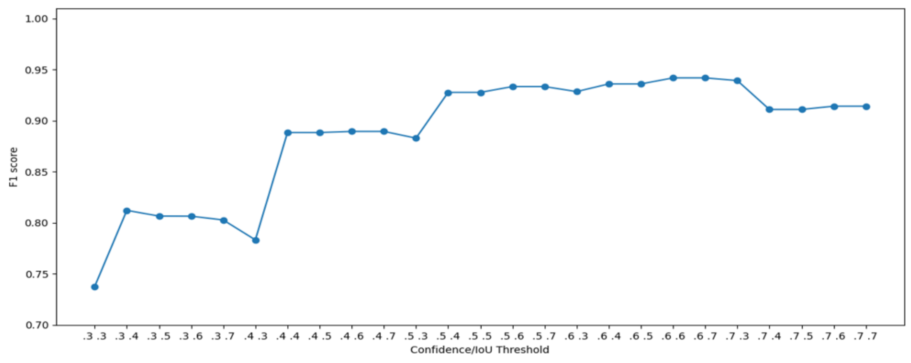

Figure 12: Performance of YOLO-NAS over dataset 3 with different confidence and IoU cutoffs

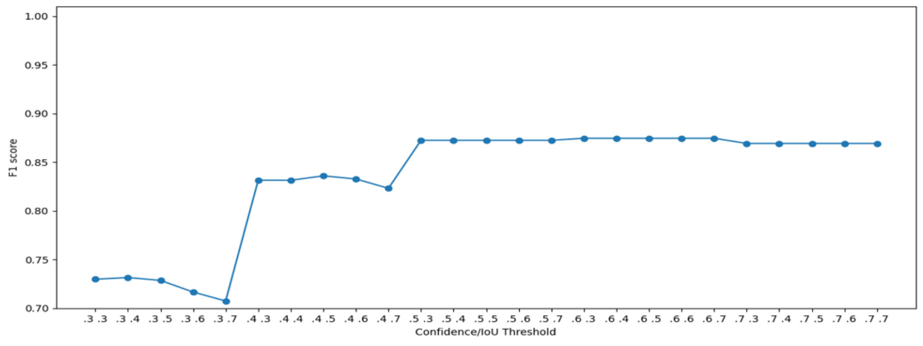

Figure 13: Performance of YOLO-NAS over dataset 4 with different confidence and IoU cutoffs

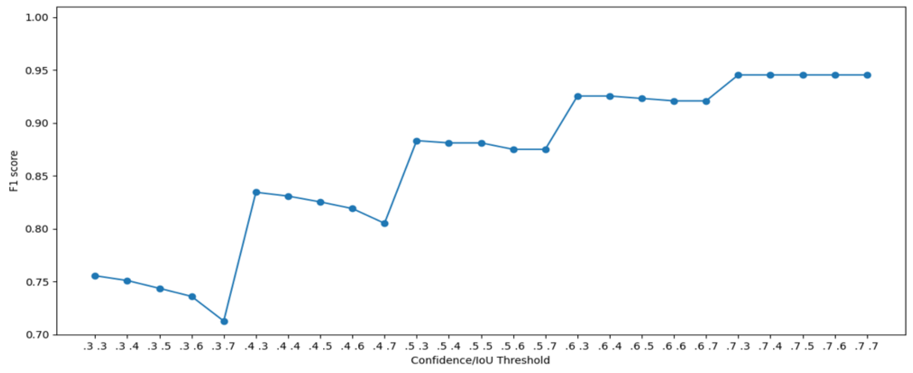

Figure 14: Performance of YOLO-NAS over dataset 5 with different confidence and IoU cutoffs

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.899	0.8514	0.9496	0.7717	153	480	391
Validation	0.9947	0.951	0.9062	1	17	44	95
Testing (polyhouses)	0.1135	0.0578	0.3164	0.0318	100	216	22
Testing (non-polyhouses)		0	0	0.53	100	0	47

Table 8: Results YOLO-NAS dataset 1

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.9162	0.7972	0.9795	0.6721	208	425	292
Validation	0.9705	0.9444	0.9495	0.9394	52	99	98
Testing (polyhouses)	0.2109	0.1106	0.6842	0.0602	100	216	19
Testing (non-polyhouses)		0	0	0.67	100	0	33

Table 9: Results YOLO-NAS dataset 2

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.8606	0.7896	0.9661	0.6676	360	624	432
Validation	0.9676	0.9503	0.9412	0.9596	90	178	182
Testing (polyhouses)	0.9572	0.907	0.8723	0.9444	100	216	234
Testing (non-polyhouses)		0	0	0.63	100	0	37

Table 10: Results YOLO-NAS dataset 3

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.8947	0.7988	0.975	0.6766	236	447	312
Validation	0.9877	0.9569	0.925	0.9911	59	112	120
Testing	0.9453	0.9091	0.9195	0.8989	74	178	174

Table 11: Results YOLO-NAS dataset 4

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.8934	0.8474	0.9485	0.7659	586	811	656
Validation	0.9703	0.9173	0.8798	0.9581	146	191	208
Testing	0.9606	0.9059	0.9016	0.9101	74	178	180

Table 12: Results YOLO-NAS dataset 5

Faster RCNN

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.996	0.978	1	0.957	153	480	460
Validation	0.591	0.7688	0.945	0.648	17	44	31
Testing (polyhouses)	0.0315	0.164	0.137	0.203	100	216	322
Testing (non-polyhouses)		0	0	0.24	100	0	76

Table 13: Results Faster R-CNN dataset 1

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.996	0.981	1	0.962	208	425	409
Validation	0.621	0.7813	0.953	0.662	52	99	69
Testing (polyhouses)	0.0512	0.267	0.211	0.365	100	216	374
Testing (non-polyhouses)		0	0	0.31	100	0	69

Table 14: Results Faster R-CNN dataset 2

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.973	0.975	0.99	0.96	360	624	605
Validation	0.656	0.8066	0.957	0.697	90	178	130
Testing (polyhouses)	0.968	0.9884	1	0.977	100	216	211
Testing (non-polyhouses)		0	0	0.13	100	0	87

Table 15: Results Faster R-CNN dataset 3

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.9934	0.998	1	0.996	236	447	445
Validation	0.53	0.73	0.886	0.621	59	112	79
Testing	0.6226	0.805	0.929	0.71	74	178	137

Table 16: Results Faster R-CNN dataset 4

	MAP	F1 score	Precision	Recall	Images	True	Pred
Training	0.714	0.757	0.774	0.741	586	811	777
Validation	0.8249	0.9	0.935	0.865	146	191	177
Testing	0.64	0.81	0.939	0.712	74	178	135

Table 17: Results Faster R-CNN dataset 5

Python Code

```
#Mount your drive in order to trian and save model on it
#mount drive
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')

# list the contents of /mydrive
!ls /mydrive

#Navigate to /mydrive/yolov4
# %cd /mydrive/

#Clone github
%cd /content/
!git clone https://github.com/ultralytics/yolov5  #
    clone yolov5 repository
%cd yolov5
%pip install -qr requirements.txt  # install

import torch
import utils
display = utils.notebook_init()  # checks
import torch # YOLOv5 implemented using pytorch
from IPython.display import Image #this is to render
    predictions


import os
from random import choice
import shutil

!unrar x
    "/content/gdrive/MyDrive/polyhouses/third_3_tables/dataset3.rar"
     "/content/"

#make dataset.yml files and put it in yolov5/data/
    directory
train in 100 epochs for dataset1

#train in 100 epochs for dataset2
%cd /content/gdrive/MyDrive/polyhouses/yolov5
!python train.py --img 608 --batch 16 --epochs 100
    --data dataset.yaml --weights yolov5l.pt --cache
```

```
#View test samples
Image(filename='runs/train/exp2/val_batch0_pred.jpg',
    width=1000)

#Trainset results for dataset 1
%cd /content/gdrive/MyDrive/polyhouses/yolov5
!python val.py --data dataset.yaml --weights
    runs/train/exp3/weights/best.pt --conf-thres .5
    --task train

#Validation results for dataset 1
!python val.py --data dataset.yaml --weights
    runs/train/exp3/weights/best.pt --conf-thres .5
    --task val

#Test results for datset1
!python val.py --data dataset.yaml--weights
    runs/train/exp3/weights/best.pt --conf-thres .5
    --task val

!python val.py --data dataset.yaml--weights
    runs/train/exp4/weights/best.pt --conf-thres .5
    --task train

!python val.py --data dataset.yaml --weights
    runs/train/exp4/weights/best.pt --conf-thres .5
    --task val

!python val.py --data dataset.yaml --weights
    runs/train/exp4/weights/best.pt -conf-thres .5
    --iou_thres .4 --task val

import numpy as np

w=0
h = 12
rl = 5
rh = 6
img_crop= np.zeros((int(2 * rh), int(0)))
img_np= np.zeros((int(20 * rl), int(20 * rh)))
y = -20
x = 50
if w!=2*rl:
            img_crop__ = np.zeros((int(2 * rh), int(2 *
               rl)))
            diff__ = 2*rl-w
            if y>rl:
                img_crop__[:,img_crop__.shape[1]-diff__:]
                    = img_np[x - rh:x + rh, 0:diff__]
                print(img_crop.shape)
                img_crop__[:,:w] = img_crop
            else:
                img_crop__[:, :diff__] = img_np[x - rh:x
                    + rh, img_np.shape[1]-diff__:]
                img_crop__[:,img_crop__.shape[1]-w:] =
                    img_crop

            print(img_crop__.shape, img_crop.shape)
%cd /content/gdrive/MyDrive/polyhouses/yolov5
!rm -rf
    /content/gdrive/MyDrive/polyhouses/datasets/test_dataset/all_test_poly
!cp
    /content/gdrive/MyDrive/polyhouses/datasets/test_dataset
/images_poly_labeled/*
    /content/gdrive/MyDrive/polyhouses/datasets/test_dataset/all_test_poly
!cp
    /content/gdrive/MyDrive/polyhouses/datasets/test_dataset/images_50_new/*
    /content/gdrive/MyDrive/polyhouses/datasets/test_dataset/all_test_poly
import os
img =
    os.listdir("/content/gdrive/MyDrive/polyhouses/datasets
/test_dataset/all_test_poly")
len(img)

#Validation on different condidence
!python val.py --data dataset.yaml --weights
    /content/gdrive/MyDrive/polyhouses/
yolov5/runs/train/first/best.pt --conf-thres .5
    --iou-thres .4 --task val
#YOLO NAS
!pip install super-gradients==3.1.0
!pip install imutils

!pip install pytube --upgrade
from super_gradients.training import Trainer

CHECKPOINT_DIR = 'checkpoints4'
trainer =
    Trainer(experiment_name='my_first_yolonas_run',
    ckpt_root_dir=CHECKPOINT_DIR)
from super_gradients.training import dataloaders
from super_gradients.training.dataloaders.dataloaders
    import coco_detection_yolo_format_train,
    coco_detection_yolo_format_val
dataset_params = {
    'data_dir':'/content/gdrive/MyDrive/polyhouses/dataset2',
    'train_images_dir':'dataset2_train',
    'train_labels_dir':'dataset2_train',
    'val_images_dir':'dataset2_val',
    'val_labels_dir':'dataset2_val',

    'classes': ['polyhouse']
}
from IPython.display import clear_output

train_data = coco_detection_yolo_format_train(
    dataset_params={
        'data_dir': dataset_params['data_dir'],
        'images_dir': dataset_params['train_images_dir'],
        'labels_dir': dataset_params['train_labels_dir'],
        'classes': dataset_params['classes']
    },
    dataloader_params={
        'batch_size':16,
        'num_workers':2
    }
)

val_data = coco_detection_yolo_format_val(
    dataset_params={
        'data_dir': dataset_params['data_dir'],
        'images_dir': dataset_params['val_images_dir'],
        'labels_dir': dataset_params['val_labels_dir'],
        'classes': dataset_params['classes']
    },
    dataloader_params={
        'batch_size':16,
        'num_workers':2
    }
)

train_data.dataset.plot()

rom super_gradients.training import models
model = models.get('yolo_nas_m',
                   num_classes=len(dataset_params['classes']),
                   pretrained_weights="coco"
```

```
                    )
from super_gradients.training.losses import PPYoloELoss
from super_gradients.training.metrics import
    DetectionMetrics_050
from
    super_gradients.training.models.detection_models.pp_yolo_e
    import PPYoloEPostPredictionCallback

#Training for dataset 1
trainer.train(model=model,
              training_params=train_params,
              train_loader=train_data,
              valid_loader=val_data)


#Faster RCNN
import os
import glob
import xml.etree.ElementTree as ET
import pandas as pd
import tensorflow as tf
print(tf.__version__)
from google.colab import drive
drive.mount('/content/gdrive')


!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive
%cd /content/
# clone the tensorflow models on the colab cloud vm
!git clone --q https://github.com/tensorflow/models.git

#navigate to /models/research folder to compile protos
%cd models/research

# Compile protos.
!protoc object_detection/protos/*.proto --python_out=.

# Install TensorFlow Object Detection API.
!cp object_detection/packages/tf2/setup.py .
!python -m pip install .

%cd /mydrive/customTF2/data/

# unzip the datasets and their contents so that they are
    now in /mydrive/customTF2/data/ folder
```

```
!unzip /mydrive/customTF2/images.zip -d .
!unzip /mydrive/customTF2/annotations.zip -d .

#creating two dir for training and testing
!mkdir test_labels train_labels

# lists the files inside 'annotations' in a random order
    (not really random, by their hash value instead)
# Moves the first 274/1370 labels (20% of the labels) to
    the testing dir: `test_labels`
!ls annotations/* | sort -R | head -274 | xargs -I{} mv
    {} test_labels/


# Moves the rest of the labels ( 1096 labels ) to the
    training dir: `train_labels`
!ls annotations/* | xargs -I{} mv {} train_labels/

#adjusted from:
    https://github.com/datitran/raccoon_dataset
def xml_to_csv(path):
  classes_names = []
  xml_list = []

  for xml_file in glob.glob(path + '/*.xml'):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    for member in root.findall('object'):
      classes_names.append(member[0].text)
      value = (root.find('filename').text  ,
               int(root.find('size')[0].text),
               int(root.find('size')[1].text),
               member[0].text,
               int(member[4][0].text),
               int(member[4][1].text),
               int(member[4][2].text),
               int(member[4][3].text))
      xml_list.append(value)
  column_name = ['filename', 'width', 'height', 'class',
     'xmin', 'ymin', 'xmax', 'ymax']
  xml_df = pd.DataFrame(xml_list, columns=column_name)
  classes_names = list(set(classes_names))
  classes_names.sort()
  return xml_df, classes_names

for label_path in ['train_labels', 'test_labels']:
  image_path = os.path.join(os.getcwd(), label_path)
  xml_df, classes = xml_to_csv(label_path)
  xml_df.to_csv(f'{label_path}.csv', index=None)
  print(f'Successfully converted {label_path} xml to 
     csv.')

label_map_path = os.path.join("label_map.pbtxt")
pbtxt_content = ""

for i, class_name in enumerate(classes):
    pbtxt_content = (
        pbtxt_content
        + "item {{\n    id: {0}\n    name: 
            '{1}'\n}}\n\n".format(i + 1, class_name)
    )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
    f.write(pbtxt_content)
    print('Successfully created label_map.pbtxt ')

%cd /content/gdrive/MyDrive/polyhouses/
third_dataset
!python generate_tfrecord.py
   /content/gdrive/MyDrive/polyhouses/
fastrcnn/test_tf2_640_19_6_23_2.csv  label_map.pbtxt
   /content/gdrive/MyDrive/polyhouses/fastrcnn/test_dataset2/
   /content/gdrive/MyDrive/polyhouses/fastrcnn/test_dataset2.record
%cd /content/
!wget
   http://download.tensorflow.org/models/object_detection/
tf2/20200711/faster_rcnn_resnet50_v1_640x640_coco17_tpu-8.tar.gz
!tar -xzvf
   faster_rcnn_resnet50_v1_640x640_coco17_tpu-8.tar.gz

!cp
   /content/models/research/object_detection/configs/tf2/
ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config
   /mydrive/customTF2/data

%cd /content/models/research/object_detection

"""
PIPELINE_CONFIG_PATH=path/to/pipeline.config
MODEL_DIR=path to training checkpoints directory
NUM_TRAIN_STEPS=50000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1
```

```
python model_main_tf2.py -- \
  --model_dir=$MODEL_DIR
      --num_train_steps=$NUM_TRAIN_STEPS \
  --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES
      \
  --pipeline_config_path=$PIPELINE_CONFIG_PATH \
  --alsologtostderr
"""

!python model_main_tf2.py
    --pipeline_config_path=/content/gdrive/MyDrive/
polyhouses/second_dataset/
faster_rcnn_resnet50_v1_640x640_coco17_tpu-8.config
    --model_dir=/content/training_dataset1
    --alsologtostderr

%cd /content/models/research/object_detection
"""
PIPELINE_CONFIG_PATH=path/to/pipeline.config
MODEL_DIR=path to training checkpoints directory
CHECKPOINT_DIR=${MODEL_DIR}
NUM_TRAIN_STEPS=50000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1

python model_main_tf2.py -- \
  --model_dir=$MODEL_DIR
      --num_train_steps=$NUM_TRAIN_STEPS \
  --checkpoint_dir=${CHECKPOINT_DIR} \
  --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES
      \
  --pipeline_config_path=$PIPELINE_CONFIG_PATH \
  --alsologtostderr
"""

!python model_main_tf2.py
    --pipeline_config_path=/content/gdrive/MyDrive/polyhouses/
/second_dataset/-8.config --model_dir=/
content/gdrive/MyDrive/polyhouses/
fastrcnn/training_dataset1/
    --checkpoint_dir=/content/gdrive/MyDrive/polyhouses/
fastrcnn/training_dataset1/ --alsologtostderr

%cd /content/models/research/object_detection

"""
```

```
PIPELINE_CONFIG_PATH=path/to/pipeline.config
MODEL_DIR=path to training checkpoints directory
CHECKPOINT_DIR=${MODEL_DIR}
NUM_TRAIN_STEPS=50000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1

python model_main_tf2.py -- \
  --model_dir
  =$MODEL_DIR --num_train_steps=$NUM_TRAIN_STEPS \
  --checkpoint_dir=${CHECKPOINT_DIR} \
  --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES
      \
  --pipeline_
   config_path=$PIPELINE_CONFIG_PATH \
  --alsologtostderr
"""

!python model_main_tf2.py
    --pipeline_config_path=/content/gdrive/MyDrive/polyhouses/
/second_dataset/
faster_rcnn_resnet50_v1_640x640_coco17_tpu-8.config
    --model_dir=/content/gdrive/MyDrive/polyhouses/
fastrcnn/training_dataset1/ --
checkpoint_dir=/content/gdrive/Mydrive/
polyhouses/fastrcnn/training_dataset1/ --alsologtostderr
```

References

Adimalla, Narsimha et al. (2020). “Appraisal of groundwater quality for drinking and irrigation purposes in Central Telangana, India”. In: *Groundwater for Sustainable Development* 10, p. 100334.

Agovino, Massimiliano et al. (2019). “Agriculture, climate change and sustainability: The case of EU-28”. In: *Ecological Indicators* 105, pp. 525–543.

Alzubaidi, Laith et al. (2021). “Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions”. In: *Journal of big Data* 8, pp. 1–74.

Asokan, Anju et al. (2020). “Image processing techniques for analysis of satellite images for historical maps classification—An overview”. In: *Applied Sciences* 10.12, p. 4207.

Balcik, Filiz Bektas, Gizem Senel, and Cigdem Goksel (2020). “Object-based classification of greenhouses using Sentinel-2 MSI and SPOT-7 images: A case study from Anamur (Mersin), Turkey”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13, pp. 2769–2777.

Baxt, William G (1995). “Application of artificial neural networks to clinical medicine”. In: *The lancet* 346.8983, pp. 1135–1138.

Bishop, Chris M (1994). “Neural networks and their applications”. In: *Review of scientific instruments* 65.6, pp. 1803–1832.

Bonnett, Raymond and JB Campbell (2002). *Introduction to remote sensing.*

Boyabatlı, Onur, Javad Nasiry, and Yangfang Zhou (2019). “Crop planning in sustainable agriculture: Dynamic farmland allocation in the presence of crop rotation benefits”. In: *Management Science* 65.5, pp. 2060–2076.

Carion, Nicolas et al. (2020). “End-to-end object detection with transformers”. In: *European conference on computer vision.* Springer, pp. 213–229.

Chen, Liang-Chieh et al. (2017). “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.4, pp. 834–848.

Dave, Chintan P, Rahul Joshi, and SS Srivastava (2015). “A survey on geometric correction of satellite imagery”. In: *International Journal of Computer Applications* 116.12.

De Wrachien, Daniele (2003). “Land use planning: a key to sustainable agriculture”. In: *Conservation agriculture: environment, farmers experiences, innovations, socio-economy, policy*, pp. 471–483.

Girshick, Ross (2015). “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

Gopani, Tarak (2020). *Grocery Item Detection using TensorFlow Object Detection API.* URL: `https://tarak-gopani.medium.com/grocery-item-detection-using-tensorflow-object-detection-api-1581fb5df6d6`.

He, Kaiming et al. (2016). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hong, Ruikai et al. (2023). “Multitemporal greenhouse mapping for high-resolution remote sensing imagery based on an improved YOLOX”. In: *Computers and Electronics in Agriculture* 206, p. 107689.

Howden, S Mark et al. (2007). “Adapting agriculture to climate change”. In: *Proceedings of the national academy of sciences* 104.50, pp. 19691–19696.

Huang, Wei et al. (2007). “Neural networks in finance and economics forecasting”. In: *International Journal of Information Technology & Decision Making* 6.01, pp. 113–140.

Huang, Yongbo et al. (2023). “Remote Sensing Object Counting Through Regression Ensembles and Learning to Rank”. In: *IEEE Transactions on Geoscience and Remote Sensing.*

Inc., Roboflow (2023). *Roboflow.* URL: `https://roboflow.com/`.

Jacobsen, Karsten (2005). “High resolution satellite imaging systems-an overview”. In: *Photogrammetrie Fernerkundung Geoinformation* 2005.6, p. 487.

Jean, Neal et al. (2016). “Combining satellite imagery and machine learning to predict poverty”. In: *Science* 353.6301, pp. 790–794.

Jonnala, Prathiba and GSR Sathyanarayana (2015). “A wireless sensor network for polyhouse cultivation using zigbee technology”. In: *ARPN Journal of Engineering and Applied Sciences* 10.10.

Jonnala, Prathiba and Sivaji Satrasupalli (2013). “Semi-Automated Polyhouse Cultivation Using LabVIEW”. In: *IJCSBI. ORG* 20118.

Kadiyala, Suneetha et al. (2014). “Agriculture and nutrition in India: mapping evidence to pathways”. In: *Annals of the New York academy of sciences* 1331.1, pp. 43–56.

Kasar, Manisha M, Debnath Bhattacharyya, and TH Kim (2016). “Face recognition using neural network: a review”. In: *International Journal of Security and Its Applications* 10.3, pp. 81–100.

Kerr, John M et al. (1996). *Sustainable development of rainfed agriculture in India.* Tech. rep. International Food Policy Research Institute (IFPRI).

Kim, Jeong-ah, Ju-Yeong Sung, and Se-ho Park (2020). “Comparison of Faster-RCNN, YOLO, and SSD for real-time vehicle type recognition”. In: *2020 IEEE international conference on consumer electronics-Asia (ICCE-Asia).* IEEE, pp. 1–4.

Kumar, Parveen, RS Chauhan, and RK Grover (2016). “Economics analysis of tomato cultivation under poly house and open field conditions in Haryana, India”. In: *Journal of Applied and Natural Science* 8.2, pp. 846–848.

Kurian, NJ (2008). “Inclusive growth in India: Agriculture, poverty and human development”. In: *Social Change* 38.2, pp. 340–342.

LeCun, Yann et al. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Li, Min et al. (2020). “Agricultural greenhouses detection in high-resolution satellite images based on convolutional neural networks: Comparison of faster R-CNN, YOLO v3 and SSD”. In: *Sensors* 20.17, p. 4938.

LLC, Google (2023). *tensorflow/models: Models and examples built with TensorFlow.* `https://github.com/tensorflow/models`.

Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2015). “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

Lovell, Sarah Taylor (2010). “Multifunctional urban agriculture for sustainable land use planning in the United States”. In: *Sustainability* 2.8, pp. 2499–2522.

Mathieson, Allan et al. (2009). “Satellite imaging to monitor CO2 movement at Krechba, Algeria”. In: *Energy Procedia* 1.1, pp. 2201–2209.

Mirabella, Orazio and Michele Brischetto (2010). “A hybrid wired/wireless networking infrastructure for greenhouse management”. In: *IEEE transactions on instrumentation and measurement* 60.2, pp. 398–407.

Murthy, D Sreenivasa et al. (2009). “Economic feasibility of vegetable production under polyhouse: A case study of capsicum and tomato”. In: *Journal of horticultural sciences* 4.2, pp. 148–152.

Raja, R et al. (1997). “Energy planning and optimization model for rural development—A case of sustainable agriculture”. In: *International Journal of Energy Research* 21.6, pp. 527–547.

Rath, Sovit (2023). *Train YOLO-NAS on Custom Dataset*. LearnOpenCV tutorial. URL: `https://learnopencv.com/train-yolo-nas-on-custom-dataset/`.

Redmon, Joseph et al. (2016). “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Reganold, John P, Robert I Papendick, and James F Parr (1990). “Sustainable agriculture”. In: *Scientific American* 262.6, pp. 112–121.

Ren, Shaoqing et al. (2015). “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems* 28.

Rezatofighi, Hamid et al. (2019). “Generalized intersection over union: A metric and a loss for bounding box regression”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 658–666.

Sharma, HK, HS Gaur, Balraj Singh, et al. (2007). “Nemic population dynamics in hybrid tomato, sweet pepper and hybrid cucumber under polyhouse cultivation”. In: *Indian Journal of Nematology* 37.2, pp. 161–164.

Skalski, Piotr (2023). *YOLO-NAS: How to Train on Custom Dataset*. Roboflow blog post. URL: `https://blog.roboflow.com/yolo-nas-how-to-train-on-custom-dataset/`.

Smith, CS and GT McDonald (1998). “Assessing the sustainability of agriculture at the planning stage”. In: *Journal of environmental management* 52.1, pp. 15–37.

Srivastava, Nitish et al. (2014). “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Sun, Haoran et al. (2021). “Mapping plastic greenhouses with two-temporal sentinel-2 images and 1d-cnn deep learning”. In: *Remote Sensing* 13.14, p. 2820.

Techzizou (2022). *Build Android App for Custom Object Detection (TF 2.x)*. URL: https://medium.com/geekculture/build-android-app-for-custom-object-detection-tf-2-x-53904a08cfa2.

Tellman, B et al. (2021). "Satellite imaging reveals increased proportion of population exposed to floods". In: *Nature* 596.7870, pp. 80–86.

Tran, Dat (2023). *datitran/raccoon_dataset: Annotated images and annotations for raccoon detection.* https://github.com/datitran/raccoon_dataset.

Ultralytics and Glenn Jocher (2023). *ultralytics/yolov5: YOLOv5 in PyTorch.* https://github.com/ultralytics/yolov5.

Vakulabharanam, Vamsi (2004). "Agricultural growth and irrigation in Telangana: A review of evidence". In: *Economic and Political Weekly*, pp. 1421–1426.

Velásquez, David et al. (2020). "A method for detecting coffee leaf rust through wireless sensor networks, remote sensing, and deep learning: case study of the caturra variety in Colombia". In: *Applied Sciences* 10.2, p. 697.

Vermeulen, Sonja J et al. (2013). "Addressing uncertainty in adaptation planning for agriculture". In: *Proceedings of the National Academy of Sciences* 110.21, pp. 8357–8362.

Verpoorter, Charles et al. (2014). "A global inventory of lakes based on high-resolution satellite imagery". In: *Geophysical Research Letters* 41.18, pp. 6396–6402.

Yadav, SK et al. (2013). "A review of organic farming for sustainable agriculture in Northern India". In: *International Journal of Agronomy* 2013.

Yosinski, Jason et al. (2014). "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems* 27.