

AI Based Data Labelling For Object Detection.



Project Team

Sl. No.	Reg. No.	Student Name
1	18ETCS002004	ADARSH ITTIGI

Supervisors: Mr. Narasimha Murthy

August – 2018

**B. Tech. in Computer Science and Engineering
FACULTY OF ENGINEERING AND TECHNOLOGY
M. S. RAMAIAH UNIVERSITY OF APPLIED SCIENCES
Bangalore -560 058**



M.S.Ramaiah University of Applied Sciences – Faculty of Engineering and Technology (FET)

Declaration

AI BASED DATA LABELLING FOR OBJECT DETECTION

The Internship work is submitted in partial fulfilment of academic requirements for the award of **B. Tech.** Degree in the **Department of Computer Science and Engineering** of the Faculty of **Engineering and Technology** of Ramaiah University of Applied Sciences. The Internship report submitted here with is a result of our own work and in conformance to the guidelines on plagiarism as laid out in the University Student Handbook. All sections of the text and results which have been obtained from other sources are fully referenced. We understand that cheating and plagiarism constitute a breach of University regulations, hence this project report has been passed through plagiarism check and the report has been submitted to the supervisor.

Sl. No.	Reg. No.	Student Name	Signature
1	18ETCS002004	ADARSH ITTIGI	

Date : 8 December 2021

Acknowledgements

It is with extreme pleasure and pride that we present our B-Tech. dissertation titled “AI Based Data Labelling for object Detection.” We would like to express our sincere thanks and gratitude to the following people, who stood by us throughout, helping us with much required inputs, guidance, knowledge and supported us.

We take great pleasure to express our sincere thanks and gratitude to academic project guide <Narasimha Murthy K.>. Asst. Professor Department of CSE, for his support, guidance and suggestions throughout the project which is leading this project for the completion.

We express our sincere thanks to, Dr. Govind R. Kadambi, our respected Dean and to Dr.PVR Murthy Sir, Head of Department of Computer Science and Engineering, for their kind cooperation and support toward out dissertation, and to the management of Ramaiah University of Applied Science for their continued support. We are thankful to the staff members of the Computer Science and Engineering, RUAS for giving us good support and suggestion.

Lastly, we would like to thank our parents and friends for their continued support, encouragement and motivation and God for paving our way of success in this object.

Abstract

YOU ONLY LOOK ONCE:

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end

directly on detection performance. Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

LABEL STUDIO:

The First Step to any Image Processing is labelling the image, so here we present to you label studio which is a Data Labelling Platform to label various kinds to Data. Like Images, Time Series, Audio, Video. It is easy to Install using the pip install label-studio command in the command prompt By Default It will give the host name as localhost:8080 but you can change it later. To start the server we need to type in the command Label-studio start command to start the server, then you'll have to login and select what data you need to label, then start labelling the data.

Table of Contents

Acknowledgements.....	2
Abstract.....	3
List of Figures.....	Error! Bookmark not defined.
List of Tables.....	Error! Bookmark not defined.
1. Introduction.....	6
1.1 Introduction To YOLO	Error! Bookmark not defined.
1.2 Unified Detection	Error! Bookmark not defined.
1.2.1: Network Design.....	2
1.2.2: Training.....	2
1.2.3: Inference.....	2
1.2.4: Limitations of YOLO.....	2
1.3 Comparison to other Detection Systems	Error! Bookmark not defined.
1.4: Experiments:	
1.4.1: Comparison to other Real-Time Systems.....	4
1.4.2: Voc 2007 Error Analysis.....	4
1.4.3: Combining Fast R-CNN and YOLO.....	4
1.4.4: Voc 2012 Results.....	4
1.4.5: Generalizability: Person Detection in artwork.....	4
1.5: Real-Time Detection in the Wild.....	4
1.6: Conclusion.....	4
2. Exploring Label Studio:	
2.1: What is label studio?	
2.2: How to install label studio.	
2.3: Label Studio Terminology.	
2.4: Components and Architecture.	
2.5: Information Collected.	
2.6: Label Studio Features.	
3. Experiment:	
3.1: Background Knowledge:	
3.1.1: JavaScript	
3.1.2: AJAX calls	
3.1.3: API's	
3.1.4: JSON Data	
3.2: AIM:	
3.2.1: To explore Label studio and label some images and get the label in a	
Html Page.	

3.3: Methodology:

3.3.1: How to Get the Data from local host or Label Studio Server.

3.3.2: Use JavaScript to get the Label of the image.

1. Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems.

Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the Classifier is run at evenly spaced locations over the entire image.

More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After Classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescores the boxes based on other objects in the scene. This complex Pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

A single convolution network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems. For a demo of our system running in real-time on a webcam please see our project webpage:

Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN. Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs. YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones. We examine these tradeoffs further in our Experiments. All of our training and testing code is open source. A variety of pre-trained models are also available to download.

1.2 Unified Detection:-

We unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and realtime speeds while maintaining high average precision. Our system divides the input image into an $S \times S$ grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $\Pr(\text{Object}) \times \text{IOU}_{\text{truth pred}}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth. Each bounding box consists of 5 predictions: x , y , w , h , and confidence. The $(x; y)$ coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, $\Pr(\text{Class}|\text{Object})$. These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B . At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\Pr(\text{Class}|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{truth pred}} = \Pr(\text{Class}) * \text{IOU}_{\text{truth pred}} \quad (1)$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

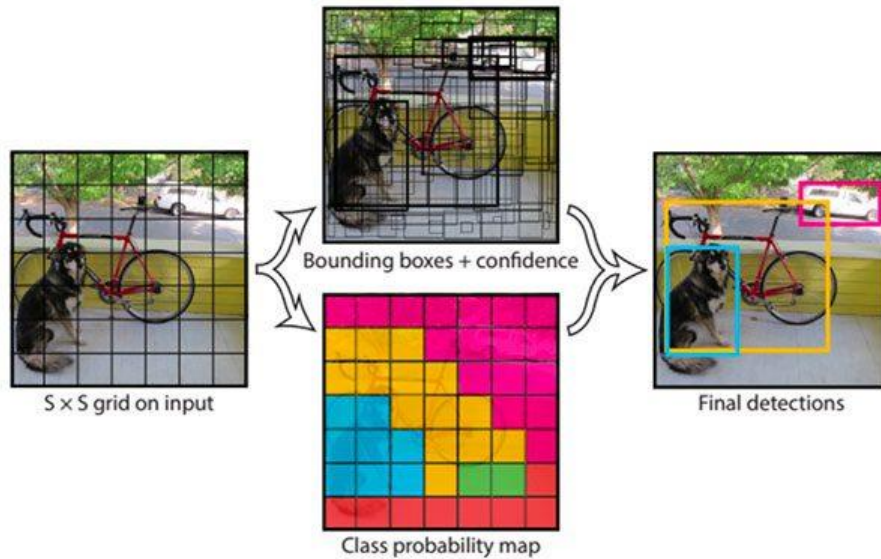


Figure 1: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

For evaluating YOLO on PASCAL VOC, we use $S = 7$, $B = 2$. PASCAL VOC has 20 labeled classes so $C = 20$. Our final prediction is a $7 \times 7 \times 30$ tensor.

1.2.1: Network Design:-

We implement this model as a convolution neural network and evaluate it on the PASCAL VOC detection dataset. The initial convolution layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. Our network architecture is inspired by the Google Net model for image classification. Our network has 24 convolution layers followed by 2 fully connected layers. Instead of the inception modules used by Google Net, we simply use 1×1 reduction layers followed by 3×3 convolution layers, similar to Lin et al. The full network is shown in Figure. We also train a fast version of YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolution layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO.

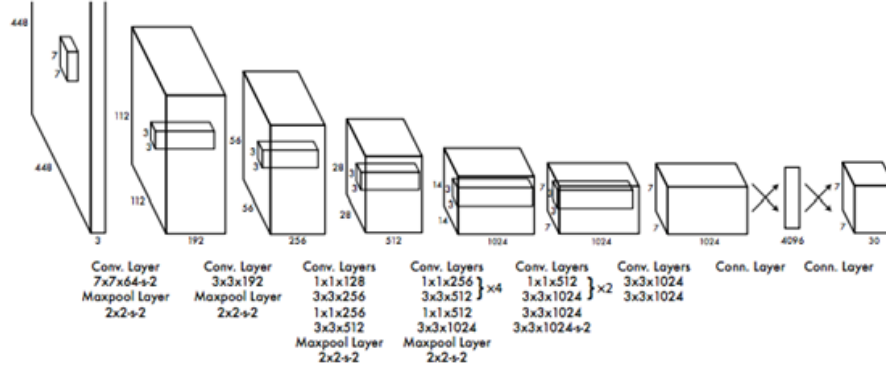


Figure 2: The Architecture. Our detection network has 24 convolution layers followed by 2 fully connected layers. Alternating 1 X 1 convolutional layers reduce the features space from preceding layers. We pre-train the convolutional layers on the ImageNet classification task at half the resolution (224 X 224 input image) and then double the resolution for detection.

The final output of our network is the 7 X 7 X 30 tensor of predictions.

1.2.2: Training:-

We pretrain our convolutional layers on the ImageNet 1000-class competition dataset. For pretraining we use the first 20 convolutional layers from Figure 2 followed by a average-pooling layer and a fully connected layer. We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe's Model Zoo.

We use the Darknet framework for all training and inference. We then convert the model to perform detection. Ren et al. show that adding both convolutional and connected layers to pretrained networks can improve performance. Following their example, we add four convolutional layers and two fully connected layers with randomly initialized weights.

Detection often requires fine-grained visual information so we increase the input resolution of the network from 224 X 224 to 448 X 448. Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1.

We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1. We use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise.} \end{cases}$$

We optimize for sum-Squared error in the output of our loss function:

$$\begin{aligned}
& \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}$$

where $\mathbb{1}_{ij}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the j th bounding box predictor in cell i is “responsible” for that prediction. Note that the loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability discussed earlier). It also only penalizes bounding box coordinate error if that predictor is “responsible” for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell). We train the network for about 135 epochs on the training and validation data sets from PASCAL VOC 2007 and 2012.

When testing on 2012 we also include the VOC 2007 test data for training. Throughout training we use a batch size of 64, a momentum of 0.9 and a decay of 0.0005. Our learning rate schedule is as follows: For the first epochs we slowly raise the learning rate from 10^{-3} to 10^{-2} .

If we start at a high learning rate our model often diverges due to unstable gradients. We continue training with 10^{-2} for 75 epochs, then 10^{-3} for 30 epochs, and finally 10^{-4} for 30 epochs.

To avoid over fitting we use dropout and extensive data augmentation. A dropout layer with rate= .5 after the first connected layer prevents co-adaptation between layers. For data augmentation we introduce random scaling and translations of up to 20% of the original image size. We also randomly adjust the exposure and saturation of the image by up to a factor of 1:5 in the HSV color space.

1.3.3: Inference:-

Just like in training, predicting detections for a test image only requires one network evaluation. On PASCAL VOC the network predicts 98 bounding boxes per image and class probabilities for each box. YOLO is extremely fast at test time since it only requires a single network evaluation, unlike classifier-based methods. The grid design enforces spatial diversity in the bounding box predictions. Often it is clear which grid cell an object falls in to and the network only predicts one box for each object. However, some large objects or objects near the border of multiple cells can be well localized by multiple cells. Non-maximal suppression can be used to fix these multiple detections. While not critical to performance as it is for R-CNN or DPM, non-maximal suppression adds 2- 3% in mAP.

1.2.4: Limitation of YOLO:-

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds. Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image. Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations

1.3: Comparison to other detection systems:

Object detection is a core problem in computer vision. Detection pipelines generally start by extracting a set of robust features from input images (Haar SIFT, HOG, convolutional features). Then, classifiers or localizers are used to identify objects in the feature space. These classifiers or localizers are run either in sliding window fashion over the whole image or on some subset of regions in the image. We compare the YOLO detection system to several top detection frameworks, highlighting key similarities and differences.

Deformable parts models. Deformable parts models (DPM) use a sliding window approach to object detection. DPM uses a disjoint pipeline to extract static features, classify regions, predict bounding boxes for high scoring regions, etc. Our system replaces all of these disparate parts with a single convolutional neural network. The network performs feature extraction, bounding box prediction, nonmaximal suppression, and contextual reasoning all concurrently. Instead of static features, the network trains the features in-line and optimizes them for the detection task. Our unified architecture leads to a faster, more accurate model than DPM.

R-CNN. R-CNN and its variants use region proposals instead of sliding windows to find objects in images. SelectiveSearch generates potential bounding boxes, a convolutional network extracts features, an SVM scores the boxes, a linear model adjusts the bounding boxes, and non-max suppression eliminates duplicate detections. Each stage of this complex pipeline must be precisely tuned independently and the resulting system is very slow, taking more than 40 seconds per image at test time. YOLO shares some similarities with R-CNN. Each grid cell proposes potential bounding boxes and scores those boxes using convolutional features. However, our system puts spatial constraints on the grid cell proposals which helps mitigate multiple detections of the same object. Our system also proposes far fewer bounding boxes, only 98 per image compared to about 2000 from Selective Search. Finally, our system combines these individual components into a single, jointly optimized model.

Other Fast Detectors Fast and Faster R-CNN focus on speeding up the R-CNN framework by sharing computation and using neural networks to propose regions instead of Selective Search. While they offer speed and accuracy improvements over R-CNN, both still fall short of real-time performance. Many research efforts focus on speeding up the DPM pipeline. They speed up HOG computation, use cascades, and push computation to GPUs. However, only 30Hz DPM actually runs in real-time. Instead of trying to optimize individual components of a large detection pipeline, YOLO throws out the pipeline entirely and is fast by design. Detectors for single classes like faces or people can be highly optimized since they have to deal with much less variation. YOLO is a general purpose detector that learns to detect a variety of objects simultaneously. Deep MultiBox. Unlike R-CNN, Szegedy et al. train a convolutional neural network to predict regions of interest instead of using Selective Search. MultiBox can also perform single object detection by replacing the confidence prediction with a single class prediction. However, Multi-Box cannot perform general object detection and is still just a piece in a larger detection pipeline, requiring further image patch classification. Both YOLO and MultiBox use a convolutional network to predict bounding boxes in an image but YOLO is a complete detection system.

OverFeat. Sermanet et al. train a convolutional neural network to perform localization and adapt that localizer to perform detection. OverFeat efficiently performs sliding window detection but it is still a disjoint system. Over-Feat optimizes for localization, not detection performance. Like DPM, the localizer only sees local information when making a prediction. OverFeat cannot reason about global context and thus requires significant post-processing to produce coherent detections.

MultiGrasp. Our work is similar in design to work on grasp detection by Redmon et al. Our grid approach to bounding box prediction is based on the MultiGrasp system for regression to grasps. However, grasp detection is a much simpler task than object detection. MultiGrasp only needs to predict a single graspable region for an image containing one object. It doesn't have to estimate

the size, location, or boundaries of the object or predict it's class, only find a region suitable for grasping. YOLO predicts both bounding boxes and class probabilities for multiple objects of multiple classes in an image.

1.4: Experiments:

First we compare YOLO with other real-time detection systems on PASCAL VOC 2007. To understand the differences between YOLO and R-CNN variants we explore the errors on VOC 2007 made by YOLO and Fast R-CNN, one of the highest performing versions of R-CNN. Based on the different error profiles we show that YOLO can be used to rescore Fast R-CNN detections and reduce the errors from background false positives, giving a significant performance boost. We also present VOC 2012 results and compare mAP to current state-of-the-art methods. Finally, we show that YOLO generalizes to new domains better than other detectors on two artwork datasets.

1.4.1: Comparison to other real time Systems:

Many research efforts in object detection focus on making standard detection pipelines fast. However, only Sadeghi et al. actually produce a detection system that runs in real-time (30 frames per second or better). We compare YOLO to their GPU implementation of DPM which runs either at 30Hz or 100Hz. While the other efforts don't reach the real-time milestone we also compare their relative mAP and speed to examine

the accuracy-performance tradeoffs available in object detection systems. Fast YOLO is the fastest object detection method on PASCAL; as far as we know, it is the fastest extant object detector. With 52:7% mAP, it is more than twice as accurate as prior work on real-time detection. YOLO pushes mAP to 63:4% while still maintaining real-time performance. We also train YOLO using VGG-16. This model is more accurate but also significantly slower than YOLO. It is useful

for comparison to other detection systems that rely on VGG-16 but since it is slower than real-time the rest of the paper focuses on our faster models. Fastest DPM effectively speeds up DPM without sacrificing much mAP but it still misses real-time performance by a factor of 2 .It also is limited by DPM's relatively low accuracy on detection compared to neural network approaches.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

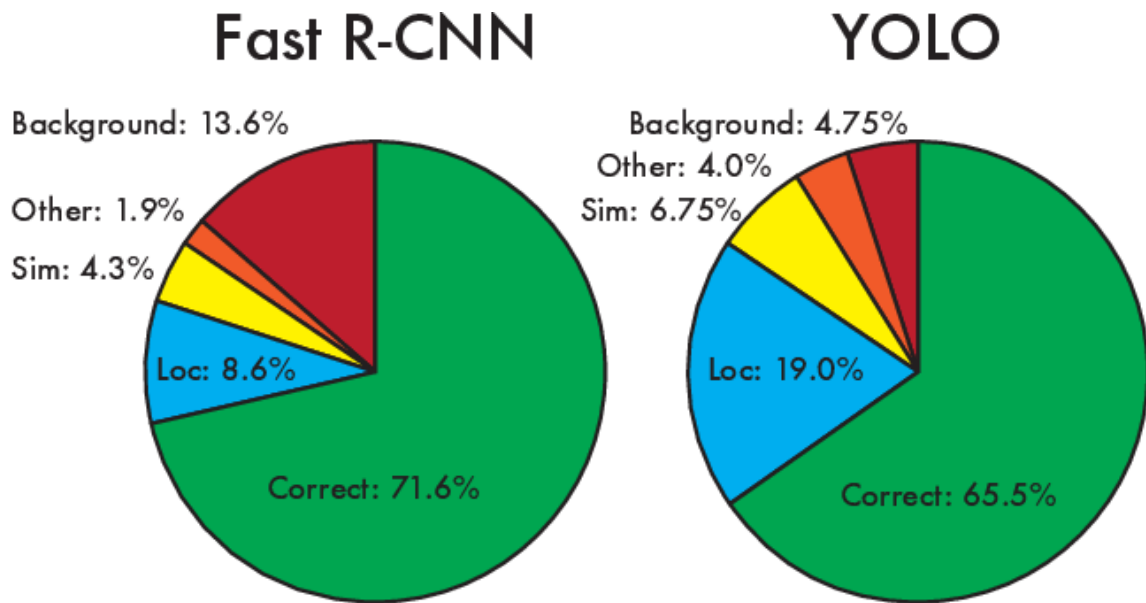
Table 1: Real-Time Systems on PASCAL VOC 2007. Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

R-CNN minus R replaces Selective Search with static bounding box proposals. While it is much faster than R-CNN, it still falls short of real-time and takes a significant accuracy hit from not having good proposals. Fast R-CNN speeds up the classification stage of R-CNN but it still relies on selective search which can take around 2 seconds per image to generate bounding box proposals. Thus it has high mAP but at 0.5 fps it is still far from realtime. The recent Faster R-CNN replaces selective search with a neural network to propose bounding boxes, similar to Szegedy et al. In our tests, their most accurate model achieves 7 fps while a smaller, less accurate one runs at 18 fps. The VGG-16 version of Faster R-CNN is 10 mAP higher but is also 6 times slower than YOLO. The Zeiler- Fergus Faster R-CNN is only 2.5 times slower than YOLO but is also less accurate.

1.4.2: VOC 2007 Error Analysis:

To further examine the differences between YOLO and state-of-the-art detectors, we look at a detailed breakdown of results on VOC 2007. We compare YOLO to Fast RCNN since Fast R-CNN is one of the highest performing detectors on PASCAL and it's detections are publicly available. We use the methodology and tools of Hoiem et al. [19] For each category at test time we look at the top N predictions for that category. Each prediction is either correct or it is classified based on the type of error:

- Correct: correct class and $\text{IOU} > .5$
- Localization: correct class, $.1 < \text{IOU} < .5$
- Similar: class is similar, $\text{IOU} > .1$



- Other: class is wrong, IOU > :1
- Background: IOU < :1 for any object

Figure shows the breakdown of each error type averaged across all 20 classes. YOLO struggles to localize objects correctly. Localization errors account for more of YOLO's errors than all other sources combined. Fast R-CNN makes much fewer localization errors but far more background errors. 13.6% of it's top detections are false positives that don't contain any objects. Fast R-CNN is almost 3x more likely to predict background detections than YOLO.

1.4.3: Combining Fast R-CNN and YOLO:-

YOLO makes far fewer background mistakes than Fast R-CNN. By using YOLO to eliminate background detections from Fast R-CNN we get a significant boost in performance. For every bounding box that R-CNN predicts we check to see if YOLO predicts a similar box. If it does, we give that prediction a boost based on the probability predicted by YOLO and the overlap between the two boxes. The best Fast R-CNN model achieves a mAP of 71.8% on the VOC 2007 test set. When combined with YOLO, its mAP increases by 3.2% to 75.0%. We also tried combining the top Fast R-CNN model with several other versions of Fast R-CNN. Those ensembles produced small increases in mAP between .3 and .6%, see Table for details.

Table 1: Model combination experiments on VOC 2007. We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.

	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	66.9	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	75.0	3.2

Table 2 : PASCAL VOC 2012 Leaderboard. YOLO compared with the full comp4 (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR-CNN-MORE-DATA [11]	73.9	85.5	82.9	76.6	57.8	62.7	79.4	77.2	86.6	55.0	79.1	62.2	87.0	83.4	84.7	78.9	45.3	73.4	65.8	80.1	74.0
HyperNet-VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	79.8	87.7	49.6	74.9	52.1	86.0	81.7	83.3	81.8	48.6	73.5	59.4	79.9	65.7
HyperNet-SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	89.4	49.4	75.5	57.0	87.5	80.9	81.0	74.7	41.8	71.5	68.5	82.1	67.2
MR-CNN-S-CNN [1]	70.7	85.0	79.6	71.5	55.5	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [30]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP-ENS-COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	68.8	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [10]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH-FGS-STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS-NIN-C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS-NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG-BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	77.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [10]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

mAP increases by 3.2% to 75.0%. We also tried combining the top Fast R-CNN model with several other versions of Fast R-CNN. Those ensembles produced small increases in mAP between .3 and .6%, see Table 2 for details. The boost from YOLO is not simply a byproduct of model ensembling since there is little benefit from combining different versions of Fast R-CNN. Rather, it is precisely because YOLO makes different kinds of mistakes at test time that it is so effective at boosting Fast R-CNN's performance. Unfortunately, this combination doesn't benefit from the speed of YOLO since we run each model separately and then combine the results. However, since YOLO is so fast it doesn't add any significant computational time compared to Fast R-CNN.

1.4.4: VOC 2012 Results:-

On the VOC 2012 test set, YOLO scores 57.9% mAP. This is lower than the current state of the art, closer to the original R-CNN using VGG-16, see Table 3. Our system struggles with small objects compared to its closest competitors. On categories like bottle, sheep, and tv/monitor

YOLO scores 8-10% lower than R-CNN or Feature Edit. However, on other categories like cat and train YOLO achieves higher performance. Our combined Fast R-CNN + YOLO model is one of the highest performing detection methods. Fast R-CNN gets a 2.3% improvement from the combination with YOLO, boosting it 5 spots up on the public leader board.

1.4.5: Generalizability: Person Detection in Artwork

Academic datasets for object detection draw the training and testing data from the same distribution. In real-world applications it is hard to predict all possible use cases and the test data can diverge from what the system has seen before. We compare YOLO to other detection systems on the Picasso Dataset and the People-Art Dataset, [two datasets for testing person detection on artwork. Figure 5 shows comparative performance between YOLO and other detection methods. For reference, we give VOC 2007 detection AP on person where all models are trained only on VOC 2007 data. On Picasso models are trained on VOC 2012 while on People-Art they are trained on VOC 2010. R-CNN has high AP on VOC 2007. However, R-CNN drops off considerably when applied to artwork. R-CNN uses Selective Search for bounding box proposals which is tuned for natural images. The classifier step in R-CNN only sees small regions and needs good proposals. DPM maintains its AP well when applied to artwork. Prior work theorizes that DPM performs well because it has strong spatial models of the shape and layout of objects. Though DPM doesn't degrade as much as R-CNN, it starts from a lower AP. YOLO has good performance on VOC 2007 and its AP degrades less than other methods when applied to artwork. Like DPM, YOLO models the size and shape of objects, as well as relationships between objects and where objects commonly appear. Artwork and natural images are very different on a pixel level but they are similar in terms of the size and shape of objects, thus YOLO can still predict good bounding boxes and detections.

1.5: Real Time Detections in the wild:-

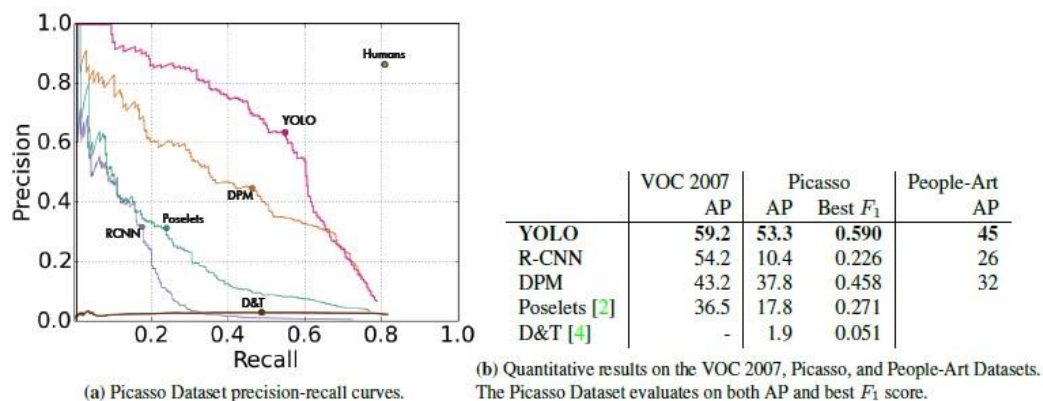


Figure 5: Generalization results on Picasso and People-Art datasets.

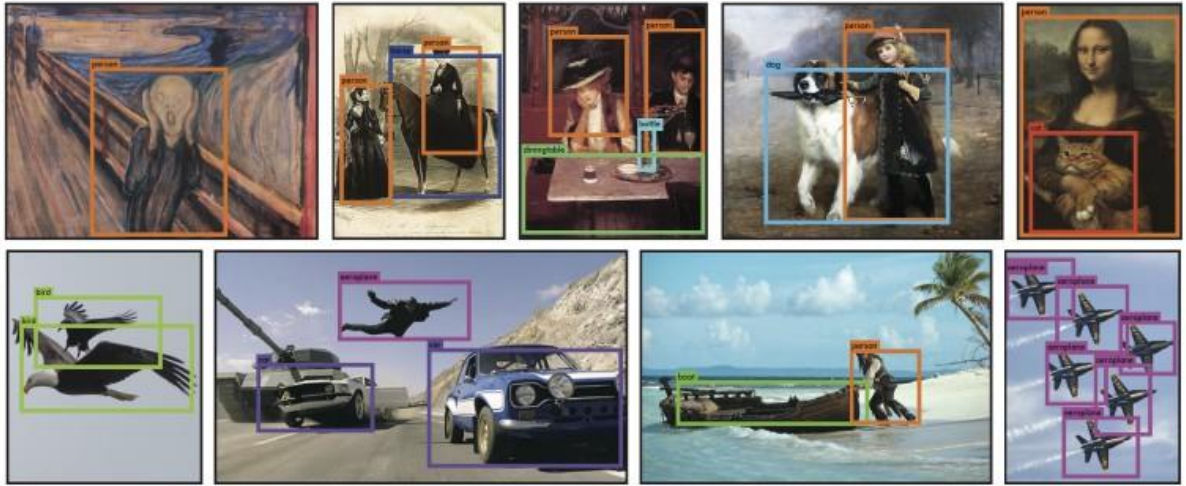


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

YOLO is a fast, accurate object detector, making it ideal for computer vision applications. We connect YOLO to a webcam and verify that it maintains real-time performance including the time to fetch images from the camera and display the detections. The resulting system is interactive and engaging. While YOLO processes images individually, when attached to a webcam it functions like a tracking system, detecting objects as they move around and change in appearance.

1.6: Conclusion:-

We introduce YOLO, a unified model for object detection. Our model is simple to construct and can be trained directly on full images. Unlike classifier-based approaches, YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly. Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection. YOLO also generalizes well to new domains making it ideal for applications that rely on fast, robust object detection.

2. Label Studio:-

2.1: What is Label Studio?

Label Studio is an open source data labelling tool for labelling and exploring multiple types of data. You can perform different types of labelling with many data formats.

You can also integrate Label Studio with machine learning models to supply predictions for labels (pre-labels), or perform continuous active learning.

Labelling Workflow With Label Studio:-

Start and finish a labelling project with Label Studio by following these steps:

1. Install Label Studio.
2. Start Label Studio.
3. Create accounts for Label Studio. Create an account to manage and set up labeling projects.
4. Restrict access to the project. Set up role-based access control. Only available in Label Studio Enterprise Edition.
5. Set up the labelling project. Define the type of labelling to perform on the dataset and configure project settings.
6. Set up the labelling interface. Add the labels that you want annotators to apply and customize the labelling interface.
7. Import data as labelling tasks.
8. Label and annotate the data.
9. Review the annotated tasks. Only available in Label Studio Enterprise Edition.
10. Export the labelled data or the annotations.

2.2: How to Install Label Studio:-

1. Install Label Studio Using

“pip install label-studio” Command in your command Prompt.

2. Start Label Studio using

“label-studio start” Command in your Command Prompt.

3. Open the Label Studio UI at <http://localhost:8080>.
4. Sign up with an email address and password that you create.
5. Click **Create** to create a project and start labeling data.
6. Name the project, and if you want, type a description and select a color.
7. Click **Data Import** and upload the data files that you want to use. If you want to use data from a local directory, cloud storage bucket, or database, skip this step for now.
8. Click **Labeling Setup** and choose a template and customize the label names for your use case.
9. Click **Save** to save your project.

You are ready to Start Labelling and annotating your Data.

2.3: Label Studio terminology:-

When you upload data to Label Studio, each item in the dataset becomes a labeling task. The following table describes some terms you might encounter as you use Label Studio.

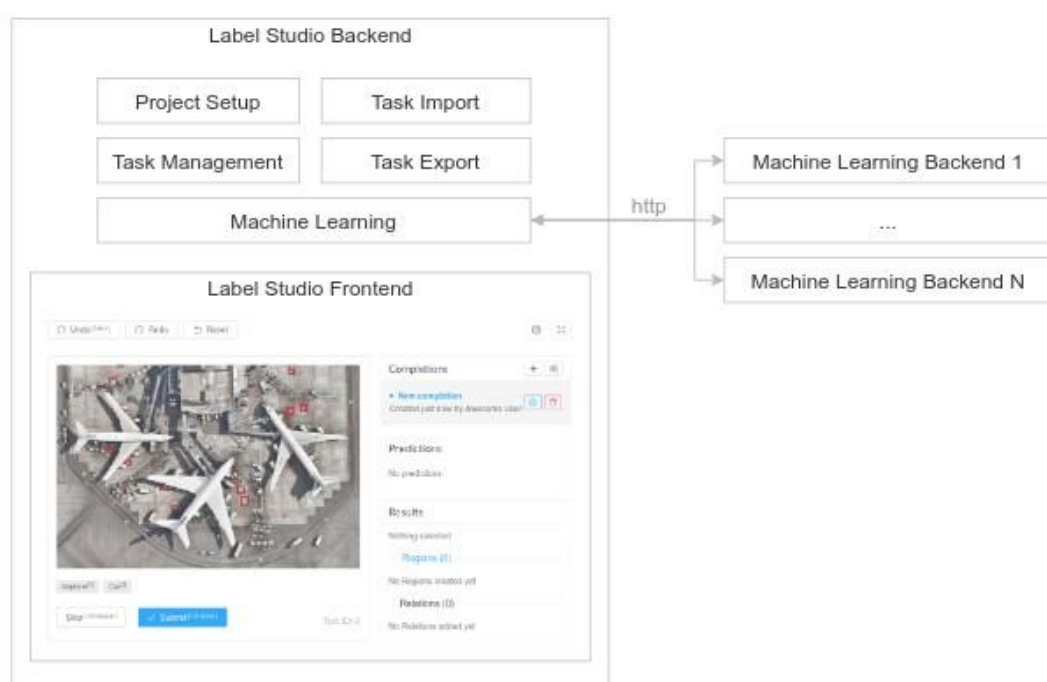
Term	Description
Dataset	What you import into Label Studio, comprised of individual items.
Task	What Label Studio transforms your individual dataset items into.
Labels	What you add to each dataset item while performing a labeling task in Label Studio.
Region	The portion of the dataset item that has a label assigned to it.
Relation	A defined relationship between two labeled regions.
Result	A label applied to a specific region.
Pre-labeling	What machine learning models perform in Label Studio or separate from Label Studio. The result of predicting labels for items in a dataset are predicted labels, or pre-annotations.
Annotations	The output of a labeling task. Previously called “completions”.
Templates	Example labeling configurations that you can use to specify the type of labeling that you’re performing with your dataset.
Tags	Configuration options to customize the labeling interface.

2.4 Components and Architecture:-

You can use any of the Label Studio components in your own tools, or customize them to suit your needs. Before customizing Label Studio extensively, you might want to review Label Studio Enterprise Edition to see if it already contains the relevant functionality you want to build.

The component parts of Label Studio are available as modular extensible packages that you can integrate into your existing machine learning processes and tools.

Module	Technology	Description
Label Studio Backend	Python and Django	Use to perform data labeling.
Label Studio Frontend	JavaScript web app using React and MST	Perform data labeling in a user interface.
Data Manager	JavaScript web app using React	Manage data and tasks for labeling.
Machine Learning Backends	Python	Predict data labels at various parts of the labeling process.



Information Collected by label Studio:-

Label Studio collects anonymous usage statistics about the number of page visits and data types being used in labelling configurations that you set up. No sensitive information is included in the information we collect. The information we collect helps us improve the experience of labelling data in Label Studio and helps us plan future data types and labelling configurations to support.

2.6: Label Studio Features:-

Label Studio is available as a Community Edition open source data labeling tool and as paid versions with extended functionality and support. Smaller organizations might want to consider the SaaS option, Label Studio Teams, and larger teams with robust data labeling needs can get the Enterprise Edition.

Functionality	Community	Teams	Enterprise
User Management			
User accounts to associate labeling activities to a specific user.	✓	✓	✓
Role-based access control for each user account.	✗	✗	✓
Organizations and workspaces to manage users and projects.	✗	✓	✓
Project Management			
Projects to manage data labeling activities.	✓	✓	✓
Templates to get started with specific data labeling tasks faster.	✓	✓	✓
Data Management			
Manage your data in a user interface.	✓	✓	✓
Import data from many sources.	✓	✓	✓
Export data into many formats.	✓	✓	✓
Synchronize data from and to remote data storage.	✓	✓	✓
Data Labeling Workflows			
Assign specific annotators to specific tasks.	✗	✓	✓
Automatic queue management.	✗	✓	✓
Label text, images, audio data, HTML, and time series data.	✓	✓	✓
Label mixed types of data.	✓	✓	✓
Synchronize data from and to remote data storage.	✓	✓	✓
Annotator-specific view.	✗	✗	✓
Annotator Performance			
Control label quality by monitoring annotator agreement.	✗	✓	✓
Manage and review annotator performance.	✗	✓	✓
Verify model and annotator accuracy against ground truth annotations.	✗	✓	✓
Verify annotation results.	✗	✓	✓
Assign reviewers to review annotation results.	✗	✓	✓
Machine Learning			
Connect machine learning models to Label Studio with an SDK.	✓	✓	✓

Connect machine learning models to Label Studio with an SDK.	✓	✓	✓
Accelerate labeling with active learning.	✓	✓	✓
Automatically label dataset items with ML models.	✓	✓	✓
Analytics and Reporting			
Reporting and analytics on labeling and annotation activity.	✗	✓	✓
Activity log to use to audit annotator activity.	✗	✗	✓
Advanced Functionality			
API access to manage Label Studio.	✓	✓	✓
On-premises hosting of Label Studio.	✗	✗	✓
Support for single sign-on using LDAP or SAML.	✗	✗	✓

3. Experiment:-

3.1. Background Knowledge:-

3.1.1: JavaScript:-

JavaScript is a programming language that adds interactivity to your website. This happens in games, in the behavior of responses when buttons are pressed or with data entry on forms; with dynamic styling; with animation, etc.

What is JavaScript?

JavaScript is a Powerful Programming Language that can add interactivity to a website. It was invented by Brendan Eich (co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation).

JavaScript is versatile and beginner-friendly. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

JavaScript itself is relatively compact, yet very flexible. Developers have written a variety of tools on top of the core JavaScript language, unlocking a vast amount of functionality with minimum effort. These include:

- Browser Application Programming Interfaces (API's) built into web browsers, providing functionality such as dynamically creating HTML and setting CSS styles; collecting and manipulating a video stream from a user's webcam, or generating 3D graphics and audio samples.
- Third-party APIs that allow developers to incorporate functionality in sites from other content providers, such as Twitter or Facebook.
- Third-party frameworks and libraries that you can apply to HTML to accelerate the work of building sites and applications.

3.1.2: Ajax Calls:-

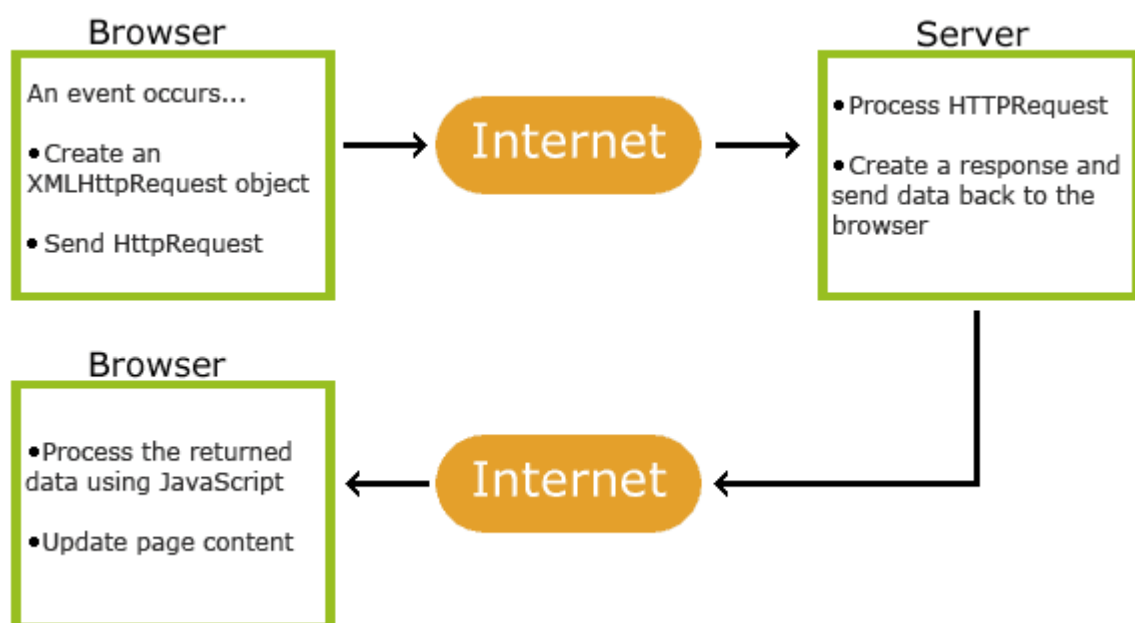
AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.



1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

3.1.3: API's:-

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API.

What Is an Example of an API?

When you use an application on your mobile phone, the application connects to the Internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions and sends it back to your phone. The application then interprets that data and presents you with the information you wanted in a readable way. This is what an API is - all of this happens via API.

To explain this better, let us take a familiar example.

Imagine you're sitting at a table in a restaurant with a menu of choices to order from. The kitchen is the part of the "system" that will prepare your order. What is missing is the critical link to communicate your order to the kitchen and deliver your food back to your table. That's where the waiter or API comes in. The waiter is the messenger – or API – that takes your request or order and tells the kitchen – the system – what to do. Then the waiter delivers the response back to you; in this case, it is the food.

Here is a real-life API example. You may be familiar with the process of searching flights online. Just like the restaurant, you have a variety of options to choose from, including different cities, departure and return dates, and more. Let us imagine that you're booking you are flight on an airline website. You choose a departure city and date, a return city and date, cabin class, as well

as other variables. In order to book your flight, you interact with the airline's website to access their database and see if any seats are available on those dates and what the costs might be.

The Modern API

Over the years, what an “API” is has often described any sort of generic connectivity interface to an application. More recently, however, the modern API has taken on some characteristics that make them extraordinarily valuable and useful:

- Modern APIs adhere to standards (typically HTTP and REST), that are developer-friendly, easily accessible and understood broadly
- They are treated more like products than code. They are designed for consumption for specific audiences (e.g., mobile developers), they are documented, and they are versioned in a way that users can have certain expectations of its maintenance and lifecycle.
- Because they are much more standardized, they have a much stronger discipline for security and governance, as well as monitored and managed for performance and scale
- As any other piece of productized software, the modern API has its own software development lifecycle (SDLC) of designing, testing, building, managing, and versioning. Also, modern APIs are well documented for consumption and versioning.

3.1.4: JSON Data:-

JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa). You'll come across it quite often, so in this article we give you all you need to work with JSON using JavaScript, including parsing JSON so you can access data within it, and creating JSON.

JSON is a text-based data format following JavaScript object syntax, which was popularized by Douglas Crockford. Even though it closely resembles JavaScript object literal syntax, it can be used independently from JavaScript, and many programming environments feature the ability to read (parse) and generate JSON.

JSON exists as a string — useful when you want to transmit data across a network. It needs to be converted to a native JavaScript object when you want to access the data. This is not a big issue — JavaScript provides a global JSON object that has methods available for converting between the two.

A JSON string can be stored in its own file, which is basically just a text file with an extension of .json, and a MIME type of application/json.

3.2: AIM:

3.2.1: To explore Label Studio and Label some images and get the label in a HTML page.

Our Aim in the project is to Explore Label Studio, as in, we have to label all types of data in label Studio, How do we do this, Firstly We will Start Label Studio in Command Prompt By using Label-Studio start command which will re-direct us to localhost:8080 server then we have to enter our login credentials or sign up if you are not already a part of the label Studio Community, and then you can start to label your images or any data.

Welcome to Label Studio Community Edition

A full-fledged open source solution for data labeling



[SIGN UP](#)

[LOG IN](#)

LOG IN

Figure 3: represents the image after we have started the label studio server and it has redirected us to the login page.

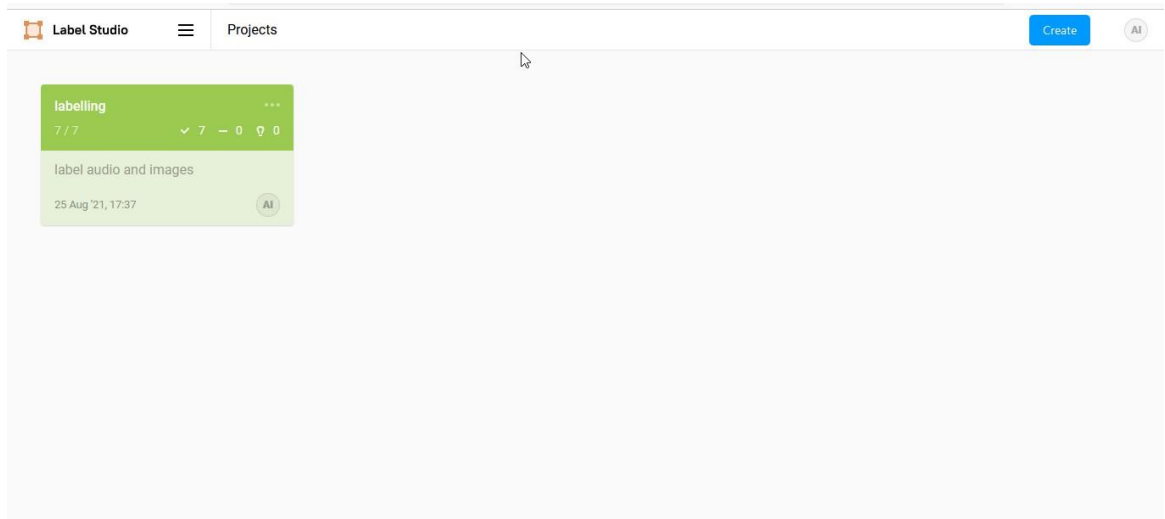


Figure 4: represents after we have login to label Studio

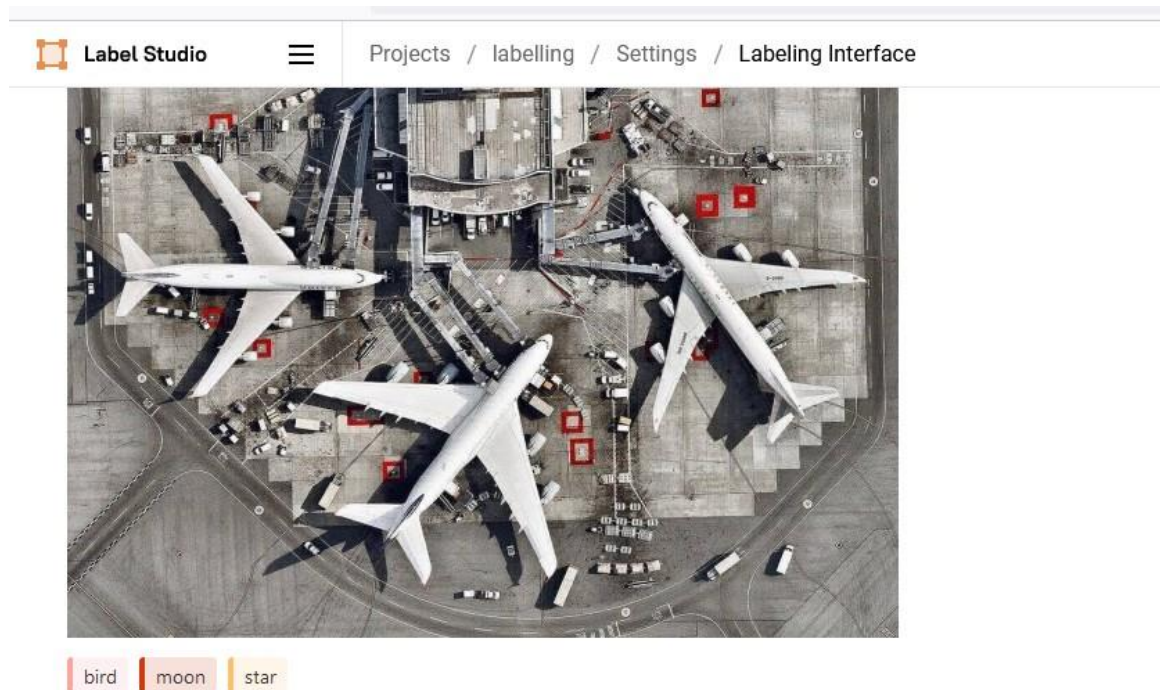


Figure 5: represents our labelling interface where we set up our labels.

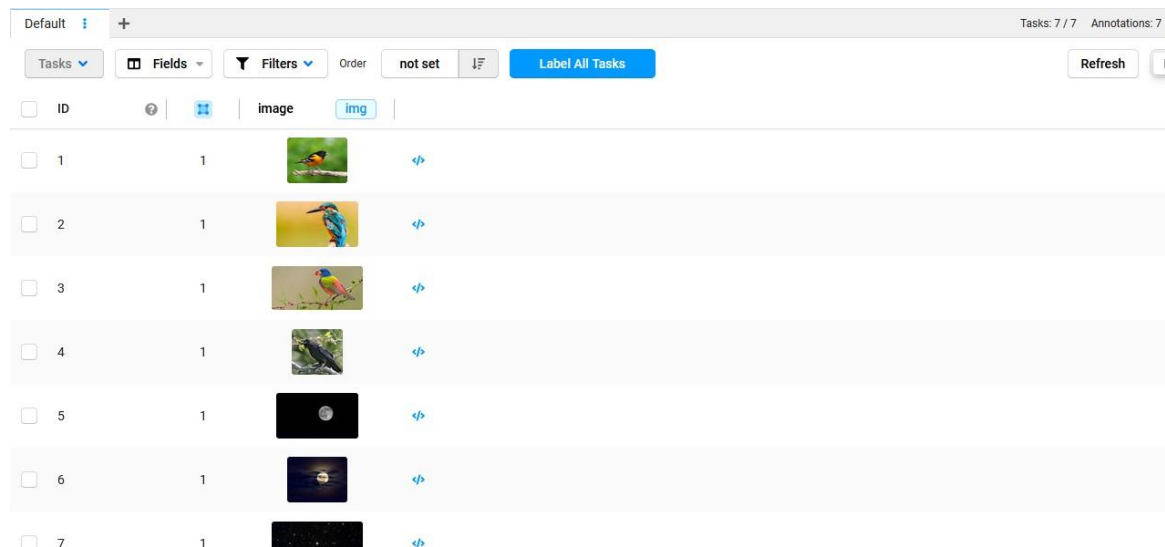


Figure 6: Sample image's that i have uploaded in label studio.

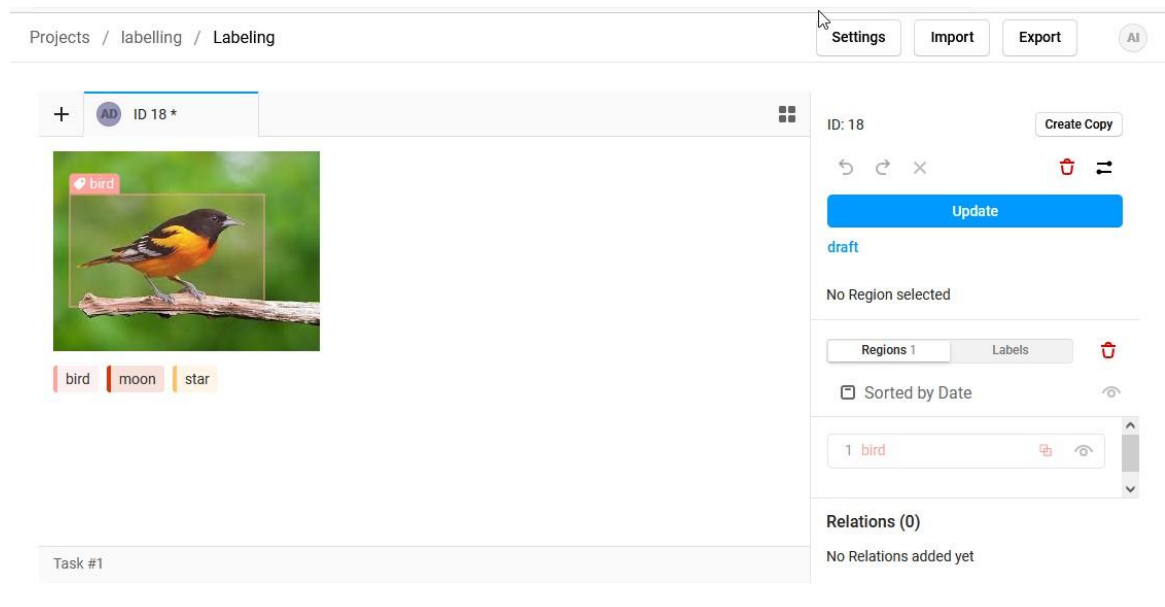


Figure 7: image labelling page.

Once all of the images are labelled, Our Next step is to get the output label and integrate it to a html page.

3.3: Methodology:-

3.3.1: How to get the data from local host or label Studio Server:-

We Need to Use label Studio Api to get the Json data of the label, As we require the json data we need to make sure of the version Compatibility of label Studio, We require version 1.2 of label

studio in order to get the json data.

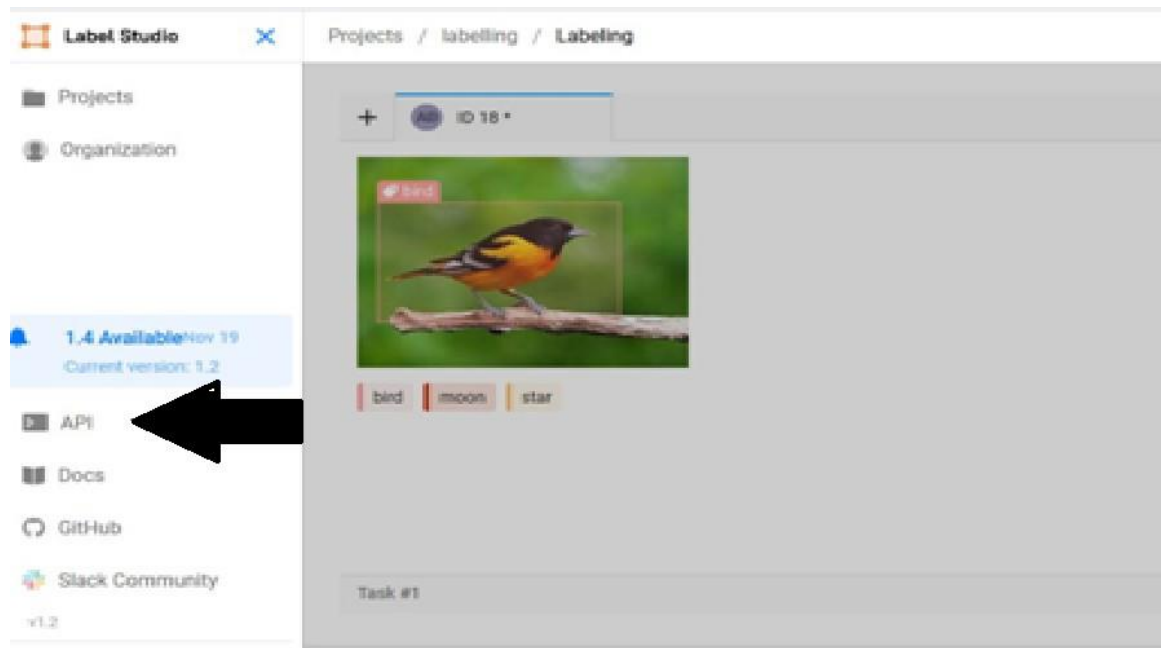


Figure 8: Go to the Api documentation to refer to the list of Api's that are present for developers to use.

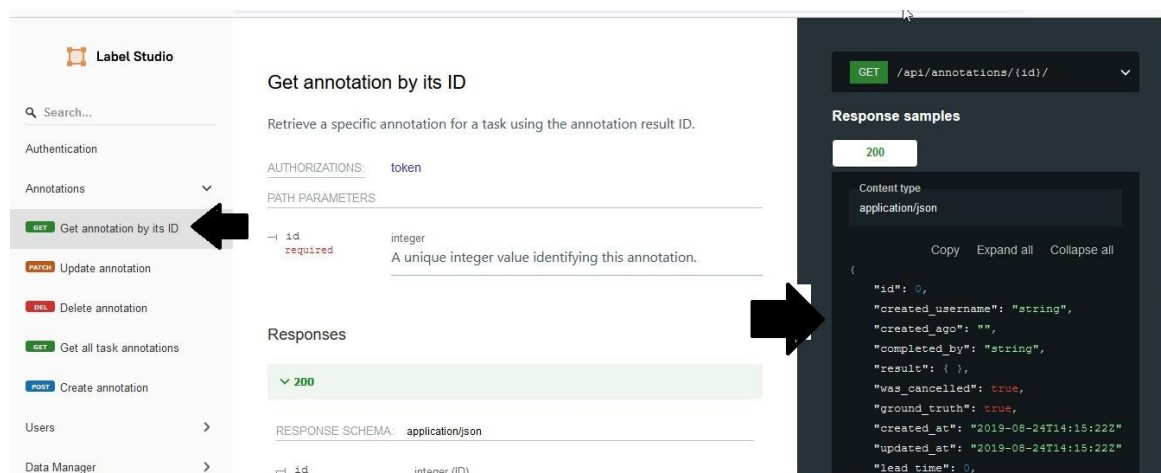


Figure 9: after going to the api documentation select Get annotations by ID and on the right side you will get to know the output format of the json data.

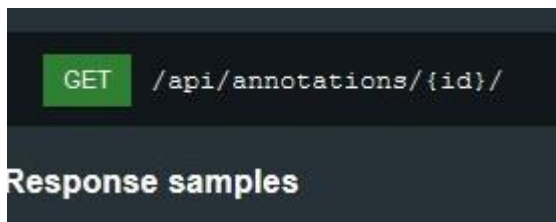


Figure 10: Select this and you will get the url for the json data where the data for which the label is present copy this url and paste it in your browser.

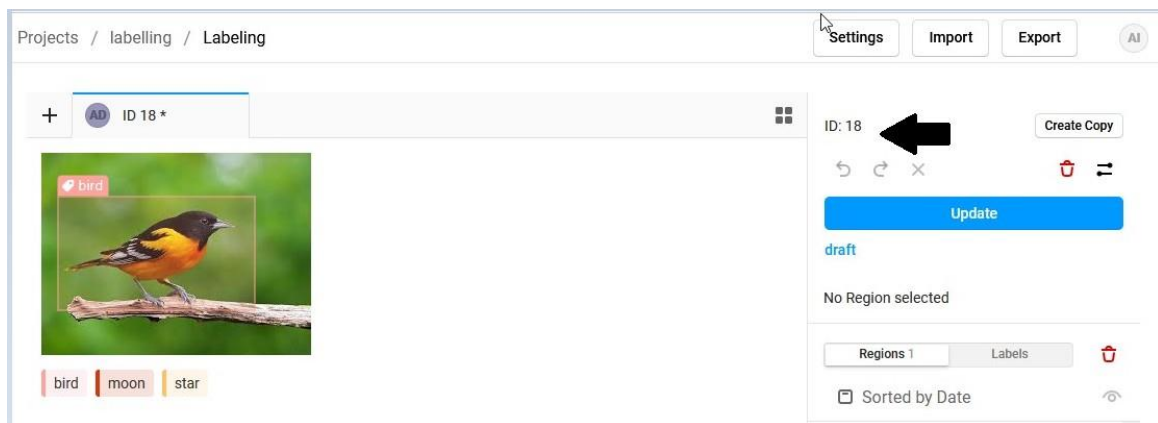


Figure 11: Replace the id part of the URL with the ID number which is present in the data label page to get the json data.

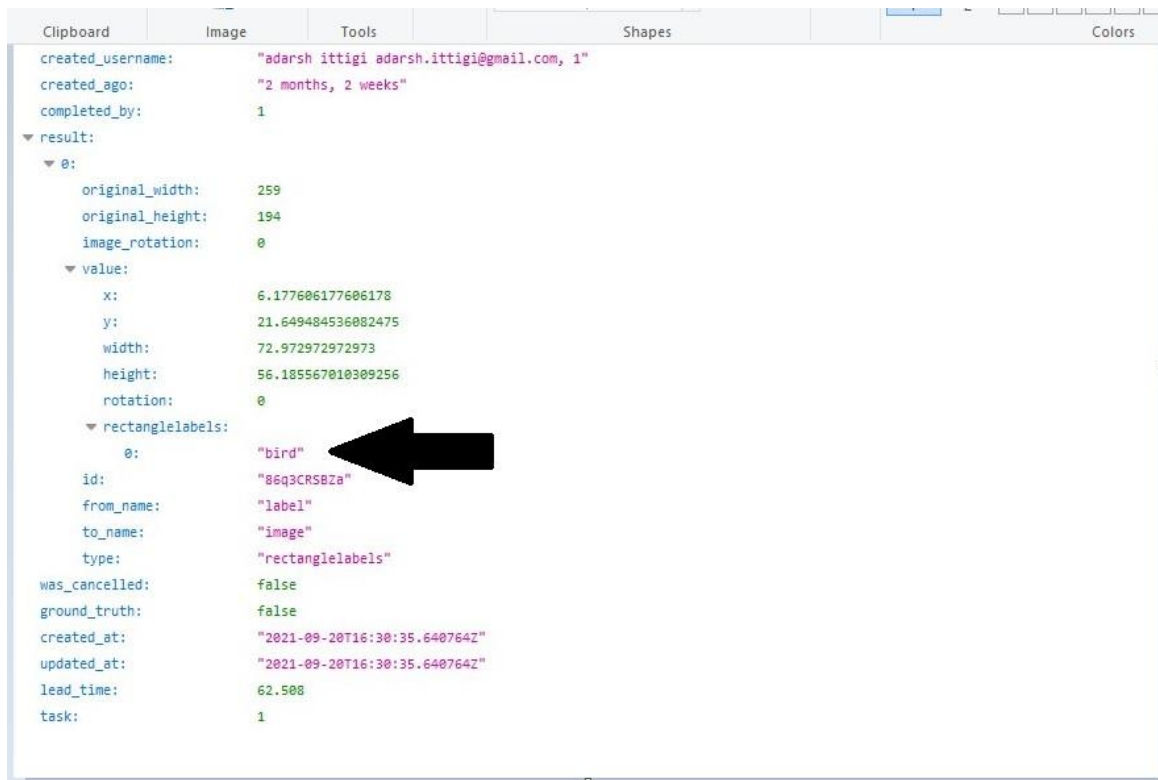


Figure 12: Represent's the json data after pasting the URL from the Get annotations Api reference.

3.3.2: Use JavaScript to get the label of the image:

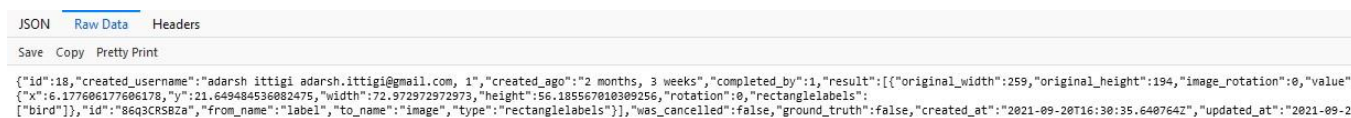


Figure 13: Convert the json data into a raw json format, and copy it and paste it in your javascript by declaring a variable.

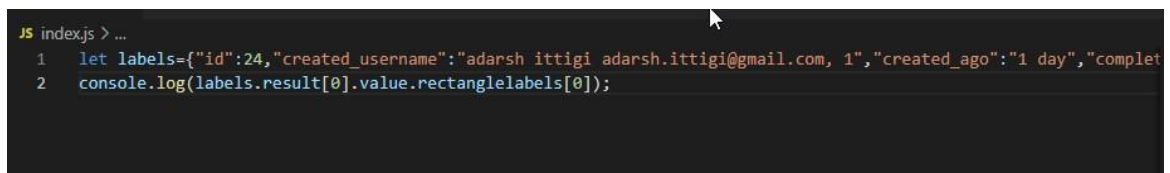


Figure 14: represents the code for creating a variable labels and pasting the raw json format and logging it into our html page.

```
index.html > html > body
1 <html>
2   <head>
3     <title>Label Studio</title>
4   </head>
5   <body style="background-color: #f0f0f0;">
6     <h1 style="color: #000080;">AI based data labelling for object detection.</h1>
7
8     <h1 style="text-align: center;">Choose files for uploading</h1>
9     <script src="index.js"></script>
10    <form action="http://localhost:8080/projects/1/data?upload.=bird1.jpg&tab=10&task=1" style="text-align: center;">
11      <input type="file" id="upload" name="upload.">
12      <input type="submit" value="submit">
13    </form>
14
15  </body>
16
17 </html>
```

Figure 15: represents the code for our html page.

AI based data labelling for object detection.

Choose files for uploading

No file selected.

Figure 16: represents our html page for the above code.

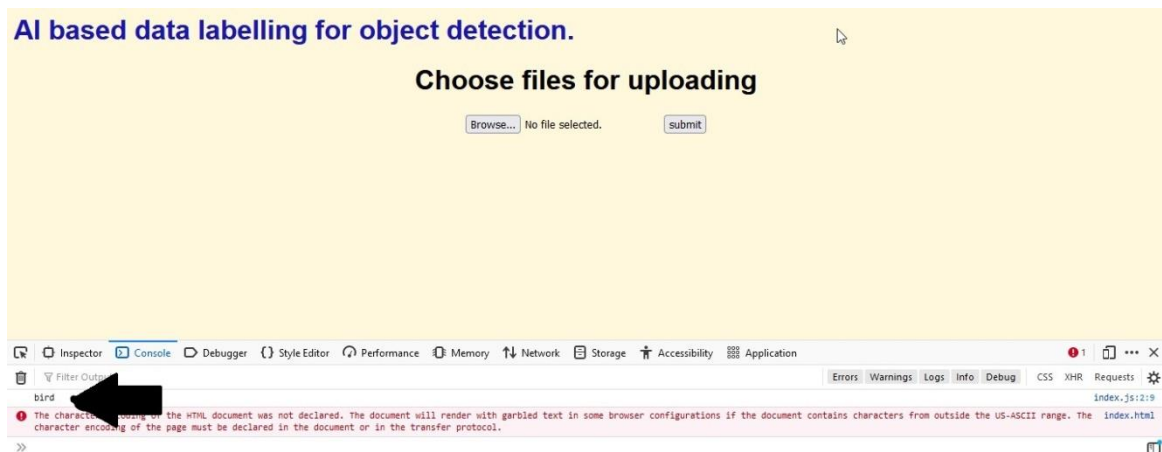


Figure 17: represents the output after applying JavaScript to our json data and console logging it into our html page, as you can see it shows bird which we have labelled it in label studio.

References:-

https://www.w3schools.com/xml/ajax_intro.asp

https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics

<https://www.mulesoft.com/resources/api/what-is-an-api>

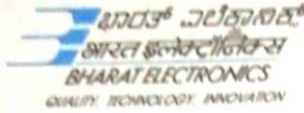

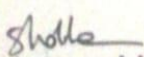
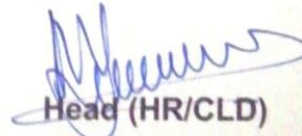
<https://labelstud.io/>

<https://pjreddie.com/media/files/papers/YOLOv3.pdf>

YOLOv3: An Incremental Improvement

Joseph Redmon, Ali Farhadi

University of Washington

 ಭಾರತ ಎಲೆಕ್ಟ್ರಾನಿಕ್ಸ್ भारत इलेक्ट्रॉनिक्स BHARAT ELECTRONICS QUALITY TECHNOLOGY INNOVATION	 Skill India कौशल भारत - कुशल भारत
CENTRE FOR LEARNING AND DEVELOPMENT	
BHARAT ELECTRONICS LIMITED (A Govt. of India Enterprise, Ministry of Defence) Jalahalli Post, Bengaluru - 560 013, India	
Certificate	
This is to certify that	
Sri./Smt./Kum. <u>ADARSH ITTIGI</u>	
Ref No. <u>1410/CLD/HR/2021-22/27/124</u>	
student of <u>RAMAIAH UNIVERSITY,</u>	
<u>BANGALORE</u>	
carried out Project Work/Internship on <u>AI BASED</u>	
<u>DATA LABELLING FOR OBJECT DETECTION</u>	
in <u>SOFTWARE</u>	
SBU/CSG of BEL, Bengaluru from <u>23rd AUGUST 2021</u>	
to <u>22nd SEPTEMBER 2021</u>	
He/She was regular and punctual in his/her attendance and his/her conduct was satisfactory during the period.	
 Project / Internship Guide	 Head (HR/CLD)
Date : <u>22-Sep-2021</u>	
Place : <u>Bengaluru</u>	