4th Year Project

AI Based Electronic Component Identifier

Violet Concordia
B00125142

Department of Engineering
School of Informatics & Engineering
Technological University Dublin, Blanchardstown
Dublin 15.

4th Year Project
29th of January, 2023

Table of Contents

Table of Figures

AI based Electronic Component Identifier

1. Details

- **Author**
  - B00125142 Violet Concordia
- **Supervisor**
  - Benjamin Toland
- **Course**
  - **Title**
    - Bachelor of Engineering (Honours) in Computer Engineering in Mobile Systems
  - **Year**
    - 4th
  - **Code**
    - TU807

2. Declaration

The material contained in this assignment is the author's original work, except where work quoted is duly acknowledged in the text. No aspect of this assignment has been previously submitted in any other unit or course.

Signed: _____ **Date:** _____

3. Abstract

- This report illustrates the development this 4th year Computer Engineering Project.
- The report hopes to successfully portray an accurate thought process and workflow of this project, in full detail, while giving credit where credit is due, and thoroughly explaining the choices made during the development of this project.
- The report has been organised into sections that each cover a wide topic, with table of contents to aid in smooth navigation of the contents.
- The project aims to develop an everyday utility tool for those that work with computers often, and hopes to improve their workflow, and ease the stress on their fingers, while providing full control of the experience to the user, by offering to the user to reprogram their device, using an intuitive GUI programmer.

4. Acknowledgement

- I am very grateful to Technological University Dublin for accepting me as a student, and providing me with the opportunity to take on this project.
- I'd like to acknowledge and express my gratitude towards my project supervisor, Benjamin Toland – who has taken on me and my custom project and has provided excellent guidance and feedback throughout the development of this project.
- I am also very grateful for the existence and availability of search engines, the primarily used for this project being Google. It is a valuable resource for research, even though you should not take every search result at face value, and ensure at least a few trusted sources agree with the findings.
- I would also like to thank my wonderful old and new friends and peers that I was able to meet thanks to my ability to attend TUD Blanchardstown Campus, and the quiet spaces provided for us to further our education with minimal interruptions.
- I am also grateful to my brother, Justas Bartnykas – who is also an engineer. I am grateful to him for introducing me to Qt Creator, which is now my go-to IDE for developement of C++ code for the past over 5 years now. I have also been provided access to a fairly expensive camera that he owned, that allowed the project to begin sooner than it otherwise could've.
- I'd like to offer my sincere apology and thanks to anyone else I may have missed that has contributed to this project in any way.

5. Introduction

5.1. *Acronyms*

- **IDE - Integrated Development Environment**
  - A sophisticated text editor, designed specifically for code development in certain languages.
  - Most IDEs today come with a high range of optional plugins that can be used to further increase production speed, and reduce redundant tasks via automation of said tasks.
- **YOLO - You Only Look Once**
  - An image detection architecture that the project is based on.
- **SIP - Single Inline Package**
  - Several of same type of component, all packaged in a single line.
    - Example: an SIP resistor, which features multiple resistors, all connected to a single ground pin.
- **AI - Artificial Intelligence**
  - A term often used in deep learning, which is the process of attempting to simulate intelligence that is similar to that of a human, by the use of a computer, in order to tackle issues that a standard style of computer operation either fails to, or performs at incredibly slow rates.
- **CPU - Central Processing Unit**
  - The component of a computer where the core computations are processed.
- **GPU - Graphics Processing Unit**
  - An optional component of a computer that is dedicated and optimised in computing graphical tasks.
- **LDR - Light Dependant Resistor**
  - A resistor that varies in resistance relatively to the amount of light the body of the component is exposed to.
- **LED - Light Emitting Diode**
  - An electrical component that emits light when current is passed through the circuit.
- **AC - Alternating Current**
  - Electrical current that oscillates.
- **DC - Direct Current**

    - Electrical current that stays constant.
- **CAD - Computer Aided Design**
    - CAD software accelerates and automates designs in various different fields.
    - Instructions are given to a computer that are translated into more complex and intuitive, usually GUI based interactive programs.
- **PCB - Printed Circuit Board**
    - A silicon board that has parts of it etched away, with only conductive tracks remaining in specific positions that are pre-planned using a CAD software.
    - Widely used to implement electronic circuits.
- **OCR - Optical Character Recognition**
    - Software based reading alphabetical characters from an image that contains written text.
- **RGB - Red Green Blue**
    - Commonly used to referred to a way of defining colors by their Red, Green and Blue properties.
- **HSV - Hue Saturation Value**
    - Commonly used to referred to a way of defining colors by their Hue, Saturation and Value properties.
- **NMS - Non-maximum Suppression**
    - A filtering technique used on predictions of object detectors.
    - Picks the smallest intersection of bounding boxes, according to their confidence values.

5.2. *The Problem*

Detection of objects from an image

- When we look at an image, we can immediately discern objects that are displayed, without ever having to think about it. It is effortless.
    - We use a combination of past experiences and deduction to determine information almost immediately.
    - This ability of ours as humans is thanks to millions of years of evolution, in a world where not detecting a potential threat makes a difference between life and death. It is in our subconscious nature to detect objects within a moment's notice.
- The problem arises when we want to implement a computer to process object detection for us.
- A computer is built upon the most simple of arithmetic tasks, combined together from an incredible amount of transistors that perform these tasks, into massive systems - which operate entirely in digital binary.
- Computers are designed to process computational tasks at complete precision and consistency.
    - A computer has no concept of learning from past experiences, nor any feelings that may affect its decisions. Given the same instructions, it will produce the same results on its first day of manufacture, and the last day of its operation (provided the unit was not damaged in a way that it would yield unpredictable results).
- When a computer is exposed to an image in form of a pixel grid, all it truly sees is numerical values that are assigned to each cell of the grid.
- Computers have no concept of color, let alone real life objects.
- The simple action of either moving or scaling an object completely changes the arrangement of the pixels that represent this object.

Examples

Figure 1: 220Ohm resistor: Body close up

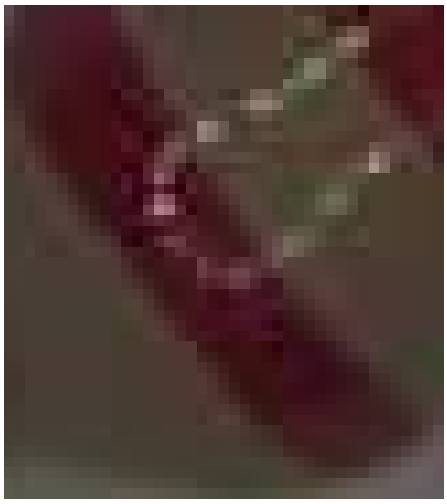


Figure 2: 220Ohm resistor: Single band close up

Figure 3: 220Ohm resistor: Single band close up, with arbitratry rotation applied

- The width and height of the image have both changed due to this rotation.
- The order of the pixels has changed drastically.
- To a person - this is no issue. It is obvious that this is still the same object. However, to a computer - the data has just shifted around entirely.

- While this is a complex issue to tackle already, it only becomes more complex when we want to detect something generic, such as a tree.

  A tree has many properties that may differ, such as: presence, color, shape, and size of leaves/needles, thickness, size, and color of trunk, branching styles, etc. While still being effortlessly identifiable by an intelligent creature.

Figure 4: Example of generic tree variations

Importance of object detection

- There are various cases in which automation of object detection as opposed to having a human constantly observing footage is beneficial.
- **Security**
  - In the field of security through digital cameras, object detection is an incredibly useful tool for monitoring potential intruders onto a facility.
    - For facilities that utilise dozens, or even hundreds of cameras - object detection is a very valuable tool that requires minimal human interaction, with high level of certainty, and 24/7 attention.
  - Depending on the security requirements, lower security facilities may not require hiring a person for constant monitoring of the security footage, and offers live notifications for any unexpected activity observed.
- **Production**
  - **Quality control**
    - During production of anything from farm produce, to electronic components - consistent detection and rejection of items with damage is essential.
    - With use of object detection, flaws can be recognised incredibly efficiently, and this information may be passed onto the production line, identifying exactly which item was detected to have flaws, and be discarded automatically, without ever requiring any human interaction.
    - If the detection is sophisiticated enough to be more reliable than a person, this opens up new opportunities for the speed and efficiency of the production, as a computer's computational power may be expanded, unlike a person.
  - **Analysis**
    - Thorough inspection of potential objects on final products is a crucial part of many fields of production.
      - Features that may've developed during the process of manufacture may be detected on the final products.
      - This includes positive, negative, or purely analytical features.
        - **Examples**

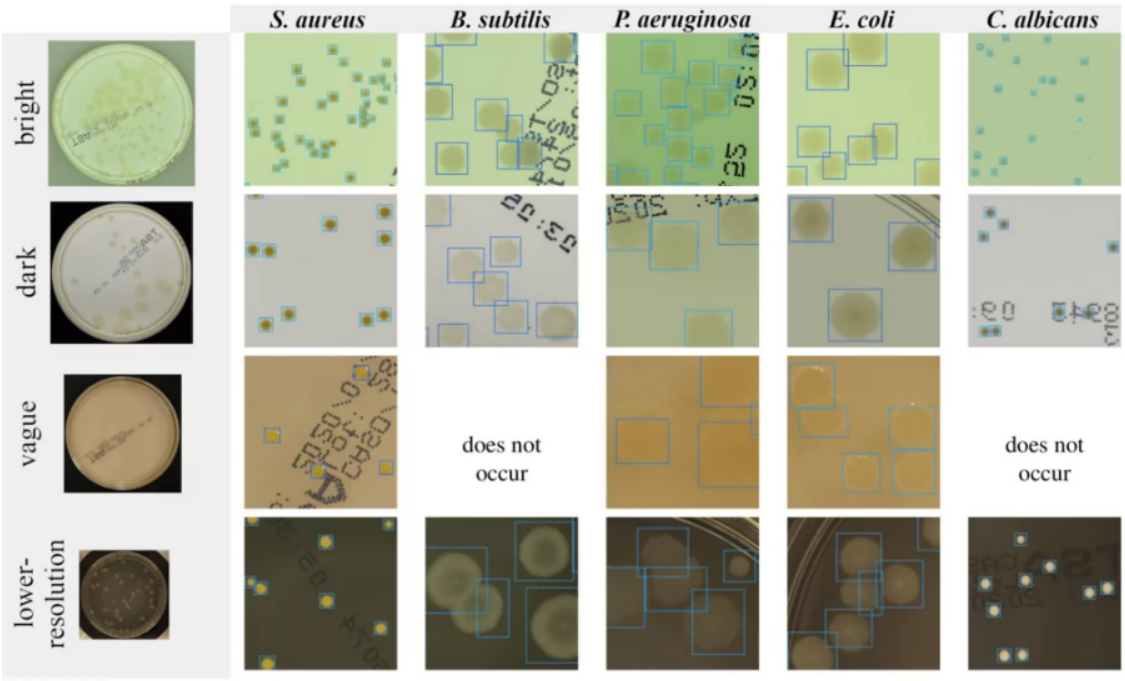


Figure 5: Analysis through bbject detection example: Petri dish [24]

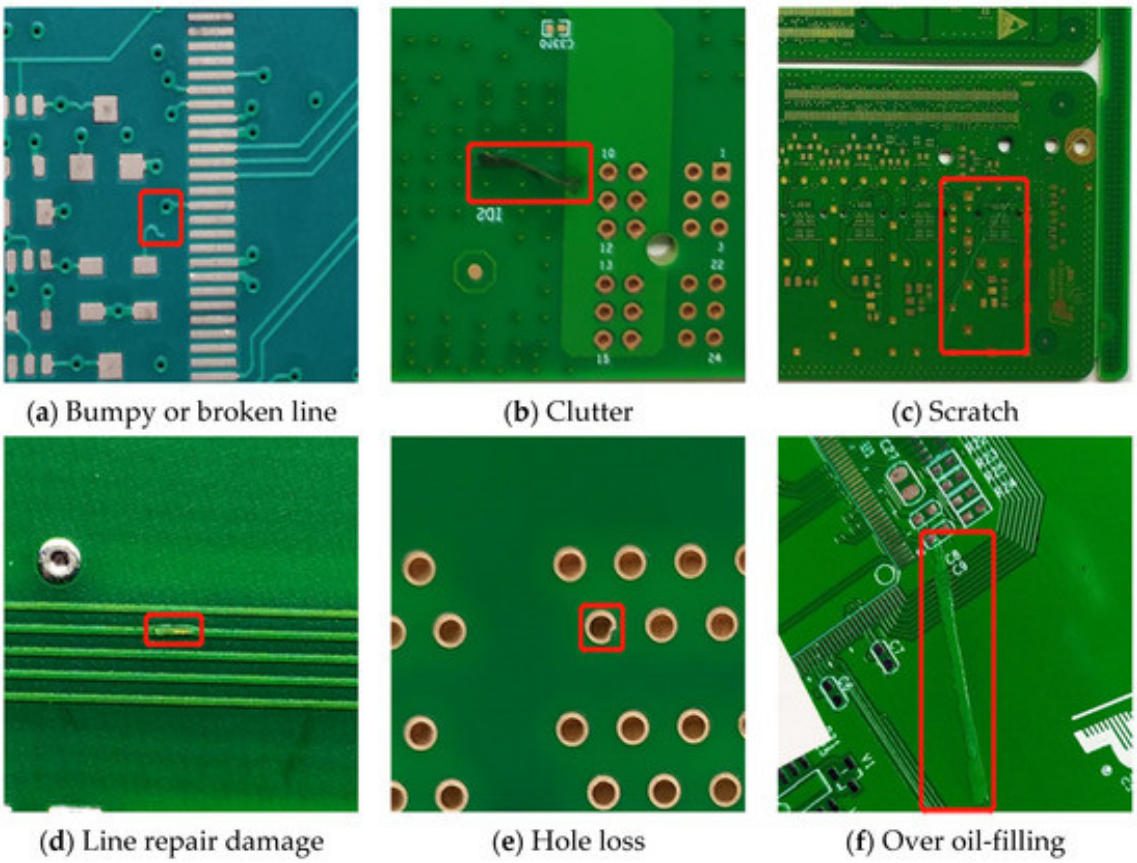


Figure 6: Analysis through bbject detection example: PCB manufacture [27]

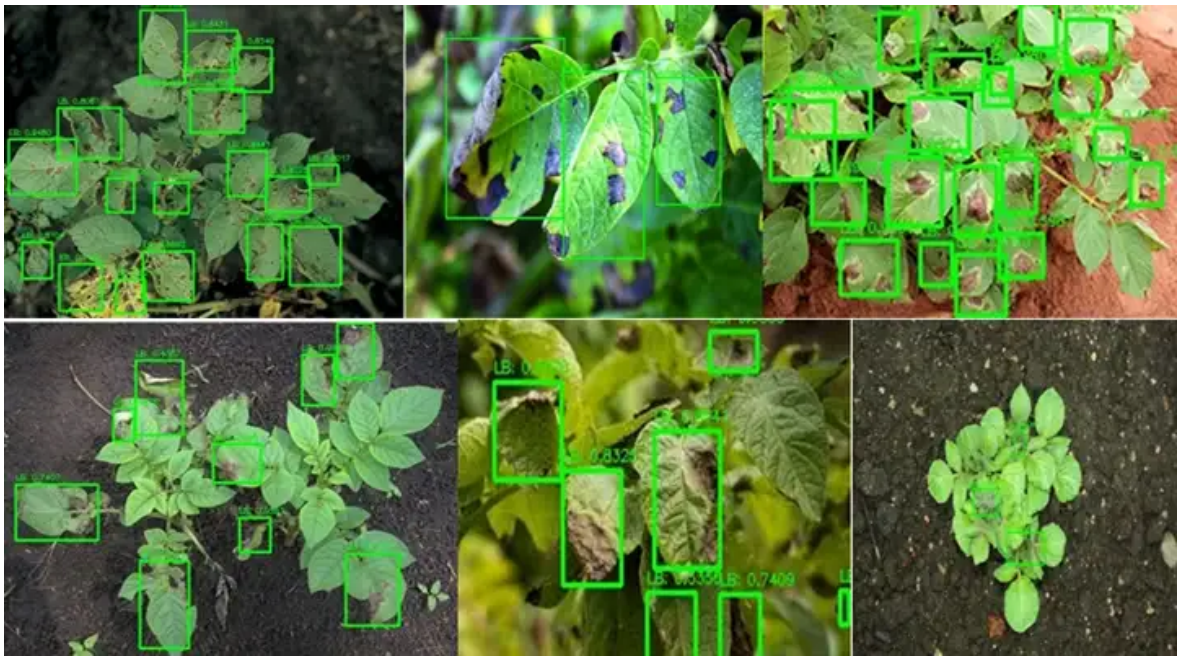


Figure 7: Analysis through bbject detection example: Crop disease [26]

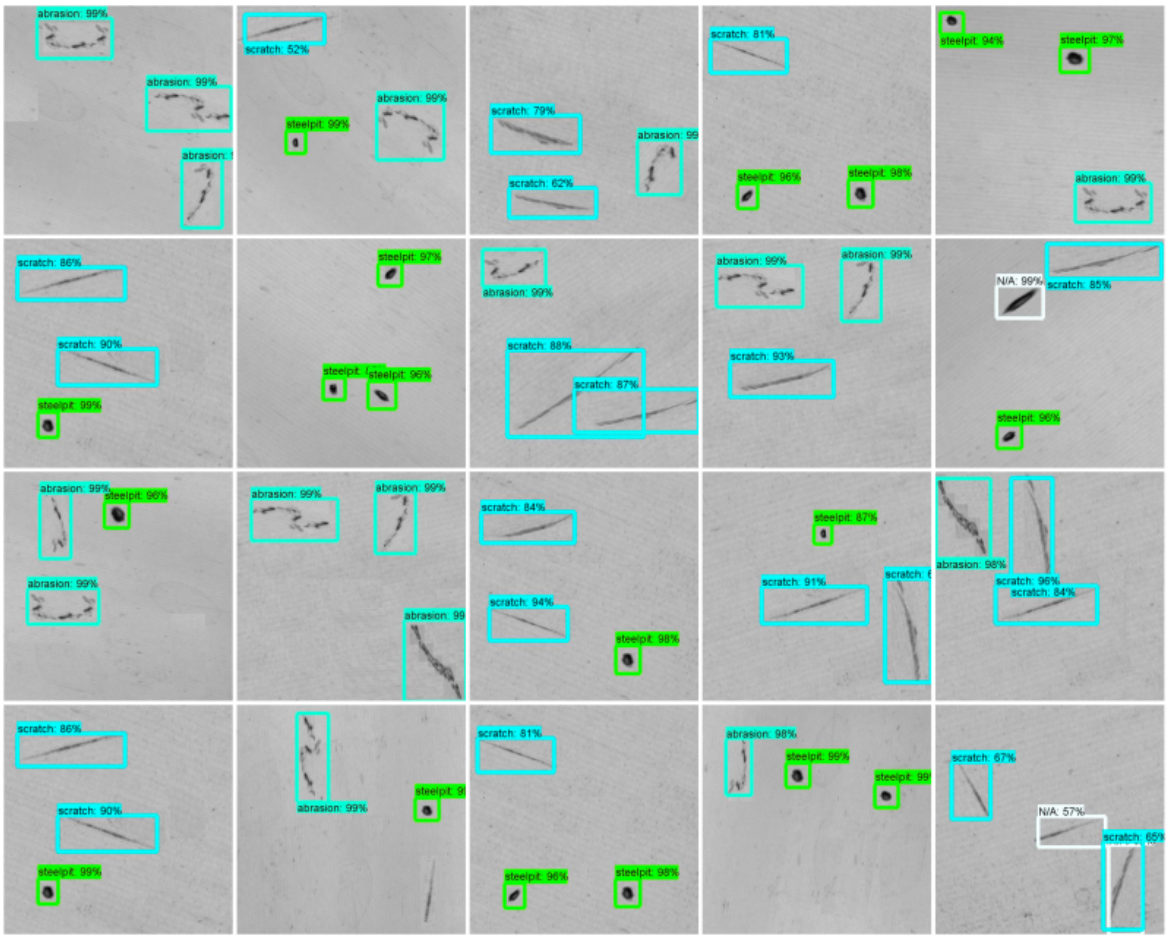


Figure 8: Analysis through bbject detection example: Surface Defection [25]

5.3. *Existing Solutions*

- **Algorithmic object detection**
  - Algorithm based detection can be used to effectively identify very specific criteria, which can be expressed as an analytical value, or a trend.
  - **Examples**
    - Specific color based properties.
    - Specific shapes by following line trends.
    - Must be coherent shapes, does not perform well with partial shapes.
    - Specific patterns.
  - However, it will struggle heavily at detecting and identifying any generic object.
    - **Examples**
      - Trees, cats, dogs, people, cars, etc.
- **AI based Object Detection**
  - **Neural Networks**
    - **Description**
      - Neural networks are an imitation of biological creatures intelligence, which is known as Artificial Intelligence. AI is designed to be ran on a computer.
      - Has the ability to be trained, which simulates past experience of biological intelligence.
      - A complex combination of usually many millions of digital neurons, with analog based values for each neuron, which result in a form of decision making based on a massive combination of criteria, rather than individual pixels.
      - These neurons work together to identify the incoming information and produce an output that resembles what the network has learned during the training of the model.
      - The networks are trained through deep learning.

- **Training using Deep Learning**
  - Training is usually based off of a pre-trained model, that is trained on a big dataset. Most pre-trained models are trained on the COCO dataset, which is publicly available, and holds a vast amount of data. This kickstarts the model with data that it can repurpose to use as a base.
  - Initially, the model parameters are set to a random state.
  - The output that it produces when fed input data, is of course, also random.
  - Epochs are ran on the model to train the model.
  - **Epoch**
    - The process of tweaking the entire model based on the existing parameters and the current output that it produces - by use of a loss function, randomness, and clever technique, in hopes of improving the detection ability of the data the model was trained upon.
    - The best performing model is kept between the newly trained model, and the previous best.
    - Many epochs are ran in order to polish the model as much as possible.
    - In the scope of this project, somewhere between 250 and 500 epochs will suffice.

- **Inference**
  - In object detection, inference is the utilisation of the model to process classifications of objects on the image.
  - Classification
    - The process of using the steps provided in the model to identify objects from an input image, through steps that depend on the architecture used.
    - The identified objects are marked with a bounding box, class they belong to, and confidence in the prediction value.

6. Description

6.1. *Summary*

6.1.1. Electrical component themed AI detection and identification.

6.2. *Features*

- **Detection of objects from an image via Inference**
  - Detect and display boundaries for each identified class, and the confidence value of this detection from the input image using Inference.
  - **Classes present in the dataset that the model will be trained upon**
    - Resistor
    - Singular
    - SIP Resistor
    - Diode
    - Capacitor
    - AC
    - DC
    - LEDs
    - Integrated Circuits
    - LDR
    - PCB terminal
- **Live Labeling**
  - The ability to take a snapshot of the current frame, defining appropriate labels, and saving this labeled snapshot for future training. All from inside the GUI.
  - Alternatively, taking snapshots of the GUI and saving them for later labeling.

6.3. *Milestones*

Sorted from highest priority, to lowest.

Each of these steps should be polished before continuing to the next one, to provide a solid foundation for the next step to be based on.

- **Base camera rig**
  - Setting up the camera on a rig.
- **Base GUI**
  - GUI with essentials to interface the camera through USB, with
    - A live display from the camera on the rig.
    - Ability to take images by pressing a button.
    - Support for running Inference.
- **Initial dataset gathering**
  - ~100 images of a single class, taken from the rig for initial training and testing of the model.
- **Initial inference model training**
  - For the purpose of testing inference on the rig.
  - The initial training should not take long at all, and does not require to be polished.
  - Training for ~100 epochs should be sufficient, with each epoch taking ~20 seconds on the machine available.
- **Inference running**
  - Proof of concept. The results will not be perfect as the dataset is minimal, and only contains 1 class.
- **Further dataset gathering**
  - At least 250 pictures of each class of every component that the project is designed to detect.
- **Further model training**
  - This training will take considerably longer than the initial training. Around 2 minutes per epoch, and should be ran for at least 300 epochs.
  - The goal is to reach 0.8 from range of 0 to 1 confidence values.
- **Post-processing**
  - Ability to gather further information from the detection bounding boxes provided by the inference.

- **Live training**
  - Optional: Ability to label the images from the device, without requiring external software.
  - It may be advantageous given the timeframe of the project to instead gather data during a session and labelling it afterwards.
    - Existing labelling related software offers quality of life features, such as rough auto labelling of the images, which only requires the user to adjust the bounding boxes and confirm their validity, rather than having to define the boxes from start to finish.
- **Testing with video footage from a mobile device**
  - After the previous steps are in good shape, investigation of moving the inference to a mobile device will begin.
  - If the confidence values are not up to standard, more data will be gathered from this and potentially other mobile devices, and further training will follow, until the results are adequate.
  - If adequate results are achieved before the deadline of this project, deployment to a mobile device will be started.
    - If the frame rates are not sufficient enough, the inference may be ran on still images to improve user experience.

7. Research

7.1. *Focus Audience*

- While this project may be retrained and refocused to be utilised for many different fields - it is trained for electrical component identification, which is focused towards engineers.
    - This project focuses on both existing engineers, and ones that are interested in becoming engineers.
        - Having access to the provided by the project quick identification of components, count of each, and any potential additional information saves time spent manually analysing this information.

7.2. *Architectures*

7.2.1. R-CNN [1]

- **Description**
  - R-CNN, which stands for Region Based Convolutional Neural Networks was released in 2013. As other object detection architectures, R-CNN takes an input image, and outlines bounding boxes where it believes an item of a certain class is present.
- **Developed by**
  - Ross Girshick
- **Disadvantages**
  - Not real-time.
  - On average, takes 47 seconds to process a single frame.
  - According to the Git repository, the project was seemingly abandoned about 5 years ago.
- **Discussion**
  - It should be noted that R-CNN has a successor called Fast R-CNN and Faster R-CNN.
  - However, even the fastest of the successors still barely manage 5 frames a second at best.
  - While 5 frames a second is an impressive and definitely useable result, there are alternative architectures that offer a significant improvement in inference time.

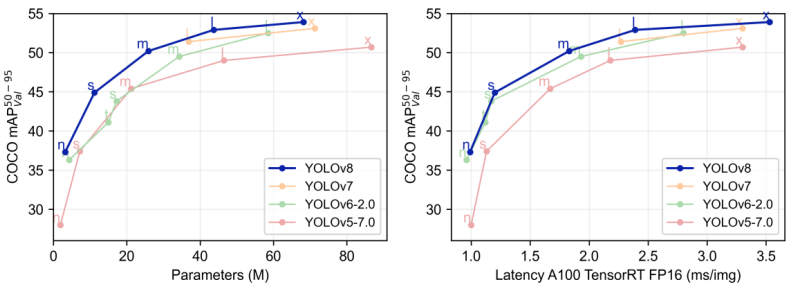
7.2.2. SSD [2]

- **Description**
  - SSD, which stands for Single Shot Detector. SSD was released in 2017
- **Developed by**
  - Max deGroot
  - Ellis Brown
- **Disadvantages**
  - According to the Git repository, the project was seemingly abandoned about 4 years ago.
- **Discussion**
  - Offers great framerates of an average of 45 frames per second when tested on a relatively old now graphics card, the NVIDIA GTX 1060.

7.2.3. Ultralytics YOLO [3]

- **Discussion**
  - YOLO, which stands for You Only Look Once is a popular image segmentation and object detection model that was originally developed by Joseph Redmon and Ali Farhadi.
  - As the name suggests, YOLO focuses on detection of multiple classes in a single "look", which is a single analysis of the entire input image.
    - An approach like this may seem too good to be true, and that it should come with signficant cost to the speed and confidence of the model.
    - But when the results are analysed - that could barely be further from the truth.
    - YOLO is an incredibly efficient and accurate architecture.
  - When compared to many other architectures before YOLO, realistically, no matter how quick the other architectures may be - this is an incredibly superior approach, as other architectures would approach detection by reanalysing the entire image for every single class that the model was trained for - increasing the time taken per detection additively per class.
  - These days most sophisticated architectures approach object detection similarly to YOLO, but YOLO is still a state-of-the-art architecture that continues to improve and grow to this day.
  - Because of how far ahead the YOLO architecture is when compared to most other architectures, is utilised very commonly throughout any object detection projects.

- **Brief History**
  - **Versions**
    - **YOLOv1**
      - The first version was released in 2015, and it very quickly became popular due to the significantly superior speed and accuracy when compared to other architectures.
    - **YOLOv4**
      - Released in 2018, Introducting of Mosaic data augmentation, and a new and improved loss function - decreasing time taken to achieve better results for the trained model.
    - **YOLOv5**
      - Released in 2020, Introducing support for Object Tracking - which allows following a moving object, and Panoptic Segmentation, which allows identification of overlapping objects, with accurate bounding boxes.
    - **Ultralytics YOLOv8**
      - The latest version of YOLO as of today. YOLOv8 is a state-of-the-art model that builds upon the already very successful previous YOLO versions, introducing new performance and flexibility features.
      - Full support for previous YOLO versions, making it incredibly convenient for existing users of previous YOLO versions to take advantage of the new features.
    - **Comparison**



  - **In general, YOLOv8 is superior to all of its predecessors.**
- While YOLOv5 is mostly underperforming when compared to the next versions, it is important to note how incredibly minimal the delays are even on a version so outdated now.

7.2.3.1. Pre-Trained Models

- YOLO offers pre-trained models that are used to start train custom models.

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Figure 9: YOLOv5 pre-trained model choice

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 10: YOLOv8 pre-trained model choice

- **Analysis of the models**
  - Model properties
    - **Size**
      - The pixel height and width the model operates up to.
    - **mAP**
      - Single-model single-scale values while detecting on the COCO val2017 dataset.
    - **Speed**
      - Averaged time taken using the Amazon EC2 P4d instance on the COCO dataset.
    - **Params (In Millions)**
      - The number of parameters that are tweaked per epoch while training, and processed during inference.
    - **FLOPS**
      - Floating Point Operations Per SecondA measure based on Floating Point Operations that is relevant in the field of Deep Learning.
    - Observations
      - In comparison of YOLOv5 and YOLOv8 versions - a clear advantage can be seen when taking into account the size of the model (param count), and the resulting mAP output, as well as the time taken.
      - Each model has its advantages and disadvantages, and should be picked depending on the project.
      - Diminishing results can be observed on the mAP values when compared to the time taken (Speed).
- In some circumstances, maximum precision is essential, and is prioritised over the hardware requirements. This is when a higher model should be chosen.
- In the scope of this project - several pre-trained models have been used, including both YOLOv5 and YOLOv8, for the purpose of comparison.

7.2.3.2. Internal steps of the You Only Look Once Inference [4]

Figure 11: YOLO detection: Input Image

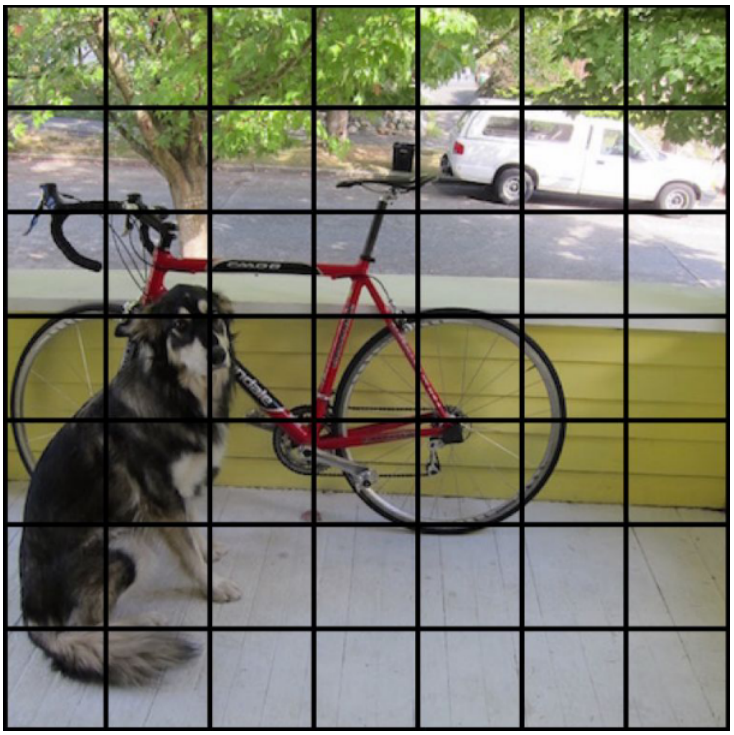


Figure 12: YOLO detection: Splitting into S by S grid, S being 7 in this figure.

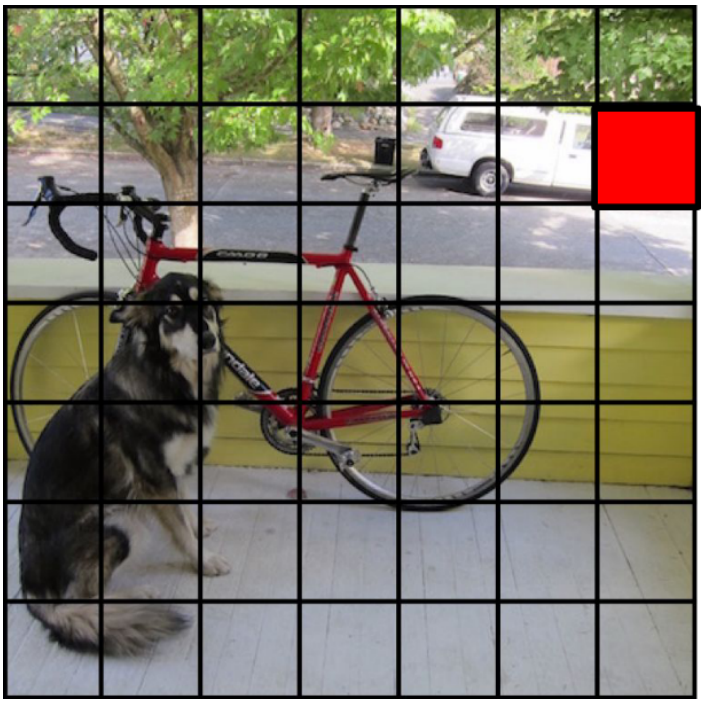


Figure 13: YOLO detection: Each cell predicts the bounding boxes and confidence values of each box.

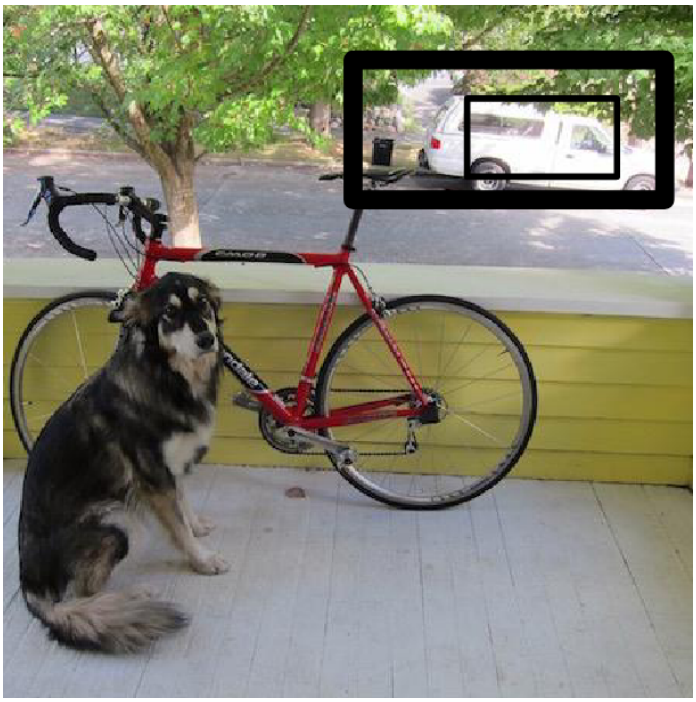


Figure 14: YOLO detection: Identified bounding box

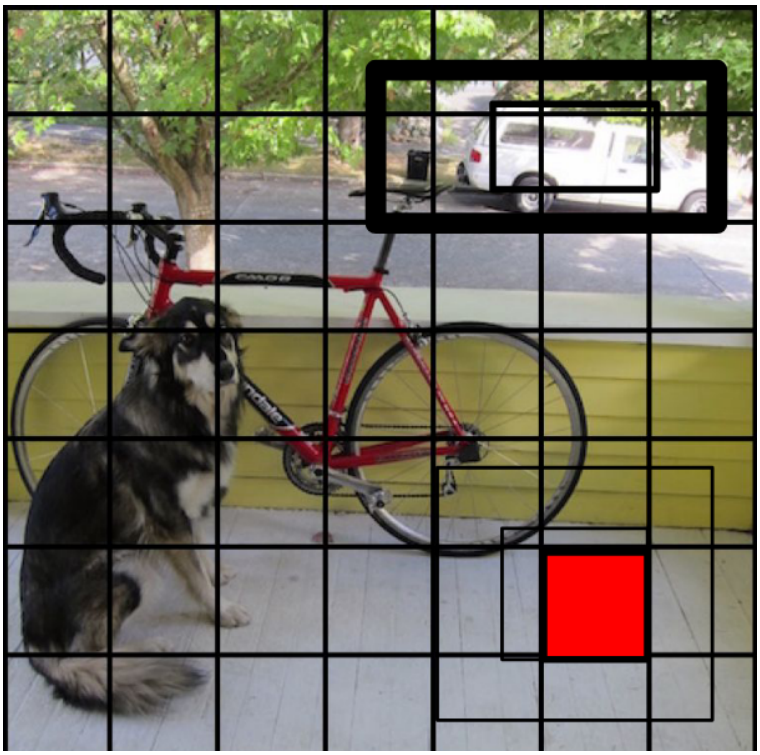


Figure 15: YOLO detection: Previous steps being repeated for each cell.

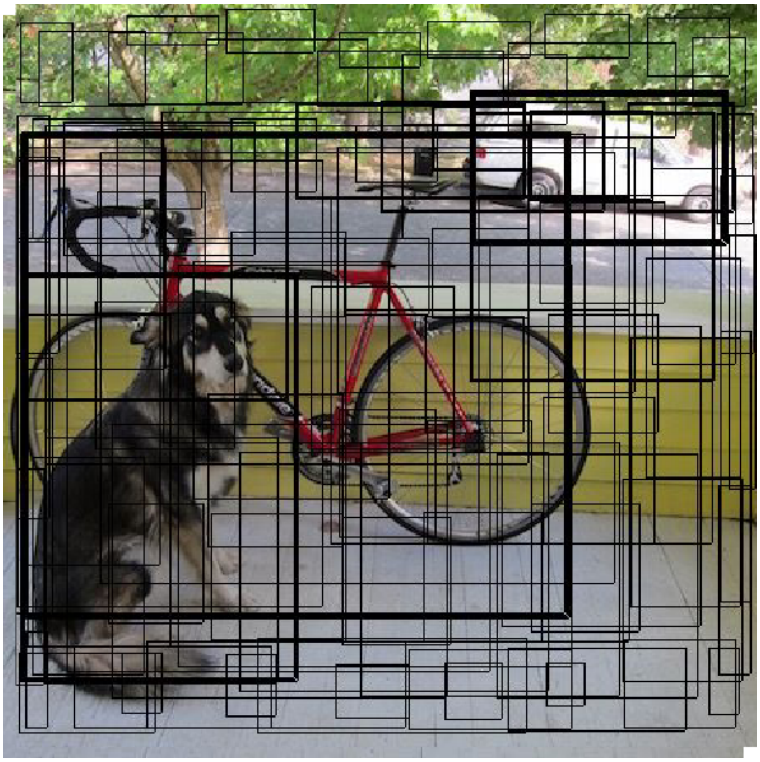


Figure 16: YOLO detection: All identified bounding boxes.

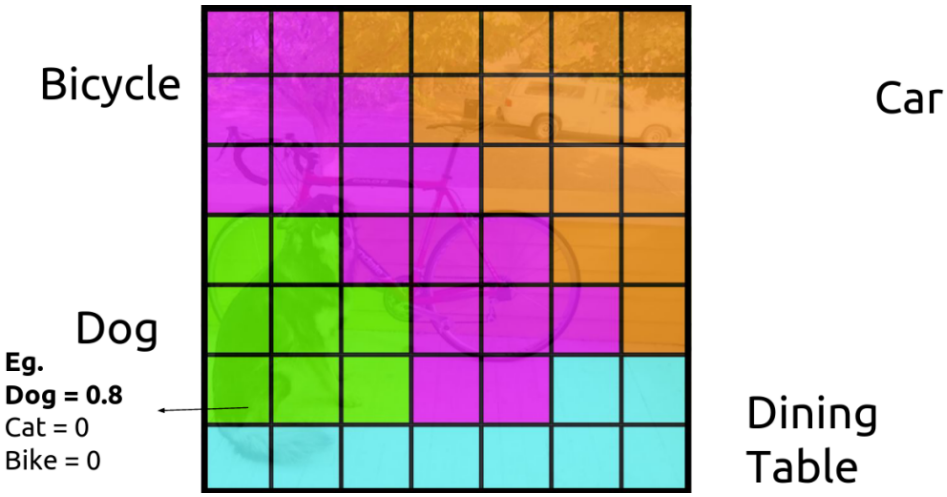


Figure 17: YOLO detection: All identified probabilities for each grid cell.

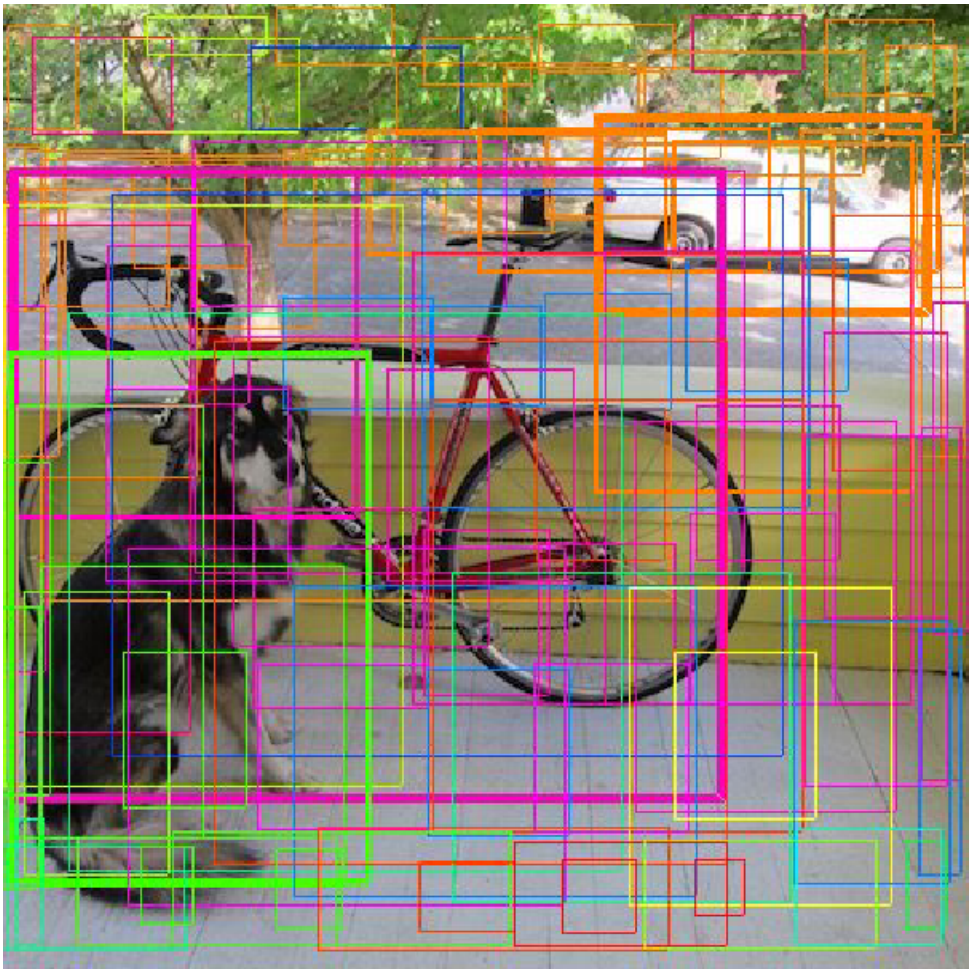


Figure 18: YOLO detection: Each bounding box is "shaded" in the case of this example with how much each one covers of which probability grid.

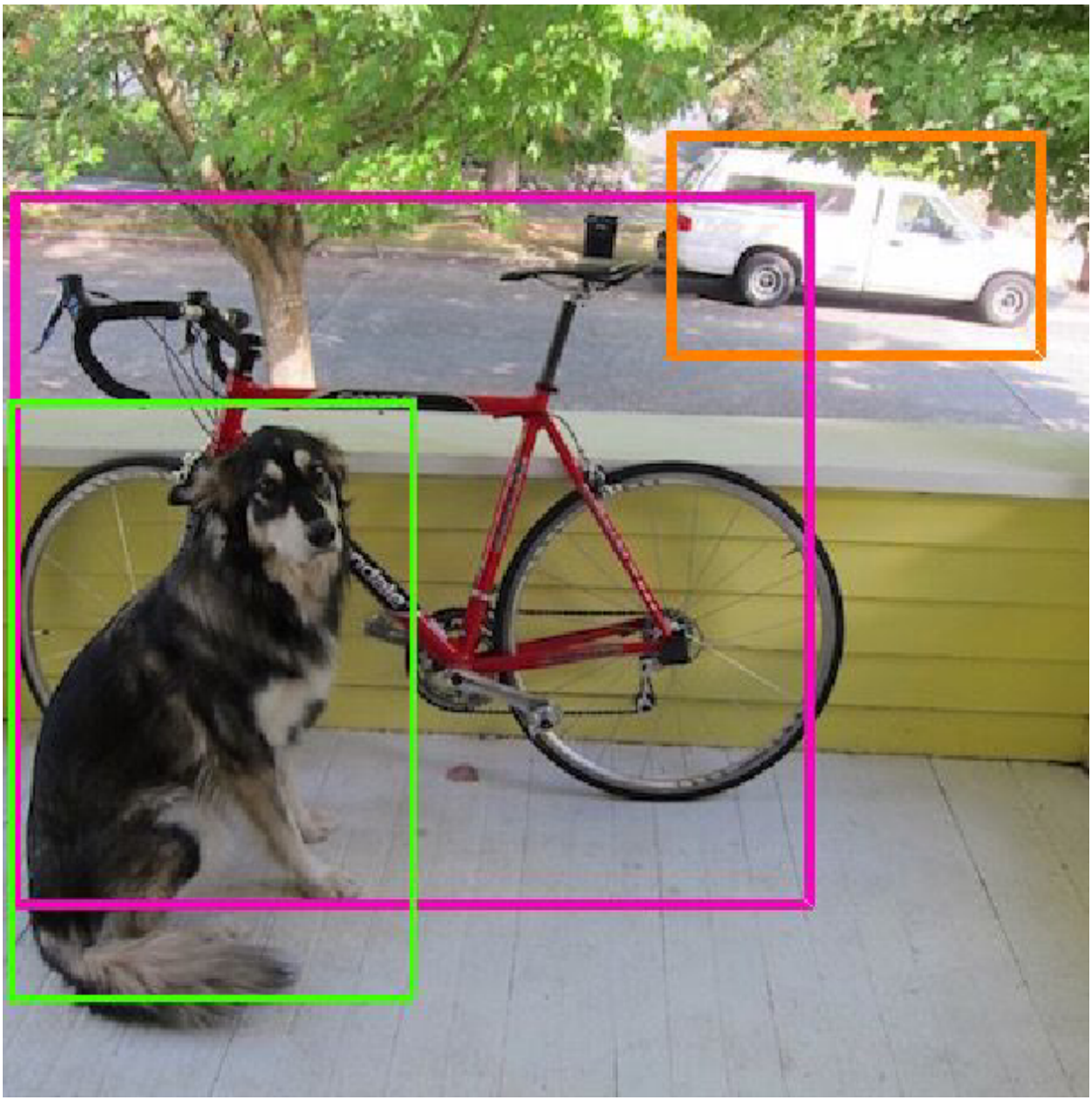


Figure 19: YOLO detection: The bounding boxes are reduced using NMS. This is the final step.

- This is the final output that YOLO provides: Bounding boxes with classification (In this example, the classification is marked by a color. The real output is a string.), and the confidence value (In this example, confidence is marked by the opacity of the boxes. The real output is a number ranging from 0 to 1).

7.3. *Rationale*

- Architecture choice
  - YOLO has been chosen as the architecture that this project utilises for the AI detection.
    - At the start of the project, there was already a high bias towards YOLO due to the highly positive past experience with YOLOv5 and all the incredible features that it offers.
    - Upon release of YOLOv8 and all the superior features and specifications that it provides on top of the previous versions - the YOLO family was an obvious choice in the architecture that will be used for the project.

7.3.1. Set Rig

- **Training**
  - **Angle range**
    - Due to the angle and lighting both being known and mostly set thanks to using a set rig, the input dataset does not need to cover angles and lighting outside what the rig will expose it to during runtime.
    - The angle range is reduced only to looking from top to down, eliminating the rest of the angle range.
  - Only the components being detected need to be trained in all angles, as opposed to the camera gathering the dataset requiring to be positioned in different angles.
  - **Lighting**
    - While the lighting will change depending on the room conditions, the ring light around the camera will provide significant consistency in lighting.
    - While this does not not eliminate the necessity to train against various lighting conditions, it does reduce their significance and increase certainty of the detection.
    - Having a top to down view also eliminates the majority of issues that come with glare from high luminosity bodies, such as clouds or the sun.
    - A set rig significantly limits the distance that the objects will be from the camera during runtime, allowing for further confidence in the predictions.

    - **Static background**
        - Apart from dust or unexpected objects present on the rig's surface, which should be removed before usage - the background that the objects are in front of will stay mostly consistent.
        - This reduces the necessity to gather data of the same object under backgrounds that are not expected to be used during runtime.
        - The sum of all the points covered above results in a significant reduction in data required to train when compared to a setup without a set rig, for equivelant confidence values during runtime.
        - The reduction of data required to train makes it feasible to train relatively high quality models from data gathered and trained from home.
- **Running**
    - The set position of the camera, a significant reduction in distance between the objects, and significant consistency of the lighting provided by the ring light, and the static background - will boost the confidence of the inference considerably.

7.4. *Hardware*

7.4.1. Training

- **Rented Dedicated Server**
  - **Advantages**
    - **Speed**
      - Utilises multiple GPUs - Quicker epoch computations, resulting in quicker training.
    - **Cloud based**
      - Allows for parallel computing, as opposed to using your personal computer at home, which will slow down any work required to be done on the computer.
  - **Disadvantages**
    - Setup time
    - Cloud based - upload and download times.
      - Datasets tend to be considerably big in size.
      - A smaller dataset of ~2000 images takes up ~3gb of space.
      - This is not a significant amount of data for a local machine to transfer, but it is a considerable amount for uploading.
  - **Cost**
    - The bigger the server - the higher the rates become.

- **Personal Computer**
  - **Advantages**
    - **Setup time**
      - Local - Provided a local machine is already owned, it is immediately available.
    - Pictures are taken from the machine itself. No upload/download times.
    - **Cost**
      - As opposed to a rented server - acquiring your own machine has the benefit of owning the machine, and being able to use it indefinitely (Or until it eventually breaks.)
      - While the initial cost of acquiring an adequate machine for deep learning is higher than renting a server for a few months, it is a worthwhile long-term investment into a machine that can be used for a variety of casual or intensive tasks.
  - **Disadvantages**
    - **Speed**
      - A local machine will likely contain one, maybe two GPUs.
      - When compared to a sophisticated server that runs many GPUs - a local machine will most likely process the training at a slower rate than a dedicated server would.

7.4.2. Inference

- **Discussion**
  - After the training is done, which is usually over the span of 10's, and sometimes 100's of hours, depending on the size of the dataset and the epoch count - Running the trained model for inference only takes time in the range of milliseconds to process a single frame.
  - How long specifically is directly tied to the speed of hardware that the model is being ran on, and the size of the model.
  - Even with all the speed optimisations offered by the YOLO family, a lower end device such as a Raspberry Pi 4 may take 1-2 seconds to process a single 360p image.
  - It is important to pick appropriate hardware for your particular use cases.

7.4.2.1. Devices

- **Microprocessors**
  - **Raspberry Pi 4 [5]**
    - **Specifications**
      - **CPU**
        - **Core Count**
          - 4
      - **Maximum Frequency**
        - 1.5GHz
      - **GPU**
        - **Core Count**
          - 4
        - **Maximum Frequency**
          - 700MHz
  - **Beaglebone [6]**
    - **Specifications**
      - **CPU**
        - **Core Count**
          - 1
      - **Maximum Frequency**
        - 1GHz
      - **GPU**
        - **Core Count**
          - 2
        - **Maximum Frequency**
          - 532MHz
  - **Nvidia Jetson Nano [7]**
    - **Specifications**
      - **CPU**

        - **Core Count**
            - 4
        - **Maximum Frequency**
            - 1.479GHz
    - **GPU**
        - **Core Count**
            - 128
        - **Maximum Frequency**
            - 921MHz
- **Observations**
    - This microprocessor is targeted towards quick graphical computations, which can instead be used for deep learning.

- **USB computation extensions**
  - **Intel Neural Compute Stick 2 [8]**
    - **Ease of access**
      - Due to the device being specialised for neural computations, it is not a common device by any means.
      - Combined with the price tag of ~100 eur, this device will likely only be owned by developers, as opposed to users.
      - As this device is unlikely to be owned by a user of the project, it would not be wise to require owning one to run our inference.
    - **Discussion**
      - The project will be able to support a compute stick as an alternative to a GPU.
      - **Advantage**
        - Offers computational power through a USB connection - can be used to run Inference on existing devices, such as a laptop.
    - **Specifications**
      - **Core Count (SHAVE)**
        - 16
      - **Processor Base Frequency**
        - 700MHz
      - **Memory**
        - 2GB

- **Android Phone [9]**
  - **Specifications**
    - Specifications depend on the specific device.
      - There are countless types of android devices on the market, all with varying specifications.
    - **Camera**
      - The vast majority of mobile phones on the market today have a built-in camera.
  - **Discussion**
    - **Ease of access**

        - Widely and easily accessible.
        - Most people own a mobile device, and have it on them in most cases.
        - The app may be obtained from an App Store, that mobile devices have easy access to, as long as they have access to the internet.

- **Windows [10]/Linux [11]/Mac [12] Desktop/Laptop Machine**
    - **Specifications**
        - Specifications depend on the specific device.
            - The specs of a desktop/laptop machine will most likely beat the specs of both a phone, and a microprocessor.
    - **Discussion**
        - **Ease of access**
            - Desktops are widely accessible in environments where it would be relevant to use this project, such as the home of the user, or the campus a student is in.
            - The machine must have permissions for USB connections and running the application.
    - **Platforms**
        - Cross-Platform
            - The application is designed through Qt Creator [13].
                - Qt Creator is cross-platform.

7.5. *Software*

7.5.1. Identification

- **Description**
  - Post-processing of the components in the bounding boxes detected by inference, which may have additional information that can be identified by a variety of approaches.

Marking Codes

- **Resistors**

Color	Color	1st Band (x10Ω)	2nd Band (Ω)	3rd Band (Multiplier)	4th Band (± Tolerance)
Black		0	0	$x10^0$	20.00%
Brown		1	1	$x10^1$	1.00%
Red		2	2	$x10^2$	2.00%
Orange		3	3	$x10^3$	3.00%
Yellow		4	4	$x10^4$	100.00%
Green		5	5	$x10^5$	0.01%
Blue		6	6	$x10^6$	0.25%
Violet		7	7	$x10^7$	0.10%
Grey		8	8	$x10^8$	0.05%
White		9	9	$x10^9$	10.00%
Gold				$x10^{-1}$	5.00%
Silver				$x10^{-2}$	10.00%

Figure 20: Resistor Color Code Table

Figure 21: 220Ohm resistor

Color codes

Brown = 1

Red = 2

Gold = 5% tolerance

- **Calculations**

$$Resistance\ (Ohm) = (10^{*}(1st\ band) + (2nd\ band)) * 10^{(3th\ band)}$$

$$(10^{*}(2) + (2)) * 10^{1}$$

$$220Ohm$$

$$220 \pm (5\%^{*}220)$$

$$5\%^{*}220 = 11$$

$$209\ to\ 231Ohm\ Resistor$$

- **Capacitors**

	1st Number (x10pF)	2nd Number (pF)	3rd Number (Multiplier x10n)	Capacitance (pF)
Examples	1	0	0	10
	3	3	1	330
	1	5	2	1500

Figure 22: Capacitor Code Table

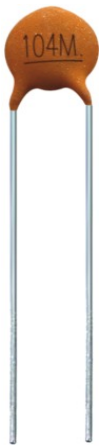


Figure 23: 100nF Capacitor

- **Calculations**

$$Capacitance\ (pF) = (10^{*}(1st\ number) + (2nd\ number)) * 10^{(3th\ number)}$$

$$(10^{*}(1) + (0)) * 10^{(4)}$$

$$100000pF$$

$$100nF$$

- **Integrated Circuits**
  - Unfortunately for the purposes of automatic identification of Integrated Circuit markings, most IC manufacturers do not follow any global standard for marking their ICs.
  - Most manufacturers tend to have their own internal IC marking standards.
  - Due to this fact - only known markings can be used to identify components.

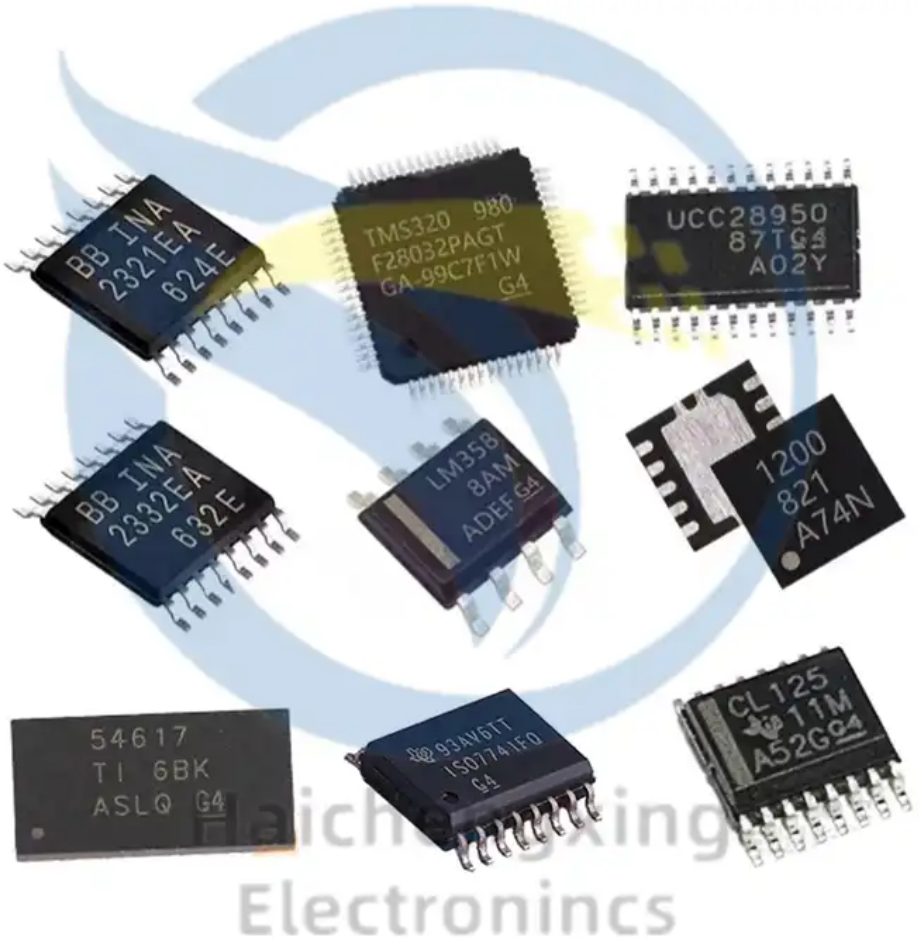


Figure 24: Variety of labeling on components that originate from different IC manufacturers.

7.5.1.1. Examples

- **Input image**

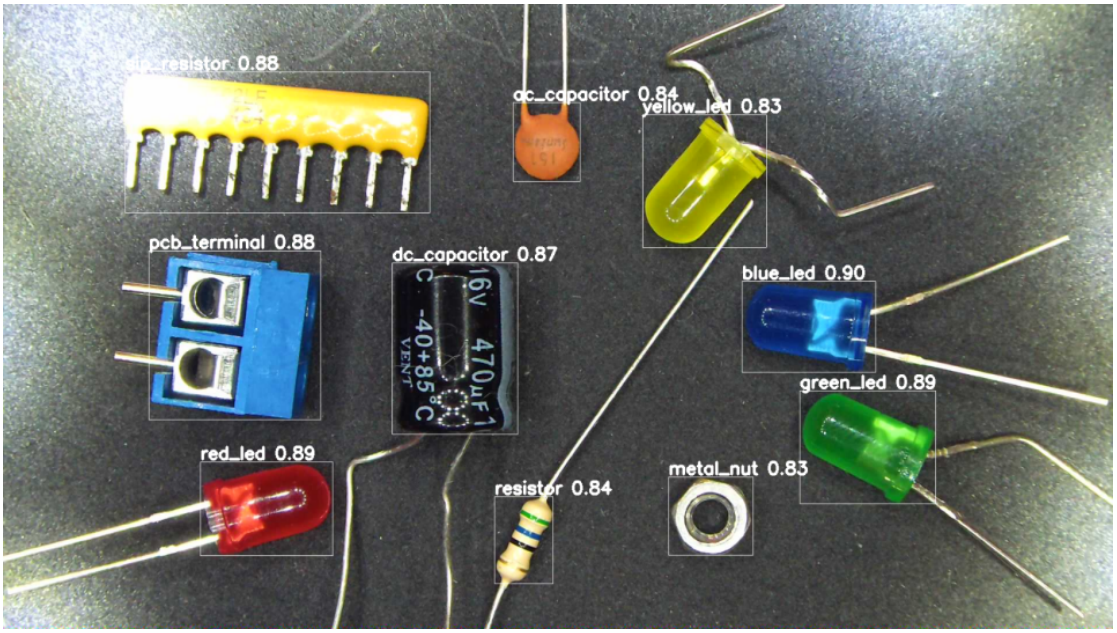


Figure 25: Example of 3rd trained model running Inference.

- **LEDs**
  - **LED color**
    - **Inference method**
      - Different color LEDs may be trained as individual classes.
      - Has the disadvantage of requiring training for each individual LED separately, as opposed to one generic LED.

- **Algorithm method**
  - The most prominent color may be identified by sorting all the colors from the image into their hue values, and checking which hue is most active.
  - Has the advantage of working on any LED.

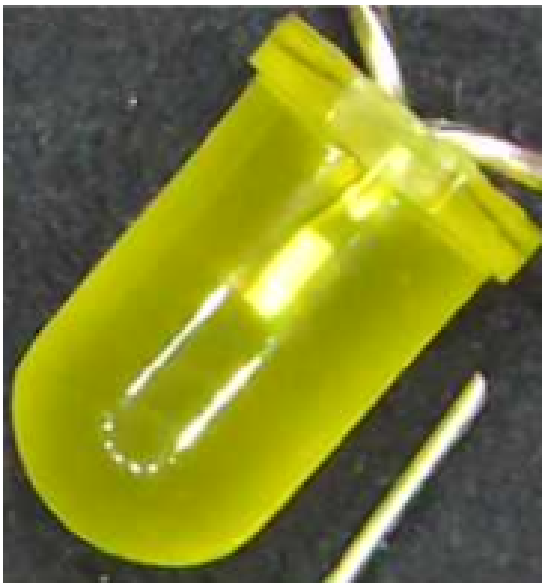


Figure 26: Color extraction example: Raw input image

- **Contrast approach**
  - Has the disadvantage of potentially giving false information if the background is too vibrant.

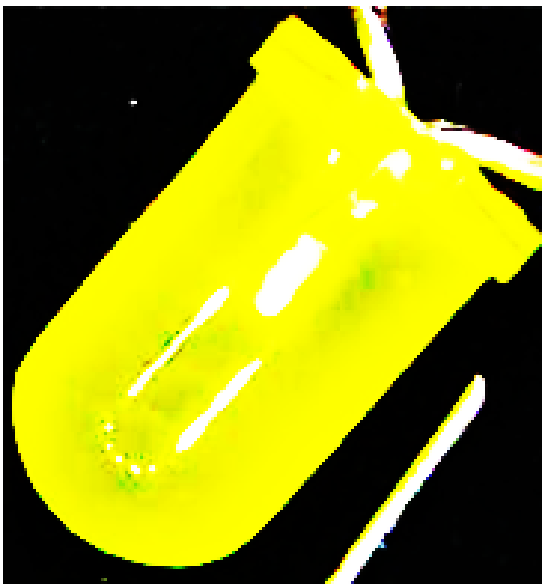


Figure 27: Color extraction example: Heavy contrast applied

- **Approaches after filtering**
  - **Colors histogram**

Figure 28: Color extraction example: Color Histogram

    - Results show clearly prominent yellow, which is accurate.

- **HSV analysis**

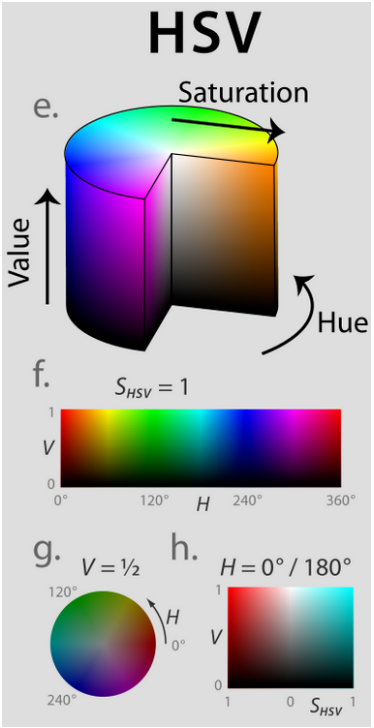


Figure 29: HSV chart overview

  - HSV, or Hue Saturation Value, is an alternative way to represent colors.
  - It can be advantageous over RGB in situations such as this.

- Taking the average of all the pixels hue values that have a value above a certain threshold. Around 0.7 on a range from 0 to 1 should be appropriate.
- Note: This example would ignore colors that are darker than 0.7, on a range of 0 to 1.
- Hue is in the range of 0 to 360 degrees.

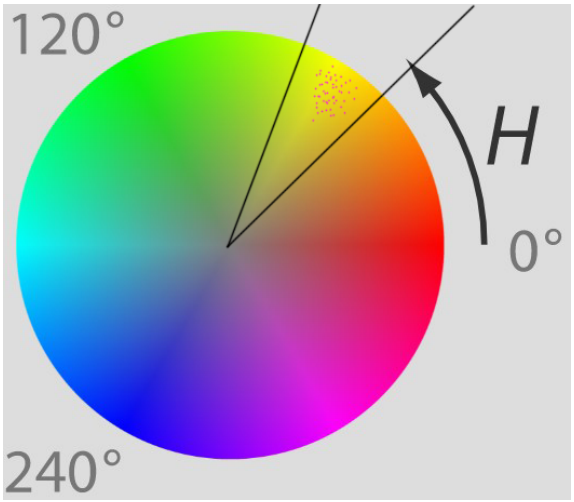


Figure 30: A circle hue chart with the data represented as pink dots.

- The pink dots represent the pixel values obtained from the previous step.
- Taking the average of this data, the result would land in the degree value that can be easily determined as yellow, by separating the hue circle into sections of colors by degrees ranges.
- Yellow is between 72° and 108° degrees on the hue circle.

- **Resistors**
  - Resistor code value
    - The color codes can be identified by processing the image using filters and otherwise until only the prominent colors remain.
    - Then, the positions of the color codes relative to the body of the resistor can be used to identify the specific positions and order of the color codes.
    - These can be processed into the actual ohm value.
- **IC Components**
  - **Pin count**
    - The pin count can be identified by processing the image using filters until there is a clear contrast between the body of the chip, and the pins.
    - One approach that could help identify the number of pins would be drawing a line between two of the pins and seeing how many of the pins touch this line. Taking the line that touches the most pins would provide the pin count of this IC.
  - **Information written on the component**
    - OCR may be used to extract the text based information.

8. Technology

8.1. *Hardware*

- **Rig**
  - A mounted camera above a surface (part of the product).
    - Produces a controlled environment live feed for the application.
- **Model Training via Deep Learning**
  - **Machine used**
    - **Personal Computer**
      - **CPU**
        - AMD RyzenTM 7 5800X3D [14]
        - **Brand**
          - AMD [15]
        - **Name**
          - Ryzen 7 58700X3D
        - **Core count**
          - 8
        - **Thread count**
          - 16
        - **Base clock frequency**
          - 3.4GHz
        - **L3 Cache**
          - 96MB
        - **Maximum operating temperature**
          - 90°C

- **Memory**
  - **Brand**
    - Corsair [16]
  - **Name**
    - Vengeance RGB PRO SL [17]
  - **Capacity**
    - 2x16GB
  - **Type**
    - DDR4
  - **Frequency**
    - 3.6GHz
- **GPU**
  - GeForce RTX 3060 Ti [18]
  - **Brand**
    - NVIDIA [19]
  - **Series**
    - 30
  - **Name**
    - RTX 3060Ti
  - **Memory**
    - **Capacity**
      - 8192MB
    - **Type**
      - GDDR6X
  - **Base clock frequency**
    - 1.41GHz
  - **CUDA core count**
    - 4864

- **Technology utilised**
  - **CUDA**
    - Special cores that are designed for compute-intensive tasks.
    - These run parallel with the CPU, and may also run parallel with multiple GPUs.
    - They are perfect for deep learning, as deep learning is incredibly compute intensive.
  - Deep learning training times are predictable, and stay mostly constant between epochs.
  - This means that there are no race conditions, and the more processing power available, the quicker the epoch will finish.
  - Deep learning computation with CPU Cores and GPU CUDA Cores running in parallel.

8.2. *Software*

8.2.1. Training

- **Labels**
  - Labeling is an essential part of training for Object Detection.
  - Labels are bounding boxes that determine what object is present where in the input dataset, in order for the model to do its best to detect them.
- **Training versus Evaluation**
  - The dataset is separated into two categories - one on which the model is being adjusted on, and another on which the model is being ran to determine the new confidence values achieved by the alterations.
  - Both datasets must be labeled for fully automated evaluation.

- **Augmentation**
  - **Description**
    - Modification of the provided data to simulate differences in the environment that would occur in a real life scenario, and to provide a challenge in form of imperfections to both train against, and test against.
- **Augmentation examples**
  - **Rotation**
    - Introduction of random rotation, with respect to the label position. Specified in a 0 to 360° range.
    - Simulates real life scenario of a different angle the feed is provided in.
  - **Blurring**
    - Introduction of random spots of blur. Specified in frequency and intensity.
    - Simulates real life scenario of change in focus, fog, and smudges on the camera lens.
  - **Resizing**
    - Introduction of random scaling of the image, at a specified frequency and range of scaling.
    - Simulates real life scenario of distance. Upclose objects will cover a far bigger pixel area in an image than a far away one would, and should be trained against this to prevent detection of only certain distance away objects.
  - **Joining up of multiple images to create new ones**
    - Cutting and joining of images into new images.
    - Creates additional images from the existing dataset that appear unique, making most of the existing dataset, with only a slight devalue in information stored in regards to training.
  - **Addition of glare**
    - Introduction of random glares, of specified frequency and intensity.
    - Simulates bright objects interacting with the lens, ensuring the model does not get confused about glare in real life scenarios.
  - **Addition of spots**
    - Introduction of random spots and smudges.
    - Simulates dust, dirt, and other particles that may be present in a real life scenario, ensuring the model does not get confused by a partial coverage of an object.

- **Visual examples**

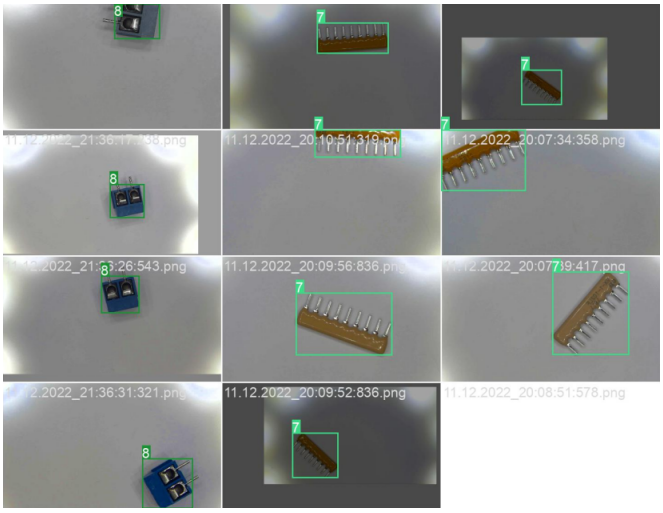


Figure 31: Mild augmentation example 1

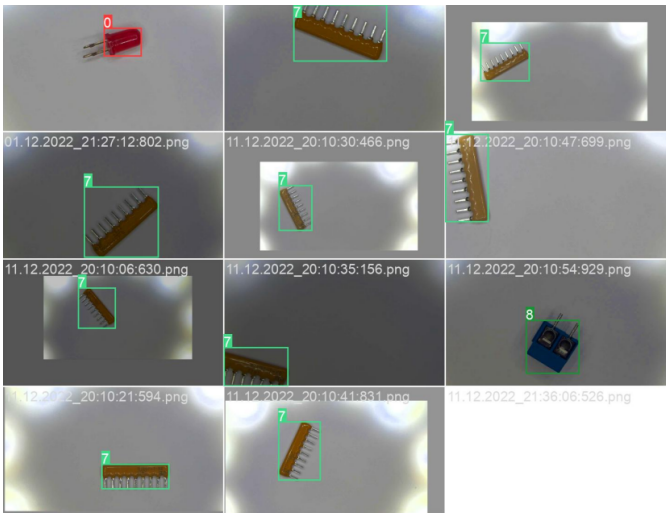


Figure 32: Mild augmentation example 2

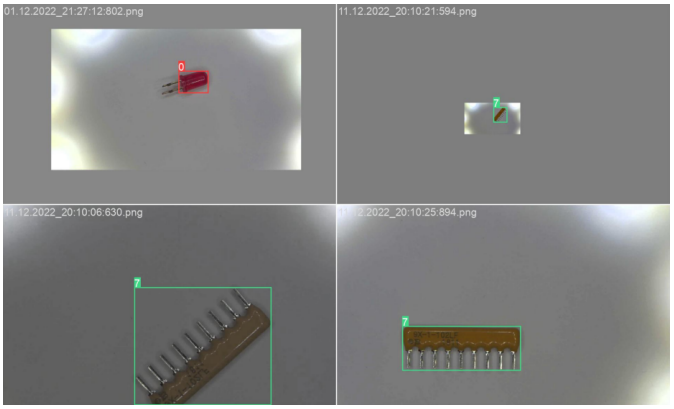


Figure 33: Mild augmentation example 3

8.2.2. Inference

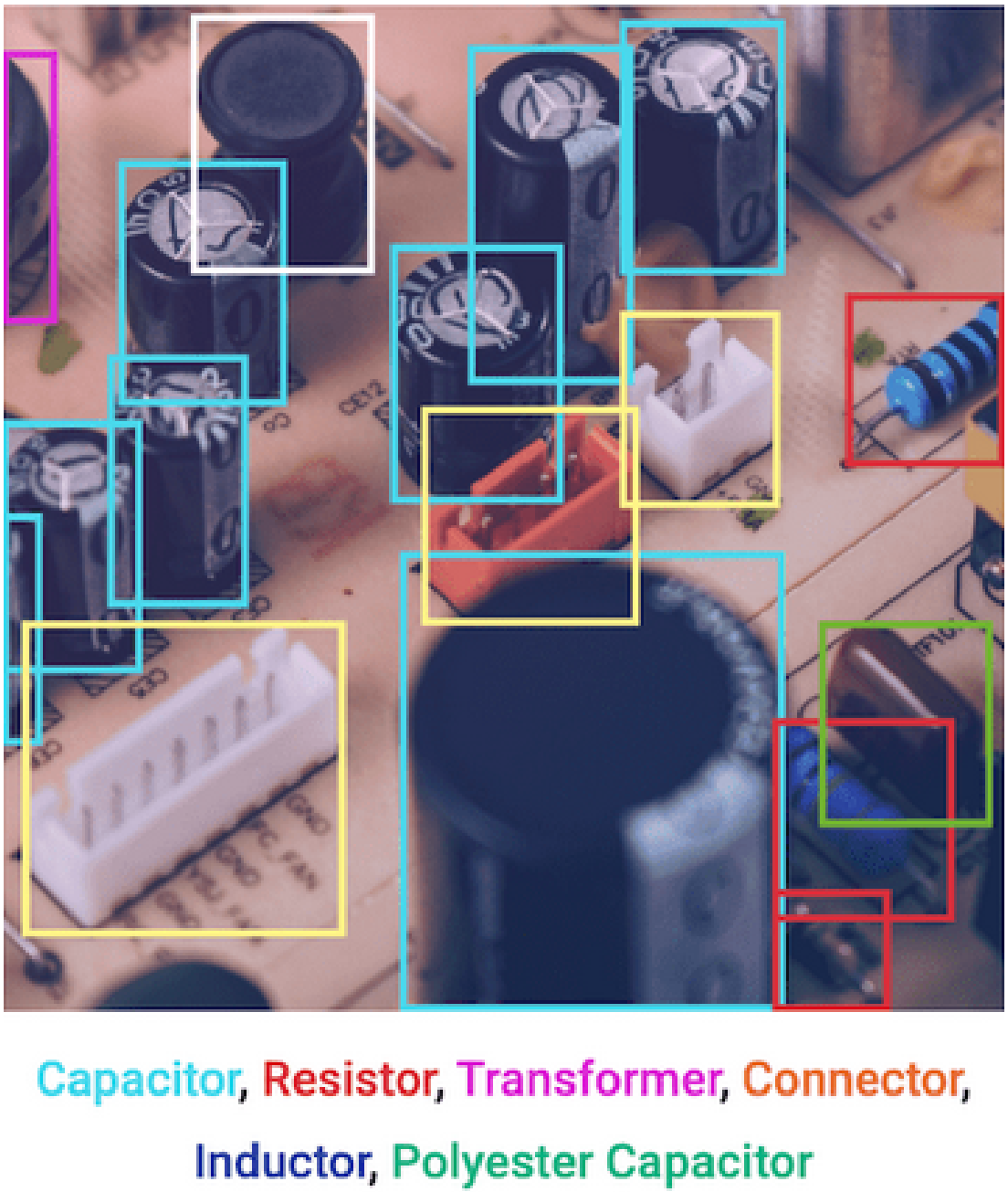
- **Application**
  - An application running inference on a live USB camera feed (optionally imported picture or video)
  - **GUI**
    - The GUI Application is based on Qt Creator, using C++
- **Inference**
  - Running on C++
  - Utilising
  - CPU
  - CUDA cores provided by the GPU
- **Architecture**
  - YOLOv5

Internal AI Object Detection steps

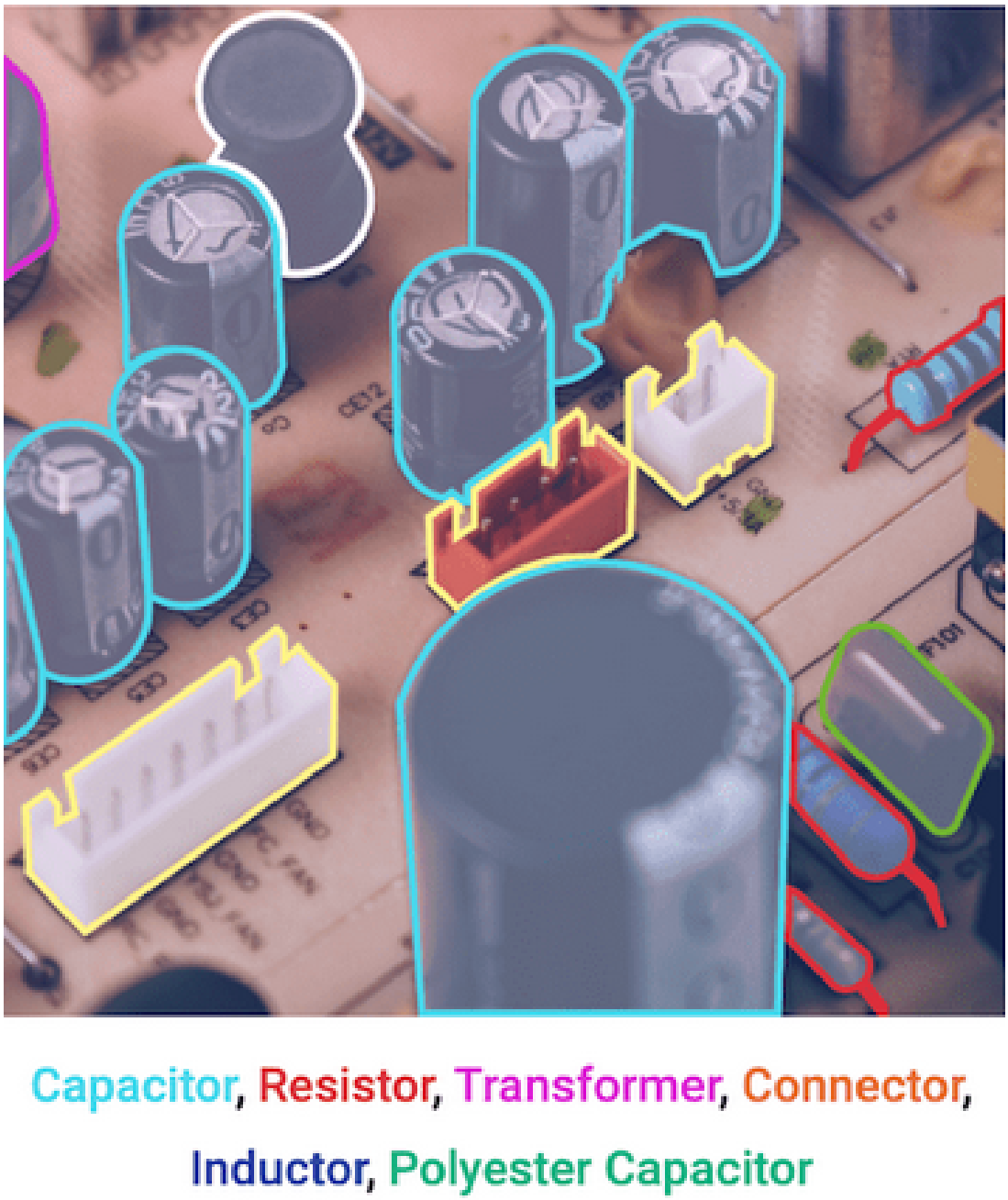
- Classification
-

Figure 34: Object Detection: Classification

- The identification of a part of an image believe to contain an item of a class the model was trained to detect.



- *Figure 35: Object Detection: Object Detection*
- The Bounding by a box of the classified segments of the image.



- *Figure 36: Object Detection: Segmentation*
- The process of identifying the exact bounding box of the item detected.

9. Timeline

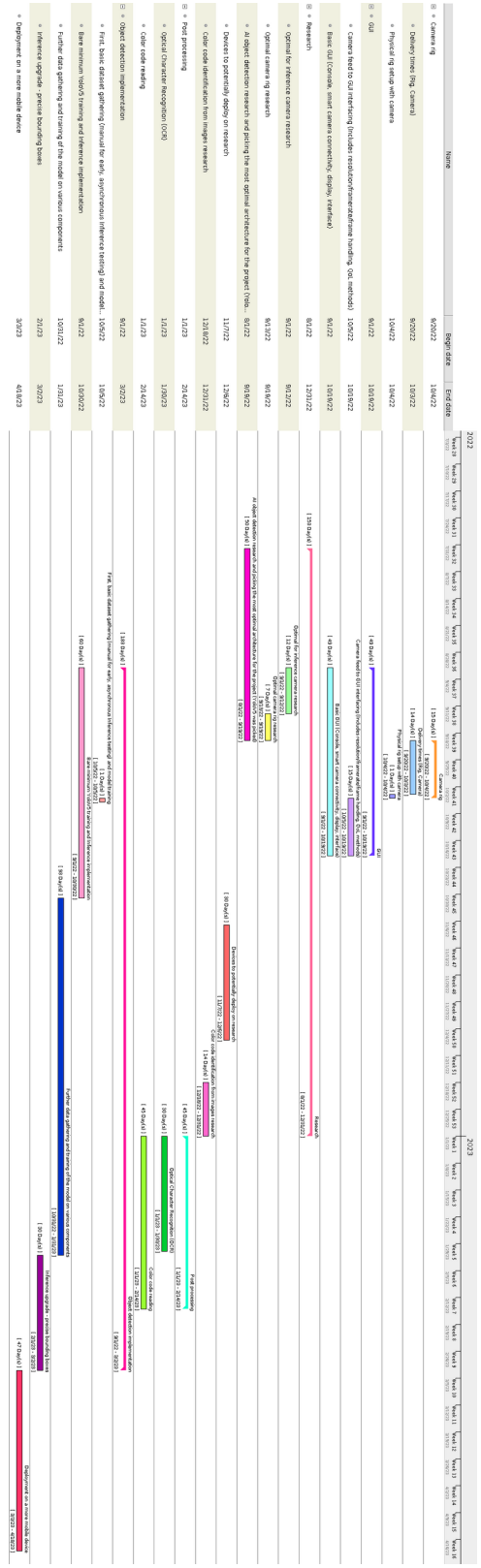


Figure 37: Project Gantt Chart

Name	Begin date	End date	Duration
Camera rig	9/20/22	10/4/22	15
Delivery times (Rig, Camera)	9/20/22	10/3/22	14
Physical rig setup with camera	10/4/22	10/4/22	1
GUI	9/1/22	10/19/22	49
Basic GUI (Console, smart camera connectivity, display, interface)	9/1/22	10/19/22	49
Camera feed to GUI interfacing (Includes resolution/framerate/frame handling, QoL methods)	10/5/22	10/19/22	15
Research	8/1/22	12/31/22	153
Optimal for inference camera research	9/1/22	9/12/22	12
AI object detection research and picking the most optimal architecture for the project (YoloV5 was picked)	8/1/22	9/19/22	50
Optimal camera rig research	9/13/22	9/19/22	7
Devices to potentially deploy on research	11/7/22	12/6/22	30
Color code identification from images research	12/18/22	12/31/22	14
Post processing	1/1/23	2/14/23	45
Optical Character Recognition (OCR)	1/1/23	1/30/23	30
Color code reading	1/1/23	2/14/23	45
Object detection implementation	9/1/22	3/2/23	183
First, basic dataset gathering (manual for early, asynchronous Inference testing) and model training	10/5/22	10/5/22	1
Bare minimum YoloV5 training and Inference implementation	9/1/22	10/30/22	60
Further data gathering and training of the model on various components	10/31/22	1/31/23	93
Inference upgrade - precise bounding boxes	2/1/23	3/2/23	30
Deployment on a more mobile device	3/3/23	4/18/23	47

Figure 38: Project Gantt Chart as a Table

9.1. *Progress*

9.1.1. Issues encountered

Augmentation rotation issue

- **Description**
- A glitch in augmentation provided by YOLOv5 [20], where rotation during augmentation has shifted the bounding boxes of the components, causing inaccurate feedback to the model, preventing it from training appropriately.
- **Example**

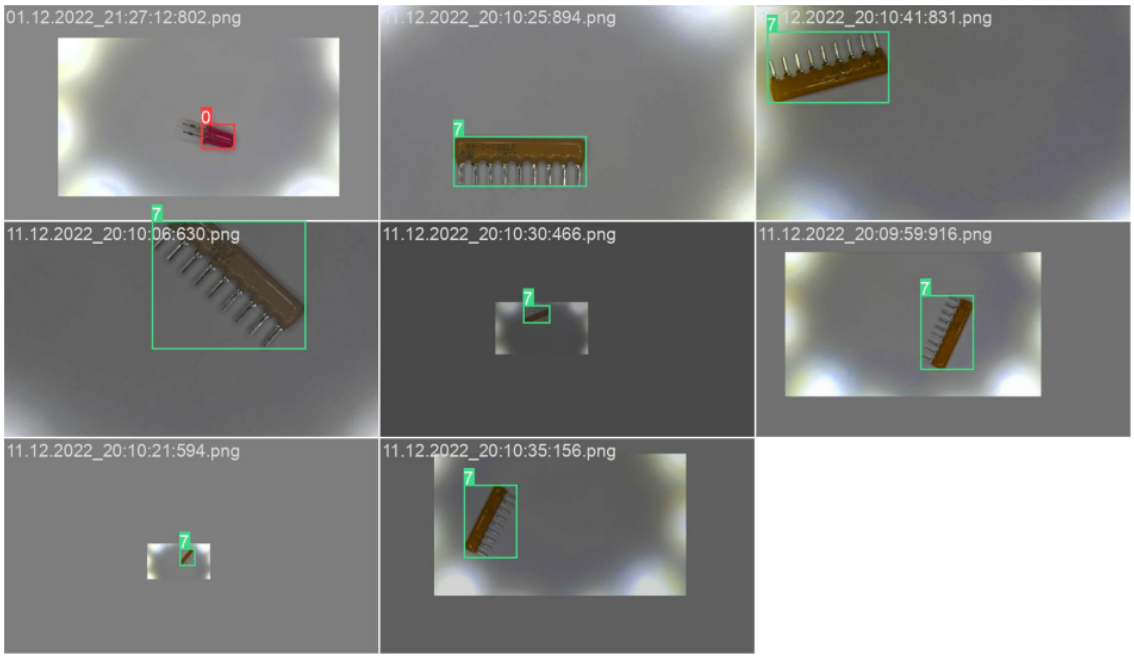


Figure 39: Rotation augmentation issue: Expected bounding boxes after rotation augmentation

- Note the snug fit of the bounding box around the edges of the component.
- That is desirable, as it provides accurate information on what the model should be looking for.

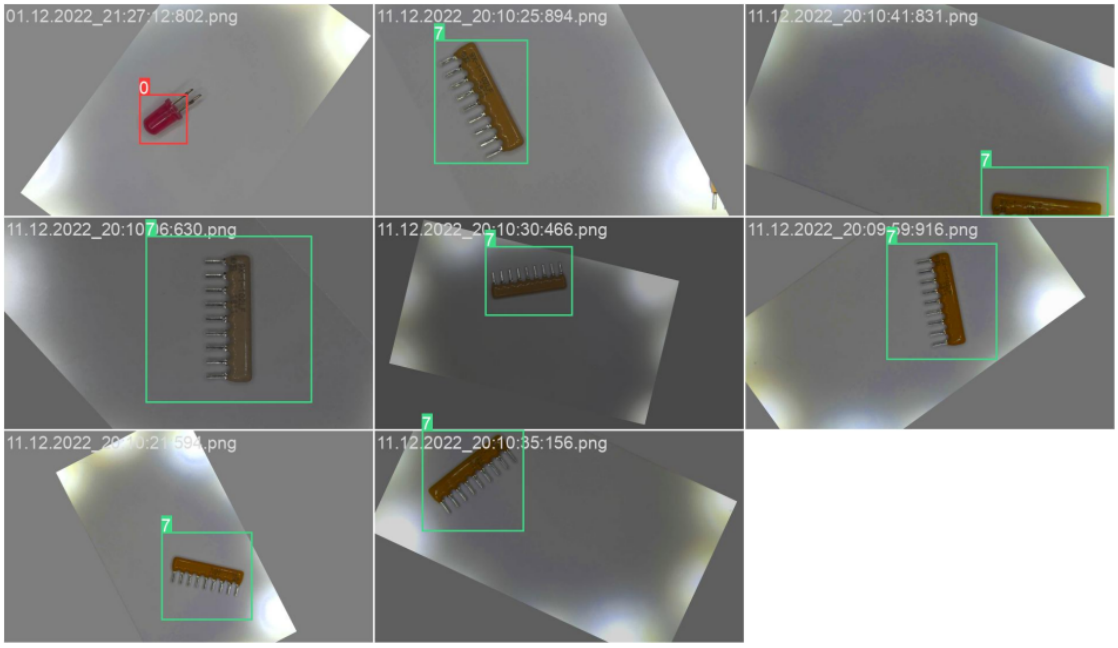


Figure 40: Rotation augmentation issue: Actual bounding boxes after augmentation

- Note the unnecessarily expanded bounding boxes.
- This will train the model in undesirable ways, detecting parts it should not.

Submitted GitHub [21] issue

- **Link [22]**
  - https://github.com/ultralytics/yolov5/issues/10639
- **Information gathered from replies as of todays date**
  - This issue has been reported to be part of YOLOv7 [23] augmentation also.

9.1.2. Training

1st batch, test run

- **Architecture**
  - YOLOv5
- **Pre-trained model used**
  - yolov5s
- **Image Count**
  - **Training**
    - 100
  - **Evaluation**
    - 20
- **Classes (1 Total)**
  - Resistor
- **Augmentation**
  - Default
- **Epoch count**
  - 400
- **Average time per epoch**
  - 34 seconds

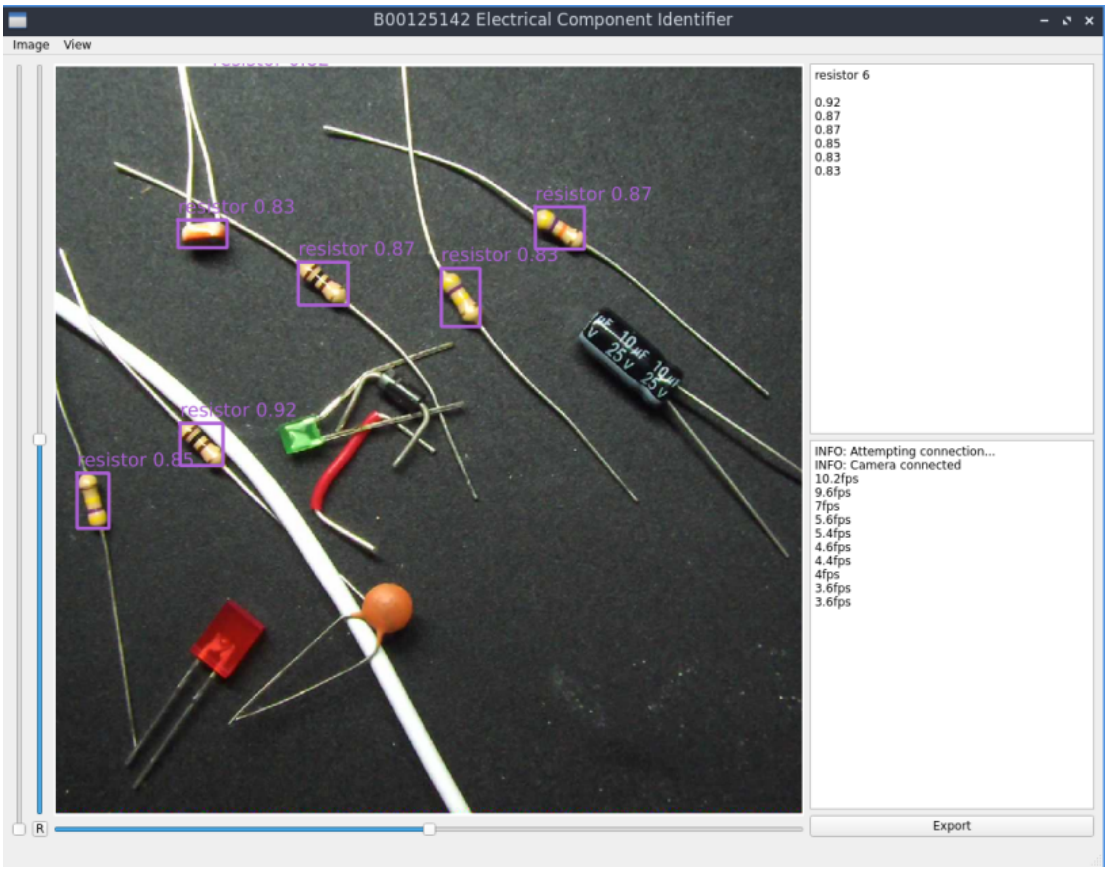


Figure 41: 1st batch model inference example

- **Observations**
  - Surprisingly good results for a model trained from 120 images, with confidence values above 0.8 and sometimes over 0.9!

2nd batch

- **Architecture**
  - YOLOv5
- **Pre-trained model used**
  - yolov5m
- **Image Count**
  - **Training**
    - 1800
  - **Evaluation**
    - 540
- **Classes (9 Total)**
  - red_led
  - green_led
  - blue_led
  - yellow_led
  - ac_capacitor
  - dc_capacitor
  - resistor
  - sip_resistor
  - pcb_terminal
- **Augmentation**
  - Default
- **Epoch count**
  - 250
- **Average time per epoch**
  - 2 minutes
- **Observations**
  - Rather poor results. Confidence values usually below 0.7, struggled to classify accurately.

3rd batch

- **Architecture**
  - YOLOv5
- **Pre-trained model used**
  - yolov5m
- **Image Count**
  - **Training**
    - 2393
  - **Evaluation**
    - 724
- **Classes (10 Total)**
  - red_led
  - green_led
  - blue_led
  - yellow_led
  - ac_capacitor
  - dc_capacitor
  - resistor
  - sip_resistor
  - pcb_terminal
  - metal_nut
- **Augmentation**
  - Default
- **Epoch count**
  - 300
- **Average time per epoch**
  - 2 minutes and 20 seconds

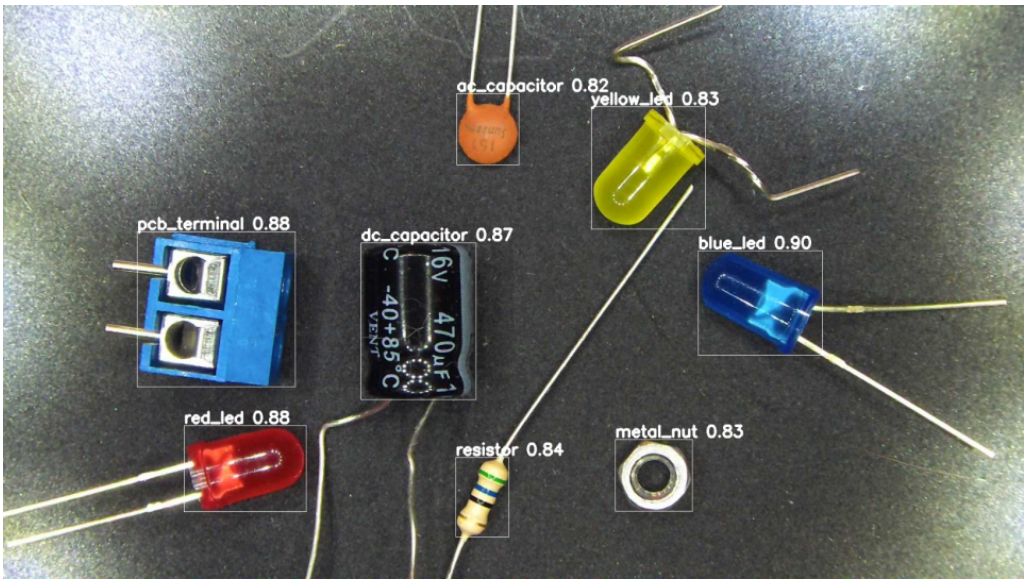


Figure 42: 3rd batch model inference example

- **Observations**
  - A ring light has been introduced for both training and inference running.
  - Great results with confidence values consistently above 0.8, classifying all classes accurately!
  - The boost in results is limited to the set rig.

10. Discussion

- The original project concept at the time of project proposition had a slight shift in focus.
    - The project was intended to focus on a field that is more familiar, which was more on the theme of a high precision tool, which requires a set rig to push beyond the limitations to produce high quality results.
    - This has eversince shifted more towards a more flexible, but less precise concept with the consideration of more generic use through mobile deployment.
        - As mobile deployment of object detection was not explored, this has put additional strain on the timeline.
- Great results in object detection have now been achieved through the set rig, however, that is not quite the case outside of a set rig.
    - The model has been train on on over 3000 images now that were taken and labeled over the course of several months.
    - At this rate, a significant increase in the dataset would be required for adequate results outside of a set rig, which would require a significant amount of additional time investment.
    - This is of course is not a very viable option without neglecting other parts of the project.
- Transition from YOLOv5 to YOLOv8 is currently being considered.
    - YOLOv8 as of right now only has experimental versions of some features that are essential to the successs of this project, which are currently being tested to see if they will be adequate enough to upgrade the current architecture.
    - Overall, YOLOv8 has shown a considerable increase in confidence values.

11. Conclusion

- Despite the additional strains, the project is currently on-track and is following the planned milestones in sequence.
- After the achieval of great confidence value results from the latest model, the next milestone is the image post-processing.

[1] R. Girshick, “rbgirshick/rcnn.” Jan. 29, 2023. Accessed: Jan. 29, 2023. [Online]. Available: https://github.com/rbgirshick/rcnn

- [2] M. deGroot, “SSD: Single Shot MultiBox Object Detector, in PyTorch.” Jan. 24, 2023. Accessed: Jan. 24, 2023. [Online]. Available: https://github.com/amdegroot/ssd.pytorch
- [3] G. Jocher, A. Chaurasia, and J. Qiu, “YOLO by Ultralytics.” Jan. 2023. Accessed: Jan. 29, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
- [4] “YOLO detection documentation.” https://web.cs.ucdavis.edu/~yjlee/teaching/ecs289g-winter2018/YOLO.pdf (accessed Jan. 29, 2023).
- [5] “Buy a Raspberry Pi 4 Model B – Raspberry Pi.” https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (accessed Jan. 24, 2023).
- [6] “BeagleBoard.org - bone.” https://beagleboard.org/bone (accessed Jan. 24, 2023).
- [7] “Jetson Nano Developer Kit,” *NVIDIA Developer*, Mar. 06, 2019. https://developer.nvidia.com/embedded/jetson-nano-developer-kit (accessed Jan. 24, 2023).
- [8] “Intel® Neural Compute Stick 2,” *Intel*. https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html (accessed Jan. 29, 2023).
- [9] “Android,” *Android*. https://www.android.com/intl/en_ie/ (accessed Jan. 29, 2023).
- [10] “Experience the Power of Windows 11 OS, Computers & Apps | Microsoft.” https://www.microsoft.com/en-ie/windows (accessed Jan. 29, 2023).
- [11] “Linux.org,” *Linux.org*, Jan. 26, 2023. https://www.linux.org/ (accessed Jan. 29, 2023).
- [12] “Mac,” *Apple (Ireland)*. https://www.apple.com/ie/mac/ (accessed Jan. 29, 2023).
- [13] “Embedded Software Development Tools | Cross Platform IDE | Qt Creator.” https://www.qt.io/product/development-tools (accessed May 03, 2022).
- [14] “AMD Ryzen™ 7 5800X3D Gaming Processor.” https://www.amd.com/en/products/cpu/amd-ryzen-7-5800x3d (accessed Jan. 29, 2023).
- [15] “Welcome to AMD.” https://www.amd.com/en/home (accessed Jan. 29, 2023).
- [16] “PC Components | Gaming Gear | CORSAIR.” https://www.corsair.com/ww/en/ (accessed Jan. 29, 2023).
- [17] “VENGEANCE RGB PRO SL 16GB (2x8GB) DDR4 DRAM 3200MHz C16 Memory Kit – White.” https://www.corsair.com/ww/en/Categories/Products/Memory/Vengeance-RGB-PRO-SL-White/p/CMH16GX4M2E3200C16W (accessed Jan. 29, 2023).
- [18] “NVIDIA GeForce RTX 3060 Family,” *NVIDIA*. https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/rtx-3060-3060ti/ (accessed Jan. 29, 2023).
- [19] “World Leader in AI Computing,” *NVIDIA*. https://www.nvidia.com/en-gb/ (accessed Jan. 29, 2023).
- [20] “ultralytics/yolov5.” Ultralytics, Nov. 06, 2022. Accessed: Nov. 06, 2022. [Online]. Available: https://github.com/ultralytics/yolov5
- [21] “Build software better, together,” *GitHub*. https://github.com (accessed May 03, 2022).

- [22] “Degrees rotation augmentation messes up the size of the labels · Issue #10639 · ultralytics/yolov5,” *GitHub*. https://github.com/ultralytics/yolov5/issues/10639 (accessed Jan. 29, 2023).

- [23] K.-Y. Wong, “Official YOLOv7.” Jan. 29, 2023. Accessed: Jan. 29, 2023. [Online]. Available: https://github.com/WongKinYiu/yolov7
- [24] “Microorganism recognition and counting on Petri plates,” *NeuroSYS: AI & Custom Software Development*, Oct. 13, 2021. https://neurosys.com/blog/recognition-and-counting-of-microorganisms (accessed Jan. 27, 2023).
- [25] Q. Jiang, D. Tan, Y. Li, S. Ji, C. Cai, and Q. Zheng, “Object Detection and Classification of Metal Polishing Shaft Surface Defects Based on Convolutional Neural Network Deep Learning,” *Appl. Sci.*, vol. 10, no. 1, Art. no. 1, Jan. 2020, doi: 10.3390/app10010087.
- [26] B. T. Marvel, “CROP DISEASE DETECTION USING IMAGE PROCESSING TECHNIQUE AND CNN NETWORKS : Part2,” *Medium*, May 01, 2022. https://medium.com/@b.thusharmarvel97/crop-disease-detection-using-image-processing-technique-and-cnn-networks-part2-b3f8588f77e5 (accessed Jan. 27, 2023).
- [27] X. Liao, S. Lv, D. Li, Y. Luo, Z. Zhu, and C. Jiang, “YOLOv4-MN3 for PCB Surface Defect Detection,” *Appl. Sci.*, vol. 11, no. 24, Art. no. 24, Jan. 2021, doi: 10.3390/app112411701.