
Development and implementation of a waste recognition system for segregation and identification of various kinds of recyclable waste

Guided by :

Dr-Ing. Prof. Jürgen Wittmann
(Technische Hochschule Deggendorf (TC CHAM))

Written By :

Akshay Kukadiya (12102274)
Arun Thavaramkunnath Ashokan (12203424)
Gaurav Sharma (12200584)
Kirtan Soni (12201189)

Date of submission: 08.07.2022



Table of Contents

Table of Contents.....	2
Abstract.....	3
Acknowledgement.....	5
1.Introduction	6
2.Literature review	7
2.1 Transfer Learning:	7
3.Model Building	9
3.1Datasets Preparation:.....	9
3.1.1 Data Set:.....	9
3.1.2Data pre-processing:.....	9
3.1.3 Steps in Data Pre-processing:.....	10
3.2 Training YOLOv5 Model:	12
3.2.1 Configuring the Dataset into the model	12
3.3 Building the Model.....	15
3.4Setting Parameters:	18
3.5Model Hyperparameters:.....	18
3.6Model parameters:	19
3.7Layer summary:	24
4Setup & Evaluation	27
4.1Dependencies And Requirements	27
4.2GPU accessing using CUDA and cuDNN :	28
4.3 Live_video.py	30
4.4 Model Architecture.....	31
4.5 Bill of Material:.....	32
4.6Training Results.....	32
5Result and Evaluation	34
References	36
Appendix.....	37

Abstract

Waste management could be a pervasive problem in today's world and is rising continuously with an increase in urbanization. Waste management includes a vibrant part to own an ecological environment. Proper waste disposal at the dumping sites has a necessary part of sorting at the bottom level. Increases in time and more manpower are required to sort waste using the normal process. Sorting waste may be wiped out in various methods and forms. Analysing and classifying the rubbish using image processing may be productive thanks to processing waste materials.

This case study aims to investigate existing research presented in studies around the globe. This can enable to work out the issues, an algorithm used and the method of these cited studies. It can even assess the proper algorithm to be employed in a future study. These case studies speak about the proposed systems during which waste segregation befell. These also discuss the drawbacks faced by the already existing systems and algorithms they used. With it, this paper gives plenty of opportunities to supply new knowledge in creating a replacement system. The following report has brief description of the model trained for segregating different type of household waste. The general collection of dataset and preparing the data for training purposes. The model is trained with transfer learning approach using the YOLO algorithm. The model can identify 8 different types of waste. This document also contains all the detailed information about the Model architectures, performance, and results that have been used for the project.

Acknowledgement

We are grateful because we managed to complete the case study of Intelligence systems in Waste recognition with AI within the given time and with guidance from Dr-Ing. Prof. Jürgen Wittmann at Technische Hochschule Deggendorf (TC CHAM). This case study could not be completed without the effort and cooperation of Prof. Jürgen Wittmann. We thank Prof. Jürgen Wittmann for the guidance and encouragement in finishing the Case study in the Sensors and Actuators project.

1. Introduction

2. Literature review

To better understand the various studies presented and to be ready to determine what reasonably algorithm is to be used best the subsequent are the various studies that used the various algorithm and studies where it applies. It also has the strength and weaknesses during which it may be utilized in deciding on what algorithm is the right one. There are many systems which will separate waste into different categories. they're the subsequent - Intelligent Waste Separator (IWS) that can replace the standard way of coping with waste; The prototype automatically places garbage in altered basins and accepts inbound wastes by employing a multimedia embedded processor, image processing specifically using the image recognition algorithm, and machine learning to pick out and separate waste (Myra G. Flores, 2015). It developed prototype consists of a shared trash bin, with supplementary basins in it, using multimedia technology Spot Garbage could be a smartphone-based application. It detects a pile of garbage and identifies the situation where the rubbish is present by using the placement access of smartphones. The app uses the convolutional neural networks architecture for identifying wastes in images - IoT-based Waste Collection System using Infrared Sensors This automatic waste segregator uses a contemporary [2].

The classification method is referred to as Convolutional Neural Networks to classify the waste into various categories. this technique paves the thanks for better recycling and reuse processes that help in efficient waste management. By using the concepts of Artificial Neural Networks and Image analysing specifically the image recognition algorithm, the project is geared toward designing and developing a system that may be effectively utilized to segregate waste. - Adaptive and Interactive Modelling System (AIMS) that uses an induction algorithm to interpret sensor data streams and produce an efficient description of object characteristics which can define material separation strategies. [2] - Waste Segregation System Using Artificial Neural Networks the project is aimed at designing and developing a system that may be effectively utilized to segregate waste. By applying the concepts of recognition and classification in Artificial Neural Networks, the proposed system is designed to rightly categorize the various forms of waste. [1]. - Automatic Waste Segregator and Monitoring System may be a system during which it sorts wastes into three different categories, namely metal, plastic, and therefore the wet (organic) waste. Other wastes are categorized as wet waste, which signifies organic waste which classifies as left-over and vegetable peels. Most of the studies presented used microcontrollers and the majority of them are just prototype systems. Variations of waste cannot be determined. a number of the studies are for special purposes only. Other types and new sorts of wastes weren't ready to recognize and determine. additionally, buying and maintaining this sort of study within the microcontroller are proven to

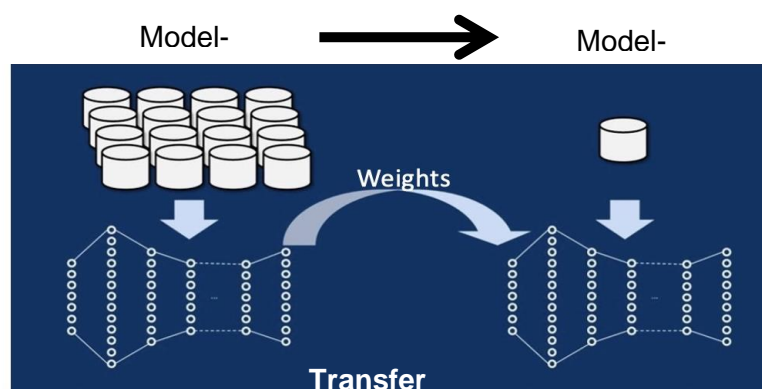
be excessively costly. Classifying waste and segregating may be a lot helpful in many areas like within the industry, household, company and fact school. No current study was being conducted on automating and segregating waste at school. Using technologies available today in sorting waste and determining which is to be recycled may be very effective techniques to handle garbage. [1].

The traditional method involves collecting waste from households and surroundings and so manual segregation of waste at a station or dumping areas. This method is time-consuming, less efficient and increases health issues. YOLO (you look only once) deep learning algorithm is employed to segregate waste as wet, dry, metallic, non-metallic, sanitary, and hazardous waste respectively. This method is Eco-friendly, less time-consuming and efficient. Once the waste is segregated properly then the decomposition or recycling and reuse of waste will be easily done. [3] Various kinds of garbage are the objects which need to detect, and frequently have to train our model before making predictions. The deep learning model is composed of 4 parts: Shallow network, Backbone network, Neck and Head. Among these, the Shallow network and also the Backbone network are mainly chargeable for extracting semantic information like the shape, colour and size from the input feature and converging them into high-dimensional feature mappings. The Neck can optimize the extracted high-dimensional feature mappings, which helps the top to decode higher-quality features.

2.1 Transfer Learning:

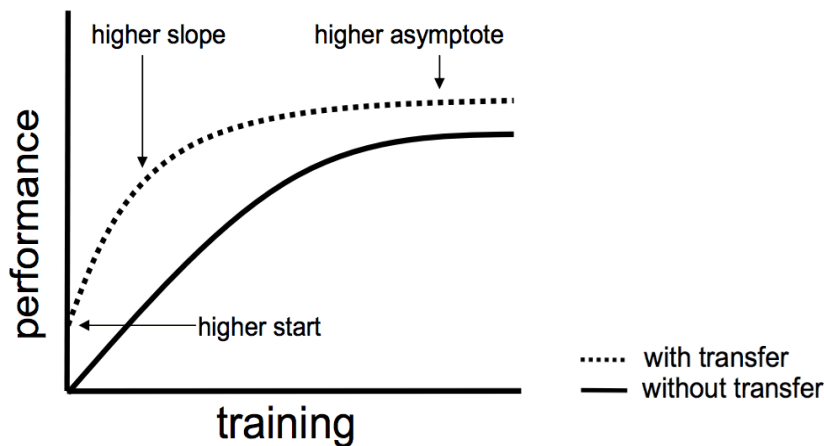
Transfer learning is a technique where a model trained on one task and re-purposed on another similar task. A training machine learning model utilizes the learning of another model. The biggest benefit of transfer learning is that one can use the weights of the State of art ML model (Best in class).

In the below example model 2 is using the learning of Model-1.



Transfer learning (Source: Research paper)

The machine learning model will have better performance when trained with transfer learning, refer to the below graph.



Transfer learning model performance (Source: Research paper)

Transfer learning has various advantages like:

1. Less training data is required.

Since the model is using weights and parameters of the already trained model. So model need not train again with a big amount of dataset.

2. Learning from State of art models (best in class).

There are already many best-in-class pre-trained models available trained with a huge dataset. Such models have many hidden layers and millions of trainable parameters. It required lots of computational power and training can take months even with the best available GPUs. So it is always a good idea to use the power of such a model.

3. Efficiently train multiple models.

Machine learning models can take a long time to properly train. We need to start from the beginning each time with a similar model. Resources and time spent training the machine learning algorithm can be distributed to all different models. The whole training process is made more efficient by re-using algorithm features and transferring information already held by the model.

4. Leverage knowledge to solve new challenges.

Supervised machine learning is one of the most popular today. However, performance may suffer if the data or environment is different from the training data. Transfer learning helps a blended approach from different models to fine-tune a solution to a specific problem with a more accurate and powerful model.

5. Simulated training to prepare for real-world tasks.

Models trained in real-world environments and scenarios, digital simulations are a less expensive or time-consuming option. Models can be trained to interact with objects in a simulated environment. Simulated environments are used to perform tasks in different scenarios, interacting with objects and environments.

- Few examples where a transfer can be used:

Natural language processing

Computer vision

Neural networks

- We have tested below transfer learning model for this project:

1. VGG16
2. ResNet50
3. MobileNet V2
4. YOLO V5

3. Model Building

After Checking performance and evaluating other models and their relative aspects, we go for the YOLOV5 model for final model training.

3.1 Datasets Preparation:

3.1.1 Data Set:

We have prepared the dataset of recyclable household waste. Dataset is prepared from images from multiple sources like Imagenet, TrashNet and google images. This is a balanced database that contains pictures from 8 different classes Electronics, Batteries, Clothes, Glass, Organic, Metal, Paper and Plastic. There are around a total of 7500+ images in our dataset.



Electronics



Batteries



Clothes



Glass



Organic



Metal



Paper

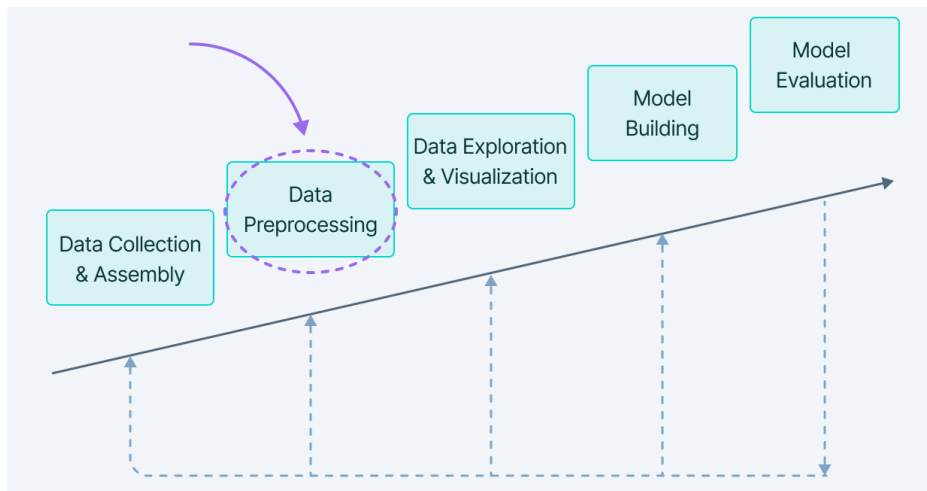


Plastic

3.1.2 Data pre-processing:

Data is pre-processed is the method of cleaning the data and making it suitable for the machine learning model. Data may contain various abnormalities like missing, inconsistent and noise due to their heterogeneous origin which may prevent the model to run properly due to the following reasons:-

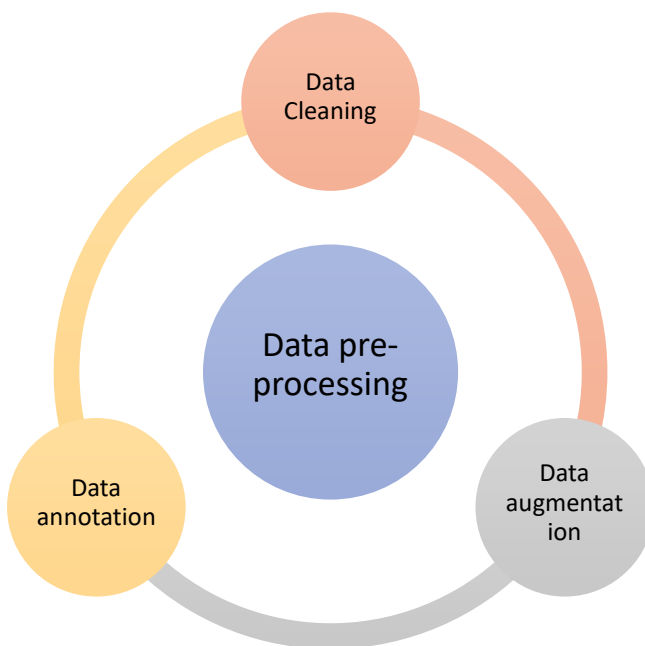
- Duplicate or missing values may give an incorrect view of the overall statistics of data.
- Outliers and inconsistent data points often tend to disturb the model's overall learning and may lead to incorrect predictions



Model training steps (Source: www.v7labs.com/blog/data-preprocessing-guide)

3.1.3 Steps in Data Pre-processing:

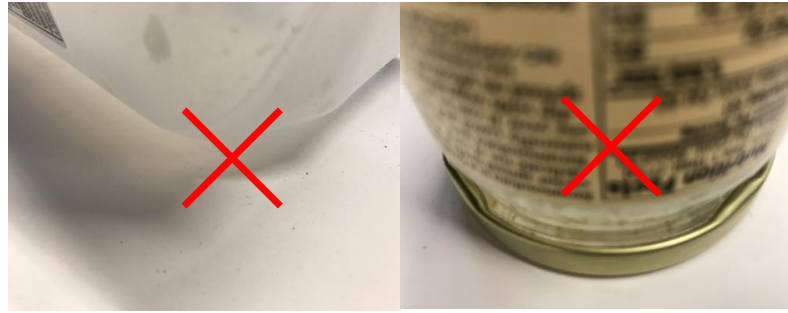
Data pre-processing consists of the following steps:



Data preprocessing

1. Data cleaning:

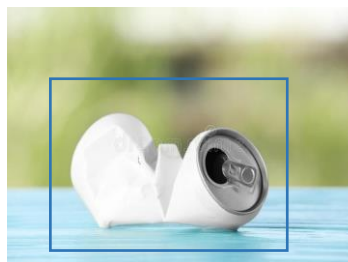
Data is clean by filling missing values, smoothing the noisy data, resolving the inconsistency and removing outliers. We have to remove many images because they were small, blurry or the image is not clear. A few examples are given below.



Picture Filtering on Dataset

2. Data annotation:

We annotated the data by creating a bounding box on the target image in a photo. Data annotation is required so that the machine learning model can understand the class of images.



Annotation in pictures

3. Data augmentation:

Image augmentation is a technique of creating a dataset from existing images by randomly transforming existing images, thereby increasing the size of the training data. We have used Roboflow software for the augmentation of our dataset. During this step, we have done the following steps.

1. Resize to 640x640 (Stretch)
2. Random shear of between -15° to $+15^{\circ}$ horizontally and -15° to $+15^{\circ}$ vertically
3. Random Gaussian blur of between 0 and 2.5 pixels
4. Salt and pepper noise was applied to 10 % of the pixels

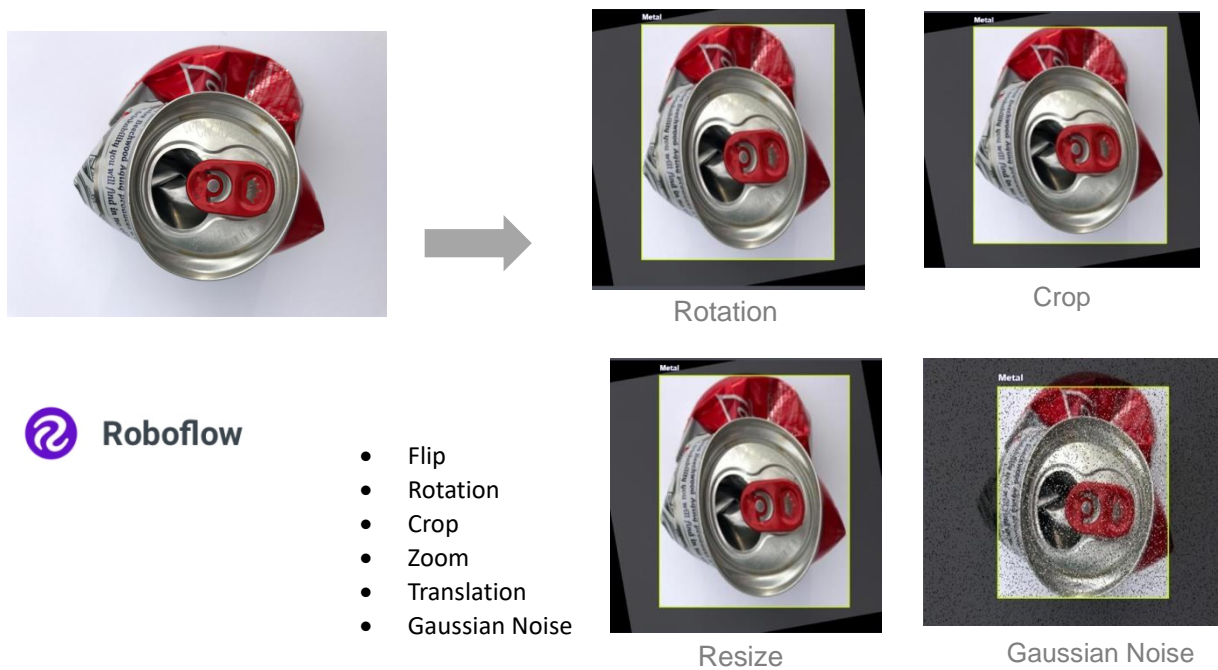


Image annotation with roboflow

3.2 Training YOLOv5 Model:

1. Configuring Dataset
2. Building the Model and setting Parameters
3. Inferences
4. Results

3.2.1 Configuring the Dataset into the model

Dataset is the most necessary part when it comes to building a good AI model, the better the dataset the better the model will train and give accurate predictions. We have finalised using the YOLOv5 model with transfer learning for the recycle waste recognition project. Because it is the most widely used deep learning-based real-time object detection globally. The collected dataset needs to be converted into a specific format so that the model can accept it. There are many types of data formats available which can be prepared with the pre-defined bounding boxes for each image for the corresponding class.

Types of popular data formats:

- a. YOLO
- b. pascal_voc
- c. coco

Each format uses a unique and specific presentation of bounding boxes. We have used the YOLO format for the dataset because the YOLOv5 and different YOLO models use a specific format of input data which consists of several files.

- a. Image folder
- b. Label folder
- c. .yaml file (data configuration file)

Image folder: This folder consists of all the images that we have collected of different classes. All the images have a unique name and they can be in any image format – “.png, .jpg, .jpeg”

Label folder: This folder consists of .txt files. All the text files are saved with the same name as the respective image file. Each text file consists of one bounding box (BBox) annotation for each of the objects in the image. This annotation lies within the range of 0-1 and they are represented in the format:

<object-class-ID> <X centre> <Y centre> <Box width> <Box height>

Example: 0 0.5173535884734922 0.4574881026602574 0.7085552834483893
0.6150162991378904

Here object-class-ID says that it is of class no. 0 which can be associated with a particular class name.

Data configuration file: This is a YAML file written in. YAML (YAML Ain't A Markup Language) format. Since we are training our model with a custom dataset, this file has to be created with those above folders. This YAML file consists of paths to the Train, Validation and Test datasets; the number of classes (NC); and the names of the classes in the same order as their index. At last, save the file. YAML extension.

```

1 path: /content/data/train_data # dataset root dir
2 train: /content/data/train_data/images/train # train images (relative to 'path')
3 val: /content/data/train_data/images/test # val images (relative to 'path')
4
5
6 # Classes
7 nc: 8 # number of classes
8 names: ["electronics",
9         "batteries",
10        "clothes",
11        "glass",
12        "organic",
13        "metal",
14        "paper",
15        "plastic"]

```

Path and Classes

There are a lot of open-source online labelling tools available, some of the popular tools are Label Img, Labelme, DarkLabel, and RoboFlow.

For our dataset, we used the online tool RoboFlow because:

- It is a good tool which can automatically convert the data into YOLOv5 Pytorch format after processing.
- We can directly import the converted dataset via an online API.
- We can upload up to 10000 images for free.

To import the dataset from Roboflow we first imported the function Roboflow from the python library roboflow.

```

from roboflow import Roboflow
rf = Roboflow(model_format="yolov5", notebook="ultralytics")

```

🔗 upload and label your dataset, and get an API KEY here: <https://app.roboflow.com/?model=yolov5&ref=ultralytics>

Importing roboflow for yolo

After executing the code we got a link to our saved project in the roboflow server, which ultimately took us to the API KEY.

(api_key="EaZoy3NiYWrnALVG8cMr")

The code we used to get the data from roboflow:

▼ Get dataset from Roboflow

```
[ ] rf = Roboflow(api_key="EaZoy3NiYWrnALVG8cMr")
    project = rf.workspace("waste-recognition").project("waste-recognition")
    dataset = project.version(2).download("yolov5")

loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in /content/datasets/Waste-Recognition-2 to yolov5pytorch: 100% [78080674 / 78080674] bytes
Extracting Dataset Version Zip to /content/datasets/Waste-Recognition-2 in yolov5pytorch:: 100%|██████████| 2620/2620 [00:02<00:00, 996.28it/s]
```

Importing annotated data

```
Waste Recognition - v3 plastic2
=====

This dataset was exported via roboflow.ai on June 15, 2022 at 10:09 AM GMT

It includes 7500 images.
electronics-batteries-clothes-glass-organic-metal-paper-plastic are annotated in YOLO v5 PyTorch format.

The following pre-processing was applied to each image:
* Auto-orientation of pixel data (with EXIF-orientation stripping)
* Resize to 640x640 (Stretch)

The following augmentation was applied to create 3 versions of each source image:
* Random shear of between -15° to +15° horizontally and -15° to +15° vertically
* Random Gaussian blur of between 0 and 2.5 pixels
* Salt and pepper noise was applied to 10 percent of pixels
```

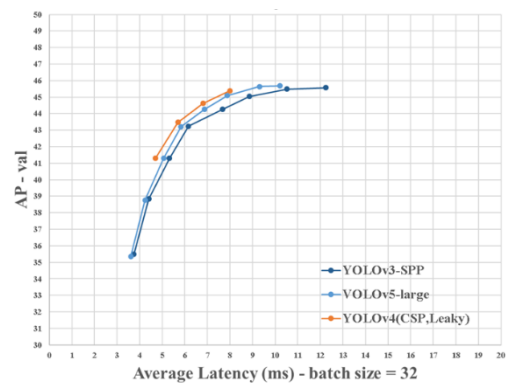
Description of Dataset

3.3 Building the Model

Training a model from scratch for object detection will be very time-consuming and will require a huge dataset and ultimately making it very expensive. The solution, we use the transfer learning method which is a useful way to retrain a model quickly on a custom dataset. We freeze part of the initial weights in place and the rest of the weights of the pre-trained model are used to compute loss and are updated by the optimizer.

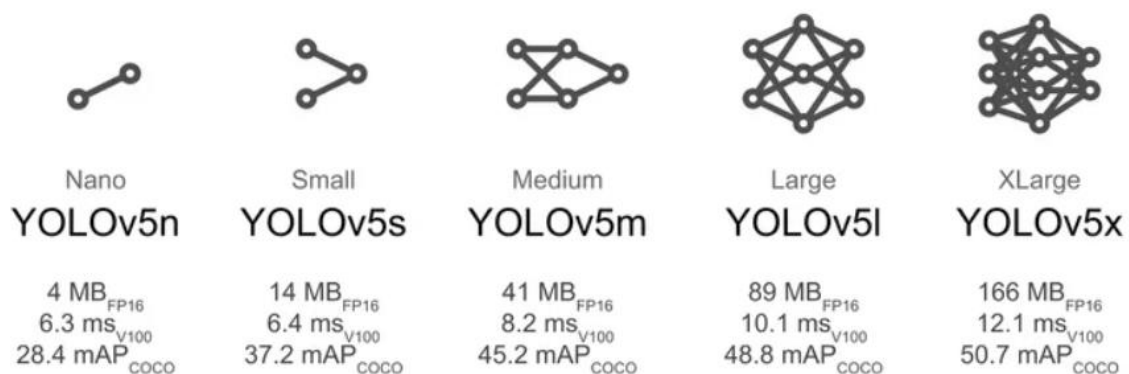
Here in our project, we have used the transfer learning method on the pre-trained YOLOv5 model.

Why yolov5 and not yolov4?

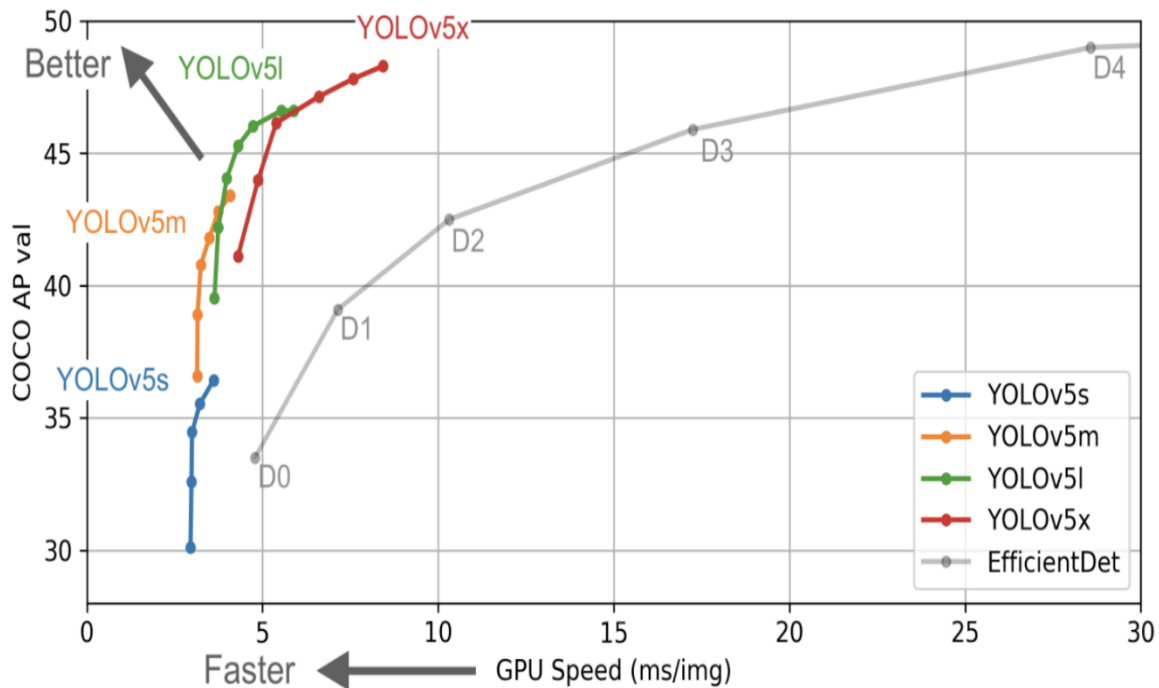


Source: YOLO/Ultralytics

- YOLOv5 is of size 27 megabytes and YOLOv4 is of size 244 megabytes. YOLOv5 is almost 90% smaller than YOLOv4, this means it can be deployed to embedded devices much more easily.
- It is faster, YOLOv4 can give 50 FPS and YOLOv5 can give up to 140 FPS when used over good processing power. So, it is about 180% faster than YOLOv4.
- YOLOv5's accuracy is 0.895mAP which is 0.1% better than YOLOv4.
- In YOLOv5 there are five variants:
- YOLOv5-n which is a nano version
- YOLOv5-s which is a small version
- YOLOv5-m which is a medium version
- YOLOv5-l which is a large version, and
- YOLOv5-x which is an extra-large version.



Source: machinelearningknowledge.ai



Source: blog.roboflow.com

- In our project, we used yolov5-s taking future aspects in mind. A small model can easily fit in any embedded device like Raspberry-pi or Jetson Nano.

Small
YOLOv5s

14 MB_{FP16}
6.4 ms_{V100}
37.2 mAP_{COCO}

We used the PyTorch framework for developing the model. PyTorch is a deep learning tensor library based on Python and Torch.

Why PyTorch?

- It uses dynamic computation graphs which help to understand the results of the model in a better and well-organized way. It has a good visualisation support library called TensorBoard, which helps developers to track the model training process in a better way.
- It is comparatively easy to use and learn due to intuitive syntax (Pythonic).
- It has strong community support which makes debugging easy.
- We started the model building in the Google colab because of the fast-processing power. At first, we cloned the repository - YOLOv5 by Ultralytics from Github and installed the requirements.txt file for YOLOv5.



▼ Setup

```
#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5 # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow
```

```
print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'})")
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 12270, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 12270 (delta 5), reused 9 (delta 1), pack-reused 12250
Receiving objects: 100% (12270/12270), 11.98 MiB | 33.17 MiB/s, done.
Resolving deltas: 100% (8491/8491), done.
/content/yolov5
596 kB 32.3 MB/s
145 kB 41.5 MB/s
178 kB 62.1 MB/s
1.1 MB 60.9 MB/s
67 kB 6.7 MB/s
54 kB 3.2 MB/s
138 kB 72.0 MB/s
62 kB 1.8 MB/s
Building wheel for roboflow (setup.py) ... done
Building wheel for wget (setup.py) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
google-colab 1.0.0 requires requests<2.23.0, but you have requests 2.28.0 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is incompatible.
Setup complete. Using torch 1.11.0+cu113 (Tesla T4)
```

Importing YOLO model

After cloning the repository, we imported all the necessary libraries:

```
import torch
import os
from IPython.display import Image, clear_output # to display images
```

3.4 Setting Parameters:

Each architecture like YOLOv5 has its parameters, this file comes in the repository that we cloned that can be adjusted. These parameters should be set at the beginning of the model training. There are two types of parameters which needs to be configured to train the model in the right direction.

- Model Hyperparameters
- Model parameters

3.5 Model Hyperparameters:

Hyperparameters are parameters whose values control the learning process and determine the learning parameters that the model ends up learning, they are essential for model

optimization. These parameters have to be adjusted by the developer. The hyperparameters are in .yaml format and generally with the name 'hyp.scratch.yaml'. The Hyperparameter file should be included in the model parameters as a path.

Here are examples of some of the parameters that come in the hyp.scratch.yaml file:

```
lr0: 0.01 # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2 # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937 # SGD momentum/Adam beta1
weight_decay: 0.0005 # optimizer weight decay 5e-4
warmup_epochs: 3.0 # warmup epochs (fractions ok)
warmup_momentum: 0.8 # warmup initial momentum
warmup_bias_lr: 0.1 # warmup initial bias lr
box: 0.05 # box loss gain
cls: 0.5 # cls loss gain
cls_pw: 1.0 # cls BCELoss positive_weight
obj: 1.0 # obj loss gain (scale with pixels)
obj_pw: 1.0 # obj BCELoss positive_weight
iou_t: 0.20 # IoU training threshold
anchor_t: 4.0 # anchor-multiple threshold
anchors: 0 # anchors per output grid (0 to ignore)
fl_gamma: 0.0 # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015 # image HSV-Hue augmentation (fraction)
hsv_s: 0.7 # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4 # image HSV-Value augmentation (fraction)
degrees: 0.0 # image rotation (+/- deg)
translate: 0.1 # image translation (+/- fraction)
scale: 0.5 # image scale (+/- gain)
shear: 0.0 # image shear (+/- deg)
perspective: 0.0 # image perspective (+/- fraction), range 0-0.001
flipud: 0.0 # image flip up-down (probability)
fliplr: 0.5 # image flip left-right (probability)
mosaic: 1.0 # image mosaic (probability)
mixup: 0.0 # image mixup (probability)
```

Source: researchgate.net

3.6 Model parameters:

These parameters are the configuration model, which is internal to the model. The model parameters are estimated while training the model. These are essential for making model predictions. These parameters decide the total amount of trainable parameters available, In our case, we have used YOLOv5 which has over 7.3 million trainable parameters.

Where did this number come from? Well, the total number of parameters is the sum of all the weights and biases which are present on the neural network.

Below are the Model parameters:

```
weights=yolov5s.pt,  cfg=/models/yolov5s.yaml ,  data=/content/datasets/Waste-Recognition-2/data.yaml,  hyp=data/hyps/hyp.scratch-low.yaml,  epochs=50,  batch_size=16,  imgsz=640,  rect=False,  resume=False,  nosave=False,  noval=False,  noautoanchor=False,  noplots=False,  evolve=None,  bucket=,  cache=ram,  image_weights=False,  device=,  multi_scale=False,  single_cls=False,  optimizer=SGD,  sync_bn=False,  workers=8,  project=runs/train,  name=exp,  exist_ok=False,  quad=False,  cos_lr=False,  label_smoothing=0.0,  patience=100,  freeze=[0],  save_period=-1,  local_rank=-1,  entity=None,  upload_dataset=False,  bbox_interval=-1,  artifact_alias=latest
```

After configuring everything we start to train the model using the following command:

```
!python train.py      --hyp      /hyp.finetune.yaml      --img 640 --batch 16 --epochs 50 --data /content/datasets/Waste-Recognition/data.yaml --cfg "" --weights yolov5s.pt --cache
```

Here:

Image – input image size

Batch – batch size

Hyp – Hyperparameters

Epochs – number of epochs

Data – .yaml file of the dataset that we created

Cfg – model selection YAML file

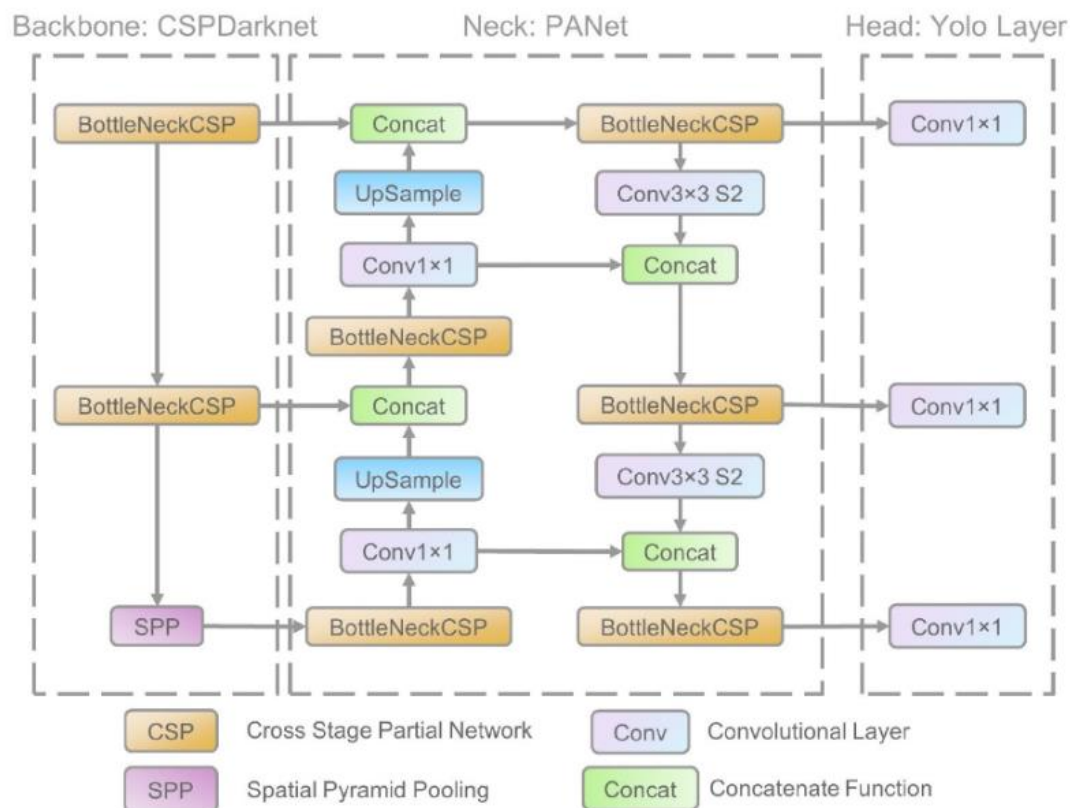
Weights – weights file to apply transfer learning to

Parameter used in our model – Epochs = 50, Input_Image_size = 640 X 640, Input_Batch_size = 16

Running this will give output as a model summary and will start the training process.

But first, let's understand the architecture of YOLOv5 and how we can apply transfer learning. Below is a sample architecture of a typical YOLOv5 model.

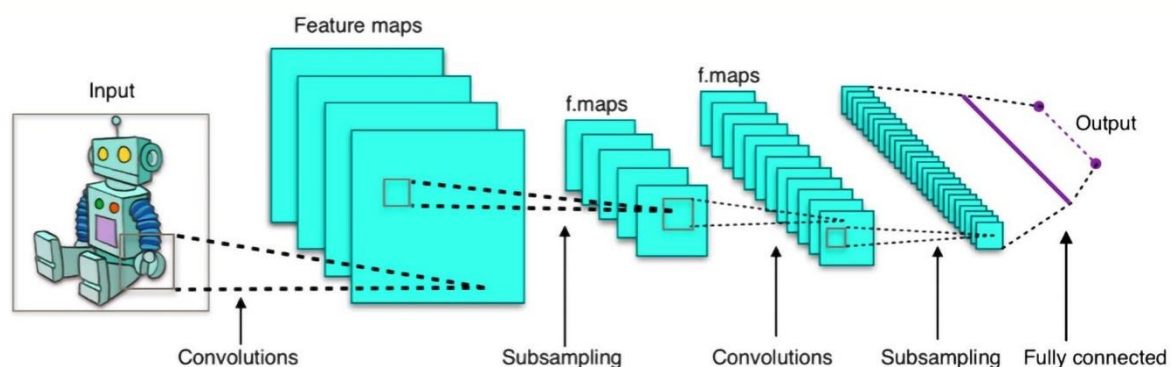
Source: researchgate.net



Source: researchgate.net

The network architecture of YOLOv5 consists of three parts:

- Backbone: CSPDarknet
- Neck: PANet
- Head: Yolo layer



Source: researchgate.net

The data are first input to the CSPDarknet, which is a convolutional neural network and backbone for object detection that uses DarkNet-53 such as YOLOv5. The CSPNet strategy is to partition the feature map of the base layer into two parts and then merge them through a cross-stage hierarchy. Then the layers are passed to the PANet or Path Aggregation Network which aims to boost the information flow in a proposal-based instance segmentation framework. Specifically, the feature hierarchy is enhanced with accurate localization signals in lower layers by bottom-up path augmentation, which shortens the information path between lower layers and the topmost feature. Finally, the Head or Yolo detection layers output detection results in the form of class, score, location, size, etc.

We override the YOLOv5 model with new parameters and weights (TransferLearning), by freezing the top 10 layers or backbone of the architecture and updating the rest of the parameters with our dataset. Default YOLOv5 file has 80 classes, but in our case, we have 8 so we have to change the nc which is the number of classes to 8. This we can do in our data configure file (".yaml").

	from	n	params	module	arguments	
0	-1	1	3520	models.common.Conv	[3, 32, 6, 2, 2]	<div style="border: 1px solid black; padding: 5px; display: inline-block; transform: rotate(-90deg); transform-origin: center;">Freeze</div>
1	-1	1	18560	models.common.Conv	[32, 64, 3, 2]	
2	-1	1	18816	models.common.C3	[64, 64, 1]	
3	-1	1	73984	models.common.Conv	[64, 128, 3, 2]	
4	-1	2	115712	models.common.C3	[128, 128, 2]	
5	-1	1	295424	models.common.Conv	[128, 256, 3, 2]	
6	-1	3	625152	models.common.C3	[256, 256, 3]	
7	-1	1	1180672	models.common.Conv	[256, 512, 3, 2]	
8	-1	1	1182720	models.common.C3	[512, 512, 1]	
9	-1	1	656896	models.common.SPPF	[512, 512, 5]	
10	-1	1	131584	models.common.Conv	[512, 256, 1, 1]	

[80, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]



[8, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]

Output Layer

Overriding model.yaml nc=80 with nc=8

The rest of the parameters are updated by the optimizer, there are 3 popular optimizers:

- Adam
- Momentum
- Stochastic Gradient Descent or SGD (by default)

We have used the optimizer SGD which is a basic algorithm responsible for having neural networks coverage. In SGD we only calculate the cost of one example for each step unlike Batch Gradient Descent (BGD) which speeds up neural networks greatly.

The equation for SGD to update parameters in a neural network:

$$\theta = \theta - \eta \cdot \underbrace{\nabla_{\theta} J(\theta; x, y)}_{\text{Backpropagation}}$$

θ is a parameter, e.g. weights, biases and activations.

η is the learning rate.

∇ is the gradient, which is taken from J.

optimizer: SGD

J is known as an objective function.

Our final model summary:

		from	n	params	module	arguments
Backbone	0	-1	1	3520	models.common.Conv	[3, 32, 6, 2, 2]
	1	-1	1	18560	models.common.Conv	[32, 64, 3, 2]
	2	-1	1	18816	models.common.C3	[64, 64, 1]
	3	-1	1	73984	models.common.Conv	[64, 128, 3, 2]
	4	-1	2	115712	models.common.C3	[128, 128, 2]
	5	-1	1	295424	models.common.Conv	[128, 256, 3, 2]
	6	-1	3	625152	models.common.C3	[256, 256, 3]
	7	-1	1	1180672	models.common.Conv	[256, 512, 3, 2]
	8	-1	1	1182720	models.common.C3	[512, 512, 1]
	9	-1	1	656896	models.common.SPPF	[512, 512, 5]
Neck	10	-1	1	131584	models.common.Conv	[512, 256, 1, 1]
	11	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
	12	[-1, 6]	1	0	models.common.Concat	[1]
	13	-1	1	361984	models.common.C3	[512, 256, 1, False]
	14	-1	1	33024	models.common.Conv	[256, 128, 1, 1]
	15	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
	16	[-1, 4]	1	0	models.common.Concat	[1]
	17	-1	1	90880	models.common.C3	[256, 128, 1, False]
	18	-1	1	147712	models.common.Conv	[128, 128, 3, 2]
	19	[-1, 14]	1	0	models.common.Concat	[1]
	20	-1	1	296448	models.common.C3	[256, 256, 1, False]
	21	-1	1	590336	models.common.Conv	[256, 256, 3, 2]
	22	[-1, 10]	1	0	models.common.Concat	[1]
	23	-1	1	1182720	models.common.C3	[512, 512, 1, False]

Head or Final Yolo detection layer:

```
24 [17, 20, 23] 1 16182
models.yolo.Detect [8, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
```

Model summary:

```
Model summary: 270 layers, 7022326 parameters, 7022326 gradients
```

Here you can see,

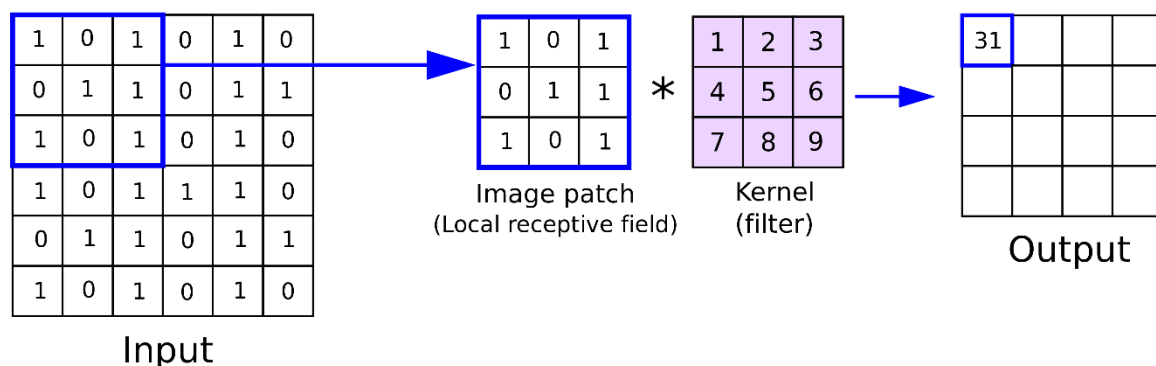
The total number of trainable parameters is – **7022326** and

The total number of layers is **270**.

3.7 Layer summary:

Convolution layer:

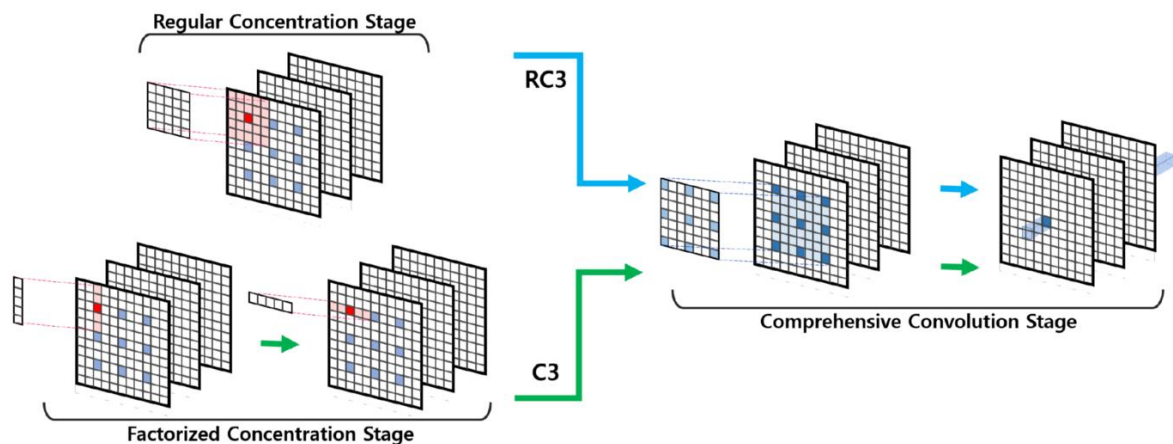
This layer makes a convolution piece that is convolved with the layer contribution to deliver a tensor of results. Assuming use_bias is True, a predisposition vector is made and added to the results. At last, if actuation isn't None, it is applied to the results too. When involving this layer as the primary layer in a model, give the watchword contention input_shape (tuple of whole numbers or None, does exclude the example hub), for example, input_shape=(128, 128, 3) for 128x128 RGB pictures in data_format="channels_last". You can utilize None when an aspect has variable size.



Source: anhreynolds.com

Concentrated-comprehensive convolution layer (C3):

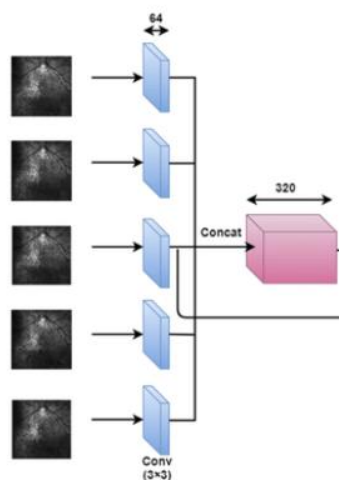
It applies the asymmetric convolutions before the depth-wise separable dilated convolution to compensate for the information loss due to dilated convolution. C3 is applied to the ESPnet to achieve better performance by reducing the number of parameters by half.



Source: sh-tsang.medium.com

Concat Layer:

Concat is just stacking channels. For example, the x1 layer has 256 channels, and the x2 layer has 256 channels. The concat layer combines these two layers, merging result will have 512 channels.



Source: researchgate.net

Upsampling Layer:

There are many convolutional or pooling layers used in the model, which downsamples the input image to an almost unusable image. Upsampling does the inverse of what convolutional and pooling do, it upsamples the image to make it input ready for another layer.

The syntax of Upsampling is:

```
CLASS torch.nn.Upsample(size=None, scale_factor=None, mode='nearest', align_corners=None,
                        recompute_scale_factor=None) [SOURCE]
```

In our model we have used `torch.nn.Upsample [size=None, scale_factor=2, mode='nearest']` on layer number 11 and 15.

```
tensor([[[[ 1.,  2.],
           [ 3.,  4.]]]])

>>> m = nn.Upsample(scale_factor=2, mode='nearest')
>>> m(input)
tensor([[[[ 1.,  1.,  2.,  2.],
           [ 1.,  1.,  2.,  2.],
           [ 3.,  3.,  4.,  4.],
           [ 3.,  3.,  4.,  4.]]]]])
```

Nearest Neighbor

1	2
3	4



1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Input: 2 x 2

Output: 4 x 4

Source: [GitHub](#)

4 Setup & Evaluation

4.1 Dependencies And Requirements

To run the final inference file Live_video.py there are lots of requirements that should be pre-installed using the pip command. Yolo model was trained on using a lot of libraries. Our approach is transfer learning with Yolo, hence these libraries are prerequisites for our model to train and test. Here are some major libraries which are used for training the model.

```
1 #basic requirements
2 ipython==8.4.0
3 kiwisolver==1.4.3
4 numpy==1.22.4
5 opencv-python==4.6.0.66
6 Pillow==9.1.1
7 tqdm==4.64.0
8 PyYAML==6.0
9 scipy==1.8.1
10
11 #Torch
12 torch==1.11.0
13 torchaudio==0.11.0
14 torchvision==0.12.0
15
16 #Tensorboard
17 tensorboard==2.9.1
18 tensorboard-data-server==0.6.1
19 tensorboard-plugin-wit==1.8.1
20
21 # Plotting
22 pandas==1.4.2
23 seaborn==0.11.2
24 matplotlib==3.5.2
25 matplotlib-inline==0.1.3
```

Installing Libraries

IPython offers a comprehensive toolset. Its principal parts are a strong Python shell that is interactive. a Jupyter kernel for use with Python code in interactive frontends like Jupyter notebooks like in Google colab. Data structures for multi-dimensional arrays and matrices are included in NumPy. On arrays, it is used to carry out a variety of mathematical operations, including trigonometric, statistical, and algebraic operations. Consequently, there are many mathematical, algebraic, and transformation functions in the library. OpenCV (Open-Source Computer Vision Library) is a free software library for computer vision and machine learning. It has 2500+ optimized algorithms included in the library, that contain a comprehensive mix of both classic and cutting-edge computer vision and machine learning techniques. These methods may be used to recognize objects, track moving objects, extract 3D models of objects, create 3D point clouds from stereo cameras, detect comparable photos in an image library etc. Pillow is being used somehow similar to Opencv for image manipulation. tqdm is

used for long loops. A smart progress bar can be produced using the Python module `tqdm` by wrapping it around any iterable. A `tqdm` progress bar displays the expected time left for the iterable in addition to the amount of time that has already passed. Built on the NumPy module of Python, SciPy is a collection of mathematical algorithms and useful functions. Giving the user advanced commands and classes for data manipulation and visualization significantly increases the power of an interactive Python session.

PyTorch is a Torch and Python-based Deep Learning tensor library that is mostly utilized in CPU and GPU applications. Since PyTorch uses dynamic computation graphs and is entirely Pythonic, it is preferred over other deep learning frameworks like TensorFlow and Keras. Torchaudio is a PyTorch module for processing audio and signals. It includes routines for I/O, signal and data processing, datasets, model implementations, and application components.

For plotting purposes, Pandas and seaborn libraries have been used. For manipulating the dataset and performing various tasks on the dataset pandas library is used. the seaborn library is used to plot the different graphs based on the dataset and visualize the data in various formats like a heat map, scatter plots, box plots, etc

Matplotlib is a Python package that allowed us to create static, animated, and interactive visualizations. Matplotlib makes simple things simple and difficult things possible. We can create publication-quality plots, make different file formats, customize visuals, and layouts and make interactive figures which can update and zoom.

4.2 GPU accessing using CUDA and cuDNN :

The final model was tested on a device with Nvidia 1650ti GPU which supports torch Cuda (Compute Unified Device Architecture). These will use the power of GPU and help the model to classify objects in less time than using just the CPU power of the device.

CUDA :

CUDA is a parallel computing platform and an application programming interface that allows the software to use certain types of graphics processing units for general-purpose processing – an approach called general-purpose computing on GPUs.

CUDA Driver:

Running a CUDA application requires the system with at least one CUDA-capable GPU and a driver that is compatible with the CUDA Toolkit. Each release of the CUDA Toolkit requires a minimum version of the CUDA driver. The CUDA driver is backwards compatible, meaning that applications compiled against a particular version of the CUDA will continue to work on subsequent (later) driver releases.

CUDA Toolkit	Minimum Required Driver Version for CUDA Minor Version Compatibility*	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 11.6.x	>=450.80.02	>=452.39
CUDA 11.5.x	>=450.80.02	>=452.39
CUDA 11.4.x	>=450.80.02	>=452.39
CUDA 11.3.x	>=450.80.02	>=452.39
CUDA 11.2.x	>=450.80.02	>=452.39
CUDA 11.1 (11.1.0)	>=450.80.02	>=452.39
CUDA 11.0 (11.0.3)	>=450.36.06**	>=451.22**

Source: [Nvidia.com](https://nvidia.com)

CUDA Libraries

CUDA Math Libraries toolchain uses C++11 features, and a C++11-compatible standard library (libstdc++ >= 20150422) is required on the host. CUDA Math libraries are no longer shipped for SM30 and SM32.

Support for the following compute capabilities is deprecated for all libraries:

sm_35 (Kepler)

sm_37 (Kepler)

sm_50 (Maxwell)

cuDNN:

cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. It allows them to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning.

- Arbitrary dimension ordering, striding, and sub-regions for 4d tensors mean easy integration into any neural net implementation
- Speed up fused operations on any CNN architecture
- Runtime fusion to compile kernels on the fly with new operators, heuristics, and fusions
- ease of use, application integration, and offers greater flexibility to developers.

4.3 Live_video.py

For testing, this file: Live_video.py file should be run in the working directory. A new environment should be set up in python (version 3.7). We used the conda environment and Pycharm as our IDE. conda environment is easy to set up and easy to handle the dependencies and requirements. We find Pycharm user-friendly for coding and handling file and their path.

```
import random
import cv2
import torch
import numpy as np

from models.experimental import attempt_load
from utils.dataloaders import letterbox
from utils.general import check_img_size, non_max_suppression, scale_coords
from utils.torch_utils import select_device
```

Importing libraries

The libraries are being imported as explained in the report. Lots of functions are also being imported from the utils file. It usually contains functions that are required from lots of places in the code. It seems logical to save code from the number of classes and namespaces that contain one or two functions. In our code, various utils classes have been used such as experimental, torch_utils, dataloaders etc.

The computational power can be changed by assigning CPU/GPU to the map_location function.

```
self.device = select_device("GPU") # for CPU device Change to cpu
self.model = attempt_load(weights) # load FP32 model
self.stride = int(self.model.stride.max()) # model stride
self.imgsz = check_img_size(img_size, s=self.stride) # check img_size
```

The function is called back from this file: <model/experimental.py>. This should also be changed.

```
model = Ensemble()
for w in weights: if isinstance(weights, list) else [weights]:
    ckpt = torch.load(attempt_download(w), map_location='GPU:0' # load GPU ; for CPU= 'cpu'
    ckpt = (ckpt.get('ema') or ckpt['model']).to(device).float() # FP32 model
    model.append(ckpt.fuse().eval() if fuse else ckpt.eval()) # fused or un-fused model in eval mode
```

After this, if the input is a video file. With the help of the OpenCV library, the video file is being converted to lots of images from video frames and this model will classify these images as

being trained on images. The image will transform the array with detect function in the code. The model weight is loaded and classify the images. The YOLOV5_Detector has a function to change the iou_thresh value (intersection over union threshold) and the confidence threshold value. These can be adjusted as per the required output quality.

<\Image.py>

```
img = cv2.imread(r"TestImages/Test (1).jpg")
# img = cv2.imread(r"TestImages/Test (2).jpg")
# img = cv2.imread(r"TestImages/Test (3).jpg")
```

For image input

<\Live_video.py>

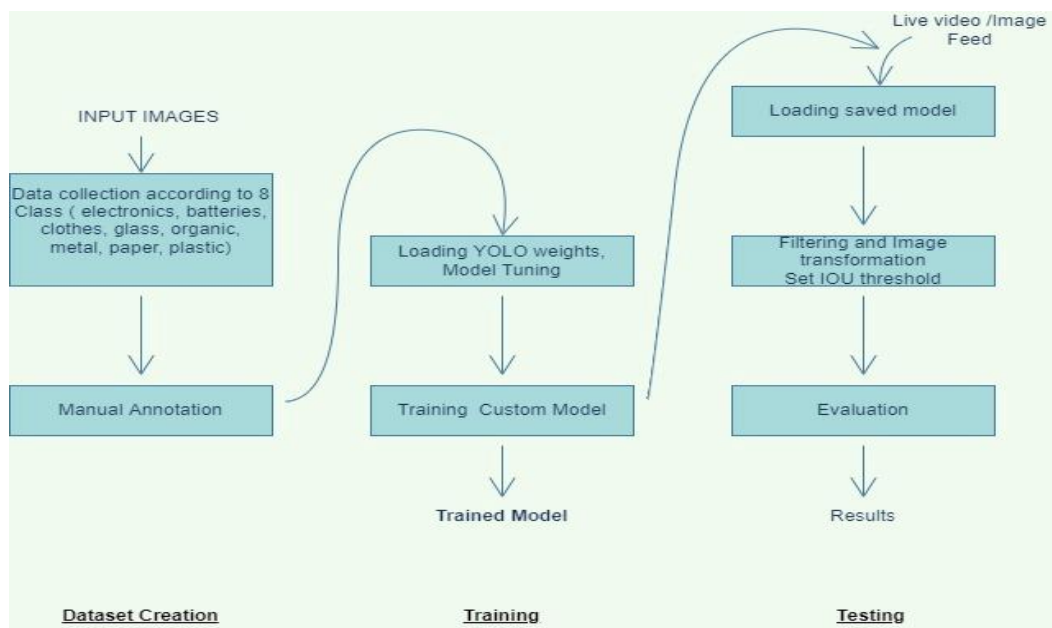
```
video_path = 0 # 0 or 1 for using device or attached external camera use.
vid = cv2.VideoCapture(video_path)
```

For camera video or video file input

The Input can be handled by writing file location and changing the variable to 0\1.

4.4 Model Architecture

The image below explains Our model architecture pipeline.





Model Architecture diagram

As mentioned in the previous report the Dataset for model training was created. After annotating the Image Model is trained. For Testing the live video or images of the photo are fed to the model. So, a video file feed or camera is attached to the end testing device. Switching between these feed data can be easily handled with the Live_video.py file, as mentioned earlier. The Data will go through the model and classify the object. The result will be saved in the working directory and shown live as being evaluated.

4.5 Bill of Material:

We need a webcam and tripod for this project. A webcam is required for capturing video and a tripod is required to hold the webcam. Video captured can be fed into the machine learning model (YOLO V5).

S.No.	Product	Qty.	Remark
1.	Logitech C270 webcam 	01	For video Input
2.	Tripod for camera 	01	For mounting the camera on the table

4.6 Training Results

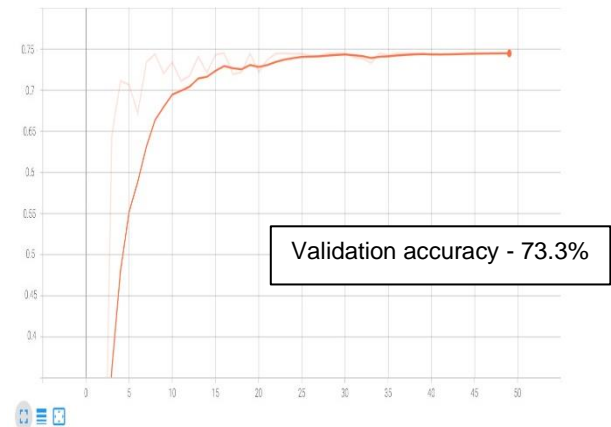
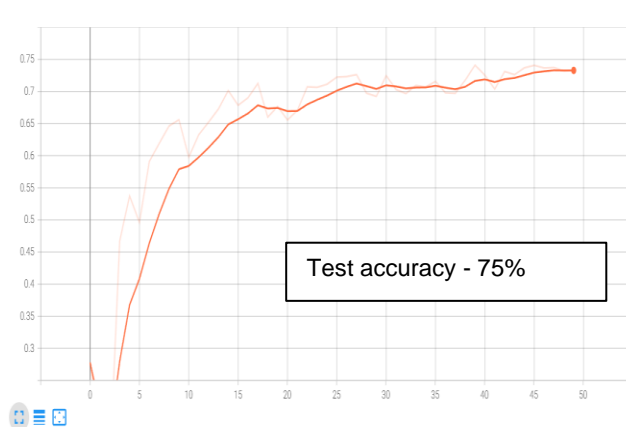
We run the model with 50 epochs and with an SGD optimizer.

We used the Torch library's TensorBoard to visualize the results. TensorBoard is a tool for providing measurements and visualization during the machine learning workflow. It enables tracking experiment metrics like loss and accuracy, visualizing the model graph, projecting embeddings to a lower-dimensional space, and much more.

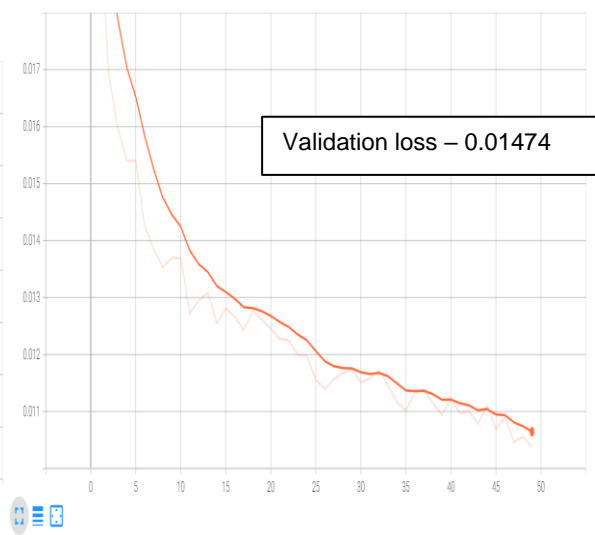
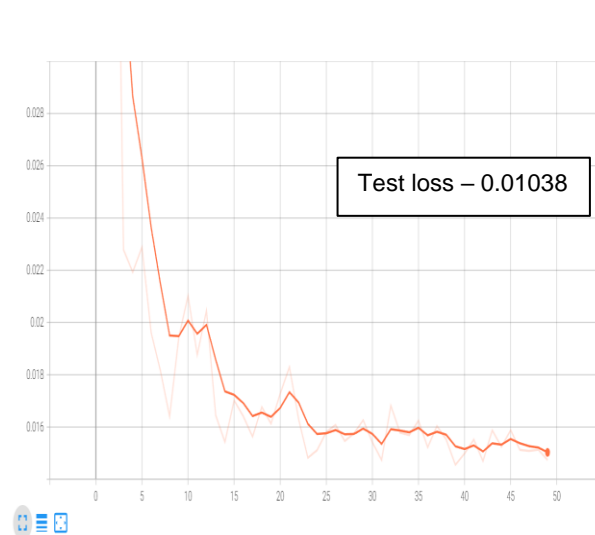
After running the training command, it stores the Test and Validation's accuracy and loss in folder named runs. The following command loads the tensorboard and executes the `--logdir` from the runs folder.

▼ Show Training Results

```
[ ] # Start tensorboard
    # Launch after you have started training
    # logs save in the folder "runs"
    %load_ext tensorboard
    %tensorboard --logdir runs
```



Test and Validation Accuracy



Test and Validation loss Graph

5 Result and Evaluation

Results and Summary:

- Our machine learning model can detect objects in the video frame.
- The model can classify object-based on a confidence level.
- Objects are classified into 8 different classes as per our project object.
- Model accuracy is about 75%.
- Model accuracy can be improved further with the following:
 - By tuning hyper parameters.
 - Adding more datasets.



Metal



Clothes



Plastic



Electronic



Organic

[Prediction results from model](#)

Limitations:

- Since the model has 75% so it can not classify complex images.
- If there are more than 3 objects in a frame then prediction reduces.
- Sometimes model confuses between metal and glass.
- The video framerate is shown to be less than 30 fps.
- Model required good GPU and CPU for smooth frame rate.

Future Scope:

- A robotic arm can be employed for waste segregation.
- This way we can achieve waste recognition and segregation.
- This complete system can be used for waste segregation and later waste can be recycled.
- The model can be deployed on a mobile application, it will give portability.
- If the model is utilised properly then the recycled waste can be used efficiently rather than in landfills.

References

- [1] J. B. T. J. Myra G. Flores, "AUTOMATED WASTE SEGREGATION," Literature Review of Automated Waste Segregation System using Machine, p. 15.3 TO 15.5, 2015.
- [2] Waste Classification System Using Deep Learning, pp. 608-610, 2019.
- [3] Meena. U, "Information and Computational Science," Waste Segregation Using Deep Learning, vol. Volume 10, no. Issue 5 - 2020, pp. 717-718, 2020.
- [4] Sylwia Majchrowska Wroclaw University of Science and Technology Wrocław, Poland, 2) Agnieszka Mikołajczyk Gdansk University of Technology ´ Gdansk, Poland, "WASTE DETECTION IN POMERANIA: NON-PROFIT PROJECT FOR DETECTING WASTE IN ENVIRONMENT " in arXiv:2105.06808v1 [cs.CV] 12 May 2021
- [5] Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.
- [6] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778.
- [7] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105.
- [8] Wu, H., Zheng, S., Zhang, J., and Huang, K. (2019). Gp-gan: Towards realistic high-resolution image blending. In Proceedings of the 27th ACM International Conference on Multimedia pages 2487–2495. ACM.
- [9] Yang, M. and Thung, G. (2016). Classification of trash for recyclability status. CS229 Project Report, 2016.
- [10] https://pytorch.org/hub/ultralytics_yolov5/
- [11] Karen Simonyan, Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition" in arXiv:1409.1556 [cs.CV] 10 April 2015

Appendix

1	absl-py==1.1.0	32	protobuf==3.19.4
2	asttokens==2.0.5	33	psutil==5.9.1
3	backcall==0.2.0	34	pure-eval==0.2.2
4	cachetools==5.2.0	35	pyasn1==0.4.8
5	certifi==2022.5.18.1	36	pyasn1-modules==0.2.8
6	charset-normalizer==2.0.12	37	Pygments==2.12.0
7	colorama==0.4.4	38	pyparsing==3.0.9
8	cycler==0.11.0	39	python-dateutil==2.8.2
9	decorator==5.1.1	40	pytz==2022.1
10	executing==0.8.3	41	PyYAML==6.0
11	fonttools==4.33.3	42	requests==2.28.0
12	google-auth==2.8.0	43	requests-oauthlib==1.3.1
13	google-auth-oauthlib==0.4.6	44	rsa==4.8
14	grpcio==1.46.3	45	scipy==1.8.1
15	idna==3.3	46	seaborn==0.11.2
16	importlib-metadata==4.11.4	47	six==1.16.0
17	ipython==8.4.0	48	stack-data==0.2.0
18	jedi==0.18.1	49	tensorboard==2.9.1
19	kiwisolver==1.4.3	50	tensorboard-data-server==0.6.1
20	Markdown==3.3.7	51	tensorboard-plugin-wit==1.8.1
21	matplotlib==3.5.2	52	thop==0.1.0.post2206102148
22	matplotlib-inline==0.1.3	53	torch==1.11.0
23	numpy==1.22.4	54	torchaudio==0.11.0
24	oauthlib==3.2.0	55	torchvision==0.12.0
25	opencv-python==4.6.0.66	56	tqdm==4.64.0
26	packaging==21.3	57	traitlets==5.2.2.post1
27	pandas==1.4.2	58	typing_extensions==4.2.0
28	parso==0.8.3	59	urllib3==1.26.9
29	pickleshare==0.7.5	60	wcwidth==0.2.5
30	Pillow==9.1.1	61	Werkzeug==2.1.2
31	prompt-toolkit==3.0.29	62	zipp==3.8.0

All required Libraries

```

class YOLOV5_Detector:
    def __init__(self, weights, img_size, confidence_thres, iou_thresh, agnostic_nms, augment):
        self.weights = weights
        self.imgsz = img_size
        self.conf_thres = confidence_thres
        self.iou_thresh = iou_thresh

        self.agnostic_nms = agnostic_nms
        self.augment = augment

        self.device = select_device("")
        self.model = attempt_load(weights, map_location=self.device) # load FP32 model
        self.stride = int(self.model.stride.max()) # model stride
        self.imgsz = check_img_size(img_size, s=self.stride) # check img_size

```

Setting up the function for model training

```

def plot_one_box(self, x, img, color=None, label=None, line_thickness=3):
    # Plots one bounding box on image img
    tl = line_thickness or round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1 # line/font thickness
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
    cv2.rectangle(img, c1, c2, color, thickness=tl, lineType=cv2.LINE_AA)
    if label:
        tf = max(tl - 1, 1) # font thickness
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=tf)[0]
        c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
        cv2.rectangle(img, c1, c2, color, -1, cv2.LINE_AA) # filled
        cv2.putText(img, label, (c1[0], c1[1] - 2), 0, tl / 3, [225, 255, 255], thickness=tf, lineType=cv2.LINE_AA)

```

Bounding box around the identified class

```

detector = YOLOV5_Detector(weights=r'K:\cs\yolo\Object detection\best.pt',
                             img_size=640,
                             confidence_thres=0.2,
                             iou_thresh=0.5,
                             agnostic_nms=True,
                             augment=True)

```

Importing model weights