Transfer Learning for Object Detection with Deep Convolutional Network

Sugunadevi Palaniappan
Intelligent Systems, Technische Hochschule Ulm

Abstract:

Object detection is a key topic in image processing and computer vision. Various algorithms are developed for this purpose. Recently, deep learning algorithms gave promising results compared to traditional methodologies. Warehouse Automation is primary focus for many eCommerce giants such as Amazon, eBay. High labour costs, inaccurate inventory, redundant processes are major problems faced in warehouses. So, this paper is driven by a particular use case where customers send back ordered items, and the retailer wants to automatically unpack the returned box and sort the returned items. For this use case, evaluation is done whether this could be achieved by Deep Convolutional Neural Networks and whether transfer learning allows to adopt to new items. An overview on object detection with deep convolutional neural networks (DCNNs) and an overview on transfer learning are also provided in this survey. Added to this, implementation results of existing object detection model are shown. Finally, the implementation results of own customized object detection model inside the box are depicted.

Index Terms: Object detection, Transfer learning, Deep Convolutional Networks (DCNNs)

1 Introduction

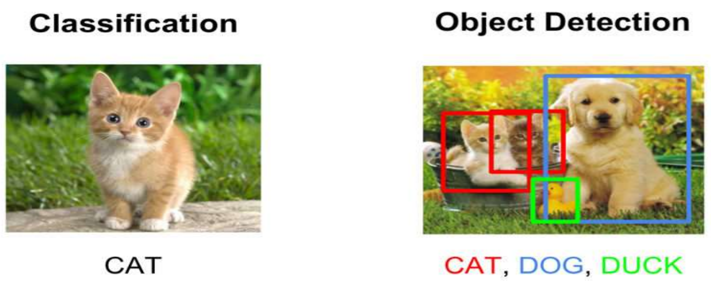


Figure 1: Computer vision tasks[30]

Object Recognition refers in general about identifying objects in images or videos. Smartphone unlocking is a good example for facial recognition, while face is detected initially, then it classifies it as human face and then the detected face is checked whether it belongs to product owner or not. The performance is measured by precision and recall. [10]

Image classification is task of assigning an input image one label from a fixed set of categories. The performance is evaluated by mean classification error on predicted objects. [10]

Object Localization is about locating the objects in an image with bounding boxes(x, y, width, height). The performance is evaluated by the distance between the expected and predicted bounding box on the objects. [10]

Object detection provides location information about the objects in an image along with class label. [30]

In Image segmentation, the exact shapes of the objects are predicted. Here, images are partitioned into segments to simplify the analysis of images. There are two types of segmentation namely Semantic segmentation and Instance segmentation. [30]

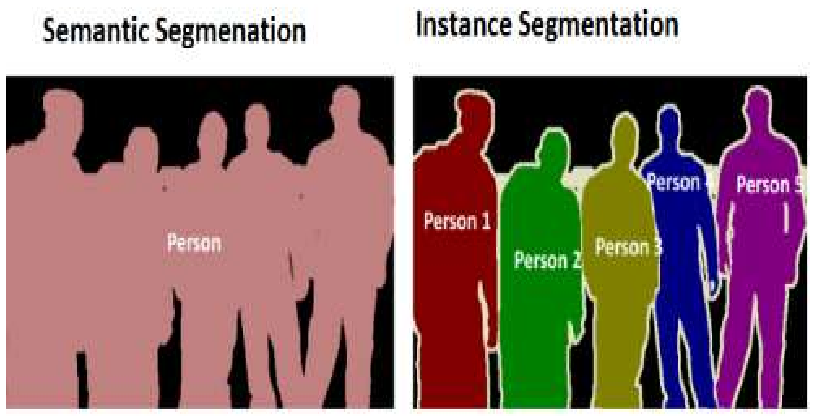


Figure 2: Computer vision tasks [10]

In Semantic Segmentation, each pixel in an image is assigned to a class label. In the above example of semantic segmentation, black pixels belong to class background and pink pixels belongs to class person. In Instance Segmentation, as before, each pixel is assigned to a particular class, in addition, different objects belonging to the same class are differentiated with different colours, i.e., in above example, different persons of class Person are differentiated with different colours. [10]

Practical Applications

Practical applications include security camera surveillance, Medical Image analysis, Autonomous driving, satellite Image analysis and Warehouse Automation. [1]

2 Theoretical Background

2.1 Deep Convolutional Neural Networks (DCNNs)

General Architecture

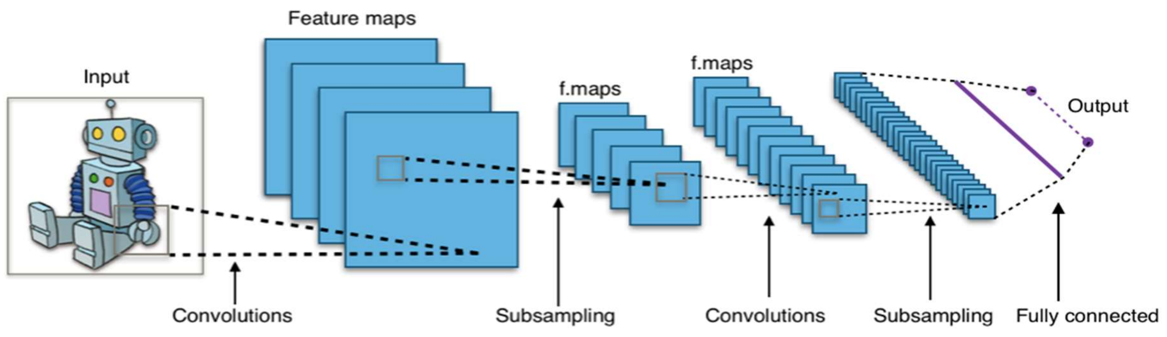


Figure 3: Generic Convolutional Neural Network Architecture [13]

Deep Convolutional neural networks consist of input layers, hidden layers and output layer. The hidden layers are convolutional layers in addition to other layers such as pooling layers. The last stage is made up of one or more fully connected layers.

Convolution layer

Feature extraction is performed in this layer. Convolution is just another way of calculating W^T*X. It is also termed as Convolved Feature or Activation Map. Convolutional neural networks automatically estimate the weights of filter.[9].

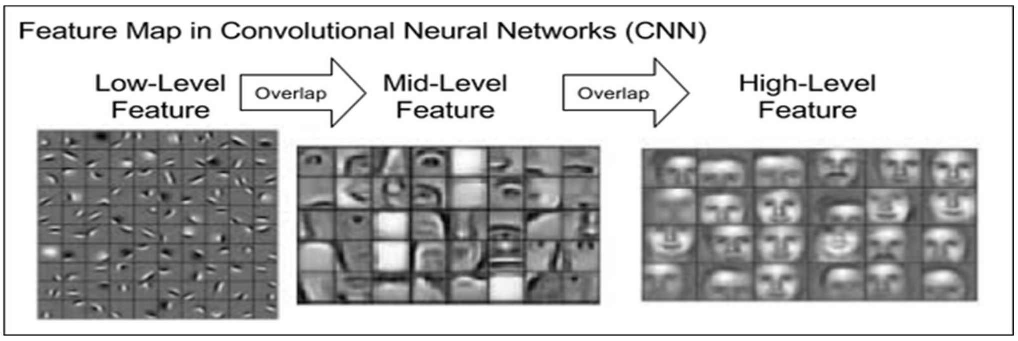


Figure 4: Examples of Feature map [27]

In the above example for face detection, initially, low level features such as edges are detected, later mid-level features like eyes, nose, ears are detected, and later high-level features of few faces are extracted.[27]. Padding is used to preserve the original dimension of corners of input images. Addition of number of zeros in padding depends upon the size of the kernel.[9]

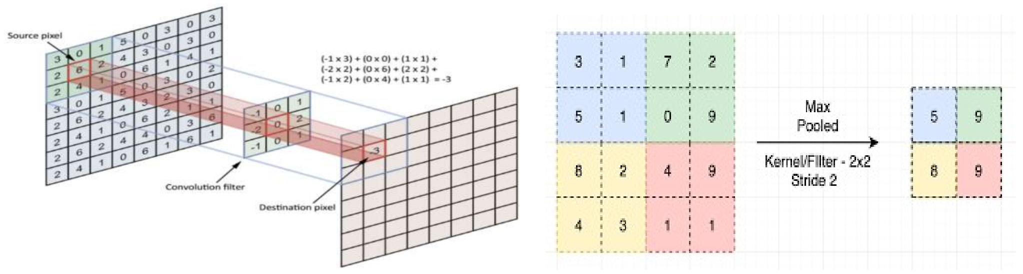

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
Destination pixel
Max Pooled
Kernel/Filter - 2x2
Stride 2

Figure 5: Convolutional layer & Pooling Layer [14]

Pooling Layer

Pooling layer is another building block of CNN which operates on each feature map. Down sampling is done using pooling layer. Pooling function such as max pooling or average pooling reduce the spatial size of the representation to minimize the computation in network.[9]

Fully Connected Layer

The final building block of CNN is fully connected layers.It is feed forward neural networks[14]

2.2 Traditional Object detection Methods

The pipeline of traditional object detection model has been divided into three stages namely

- Information Region Selection,
- Manual Feature Extraction and
- Classification[11]

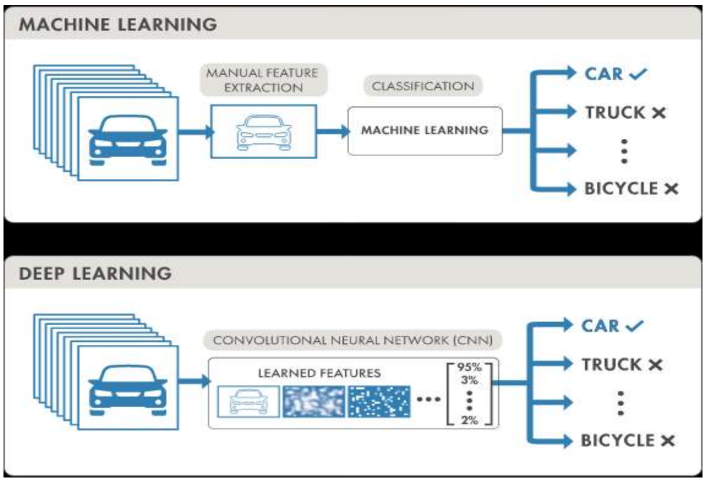


Figure 6: Comparison between Traditional & Deep learning based Object detection [11]

For Region Selection, in most cases, the sliding window technique is used where rectangular window slides through the entire image. Every part of the image is captured by sliding through the entire image. It is exhausting strategy to find all possible positions of objects in an image[17]

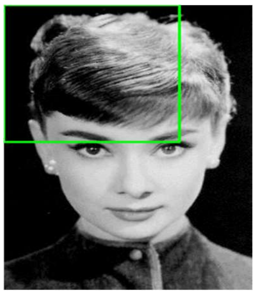


Figure 7: Example of sliding window approach.[17]

Feature extraction is done manually. Harris Corner Detection, Scale-Invariant Feature Transform (SIFT) are some techniques used.[17]

Classification could be done by logistic regression, support vector machine (SVM), Adaptive Boosting etc. Commonly, SVM is used due to their high performance in small training datasets. When using HOG descriptor and linear SVM for object classification, multiple bounding boxes are drawn around objects. So, reducing of number of bounding boxes on an object to its actual count is termed as non-maxima suppression.[17]

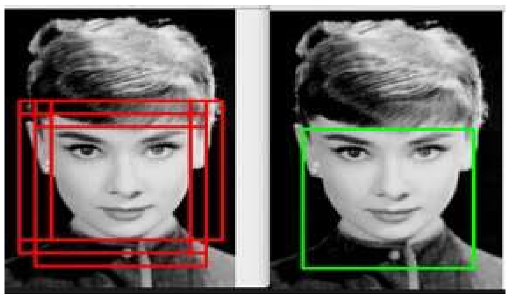


Figure 8: example of non-maxima suppression [17]

Some examples of traditional object detection models are Histogram of Oriented Gradients and Haar feature. [17]

2.3 Deep learning-based Object Detection Frameworks

After the success of using Deep Convolutional Neural Networks for image classification, object detection progressed more efficiently. [28]

There are two types of deep learning-based object detectors namely,

- one stage detectors
- two stage detectors (Region Proposal based Frameworks)[28]

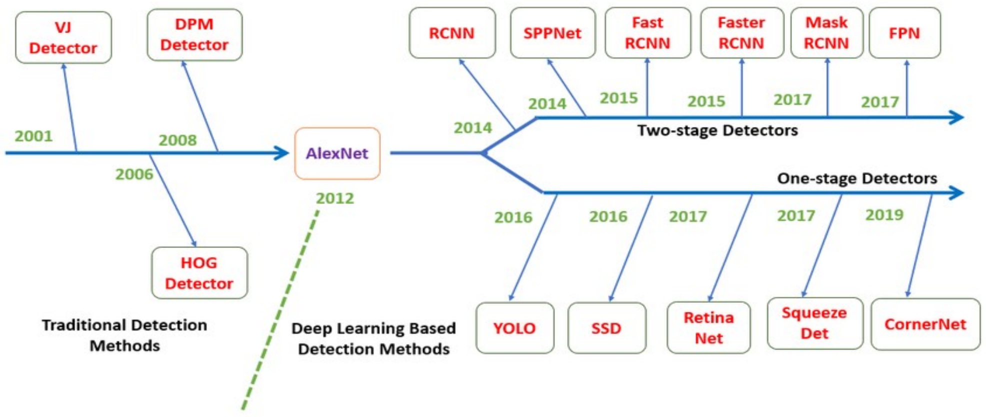


Figure 9: Milestone of Object detectors [28]

Region based CNN(RCNN), Mask RCNN, Fast Region Based CNN (Fast-RCNN), Feature Pyramid Network (FPN) are some two stage detectors. The performance of detection is better in two stage detectors. [3].

Single Shot Detector(SSD), You Only Look Once(YOLO), Retina Net, Squeeze Det, CornerNet are some one stage detectors. [3].

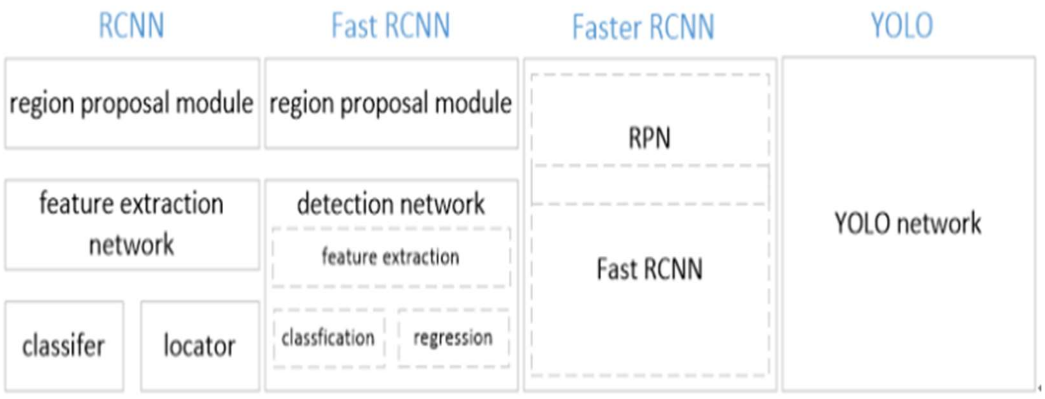


Figure 10**:** Modules of various two stage detectors [24]

In RCNN, there are lots of duplicated computations as the features of each region proposal are extracted by DCNNs separately. So, training and testing are time-consuming. [24]

Region Proposal Module:

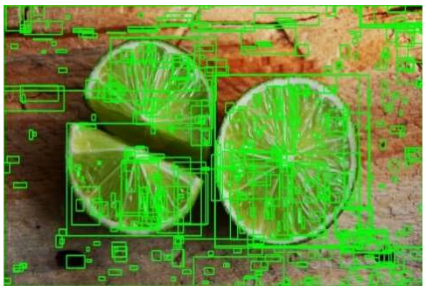


Figure 11: example of selective search[17]

There are several approaches to generate region proposals. Selective search is a clustering-based approach which attempts to group pixels and generate proposals based on the generated clusters. A sliding window is another technique used for region proposals. [17]

Feature extraction:

Using Deep Convolutional network, features from each candidate regions are extracted. The goal of feature extraction is to obtain fixed set of visual features.[17]

Classifier:

Features are classified as one of the known class. [22]

Evaluation Metrics

- Frames per second (FPS)
- mean average precision (mAP)
- precision-recall curve (PR curve)
- intersection over union (IoU)[18]

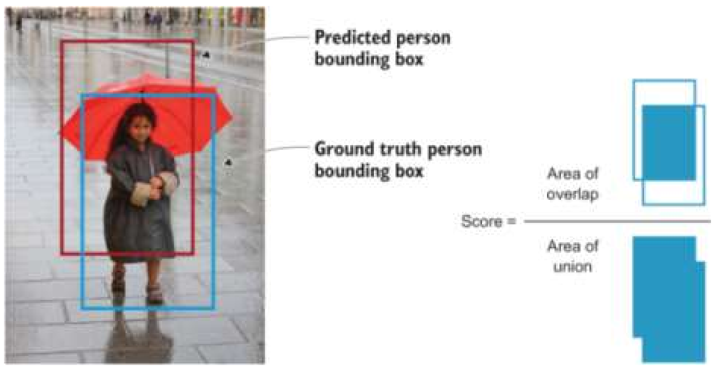

Predicted person bounding box
Ground truth person bounding box
Area of overlap
Score =
Area of union

Figure 12: IOU Score between ground truth bounding box and predicted bounding box [18]

Two Approaches to object detection with deep learning

There are two approaches for object detection using deep learning.

1. Training a model from scratch
2. Using a pre-trained model [8]

When there are enough training samples, model could be trained from scratch, it might take many hours or even days. Using a pre-trained model is about using existing model trained on millions of images termed as transfer learning.[8]

2.4 Transfer Learning

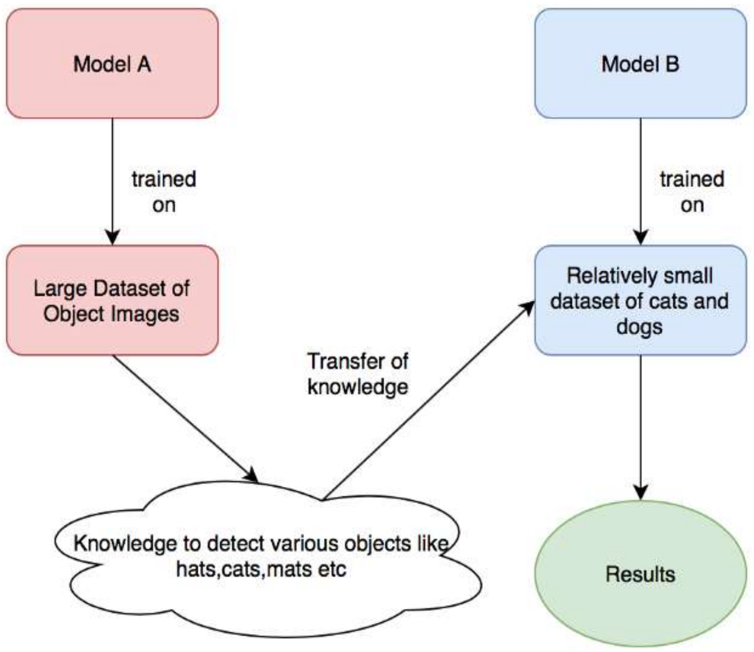

Model A
trained on
Large Dataset of Object Images
Model B
trained on
Relatively small dataset of cats and dogs
Transfer of knowledge
Knowledge to detect various objects like hats,cats,mats etc
Results

Figure 13: Transfer Learning[8]

In the above example, knowledge gained in task 1- detecting Mats, Cats and hats can be applied to task 2- detecting Dogs and Cats.[8]

Transfer learning is knowledge sharing. Considering the real time example, if you are good at C++, Java Programming, then you can transfer your knowledge in learning new language Python. When you use pre-trained models to a new task, then it is transfer learning When we don’t have sufficient labelled datasets, we could opt for transfer learning. Knowledge gained from existing labelled model 1 helps in increasing the performance of model 2 which has smaller datasets. [7].

It takes a lot of work and time to train a deep neural network and requires costly GPUs. So, we can use Models trained by huge companies. It reduces tremendous amount of time and money to train new deep neural networks.[8]

Transfer learning Workflow

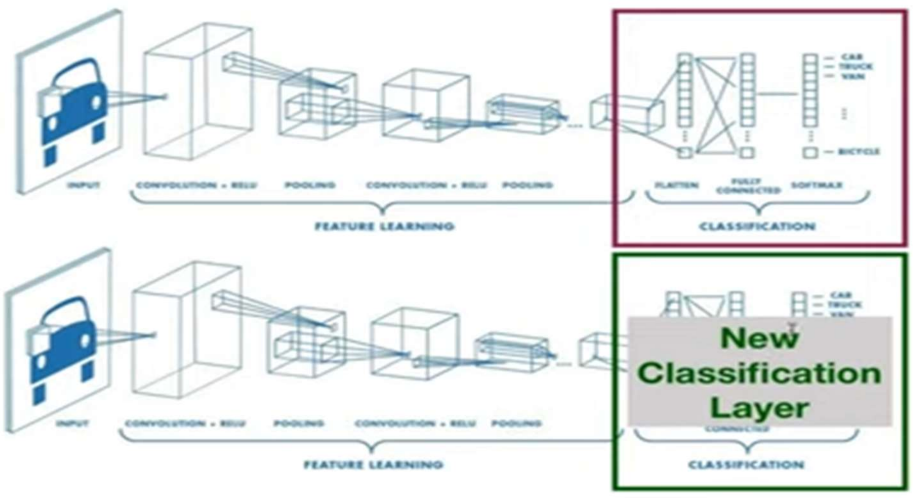


Figure 14: Transfer learning Workflow[29]

When model developed for first task can be reused as a starting point for the model on second task. It helps in rapid generation of new models. Training a completely new model from scratch is time-consuming. [29]

If we use pre-trained model, we exchange only the last layer and does not require the whole model to be trained again. Convolution and pooling layers of pre-trained models which are learned features are freezed without training. The last dense layer (classification layer) is replaced depending upon the own customized model. [29]

Transfer learning-Small Data

Considering the scenario of having small datasets, we can't train the model better than existing pre-trained model. So, we will just remove uppermost layers in the pre-trained model and retrain with our own new layers at top. During training of the new model, previous Model layers are locked and only the newly added layers are trained. Functions like Trainable_Parameters and Freezing specific layers can be used. [9]

Transfer learning-Mid-size Data

Here, for the first half of layers, we utilize the initial set of layers from the existing model and use pretrained weights. For the second half of layers, we can train the layers from beginning or start weight optimization from existing set of weights. [9]

Transfer learning- Enough Data

Even though, we have enough data to train the model. Pre-trained model can still be helpful. We can use pretrained weights as starting point to train the new model. [9]

Applications of Transfer learning

Learning from simulations

Gathering of datasets to train the model is very expensive and time consuming. Particularly in medical fields, data confidentiality is big hindrance in collecting data. In the field of autonomous driving and Robotics, we need lots of data to train the model efficiently. To solve the above problems, simulating artificial data will be a great solution.[4]

Adapting to new domains

Domain adaption is very important in computer vision. In voice recognition domains, different accents from native speakers, immigrants and children need to be adopted. In standard natural language processing, different text types/genres namely news data, social media messages should be adopted. Different visual domains adoption is also very important. Transfer learning helps in solving the above specified problems[4].

Transferring knowledge across languages

Cross lingual embedding models also plays a vital role in NLP. Cross lingual adaption methods with numerous labelled data in English could be used to languages with low resources.[4]

Drawbacks of Transfer Learning

General purpose CNN models such as AlexNet, VGGNet are trained with general images such as dog, cat, water bottles etc. Suppose you want to retrain the Model for Satellite Image Analysis or Medical Images such as Blood Segmentation, Lung Segmentation or MRI Scan Segmentation. There is no similarity between source domain (general purpose eg. VGG) and target domain (Medical). In this situation, transfer learning can't help much. [9].

3 Practical Work

3.1 Existing Model Implementation (Out-of-Box)

The existing model implementation is same as reference [31] without any modifications.

Configurations:

Dataset: MS COCO17 with 80 Object Classes

Training: ~118k images, val: ~5k images, Test:~41k images

Deep Learning Framework: PyTorch

Object detection Model: Faster R-CNN model (detecto)

Real time detection: Detect-live utility function

Programming Language: Python

Operating system: Windows10

Coding Platform: Jupyter notebook

Detecto is a Python package built on top of PyTorch that allows to build object detection models. Detecto's Model class is built on Faster R-CNN ResNet50 architecture. It is pre-trained on COCO 2017 dataset. This model is able to detect 80 object categories. Model is from torchvision's models sub-package. Real time detection is done by detecto.visualize.detect_live function. Default model is implemented here. [31]

Results from existing models

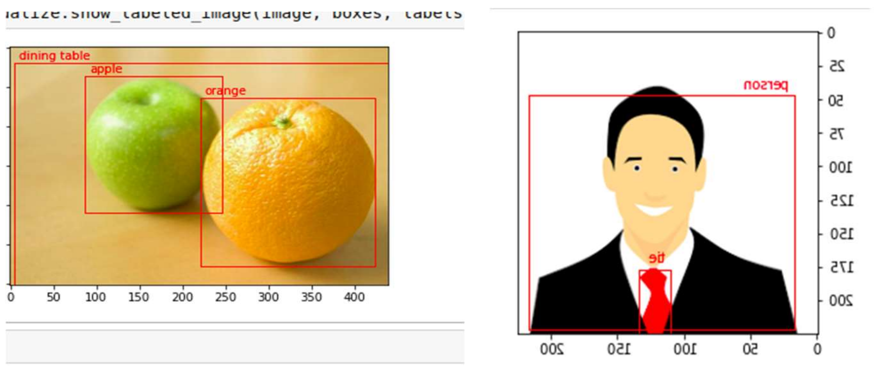

```
ualize.show_labeled_image(image, boxes, labels
```

Figure 15: Implementation of existing model

The above figures show results of object detections(out-of-box) on example images. Class labels are predicted correctly as apple, orange, dining table and person along with bounding boxes around the objects showing the location information.

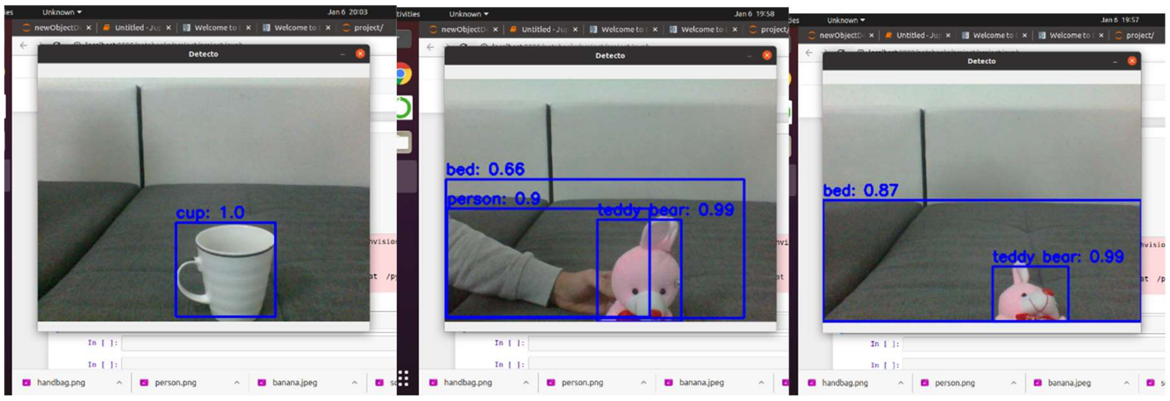


Figure 16: Implementation of existing model

The above figures show results of object detections(out-of-box) on real-time. Class labels are predicted correctly as cup, person, teddy bear and the confidence value of prediction boxes are very high enough nearing the probability 1. Bounding boxes are also drawn exactly around the objects in the images. So, results are reproduced exactly as the author.

3.2 Own Customized Implementation

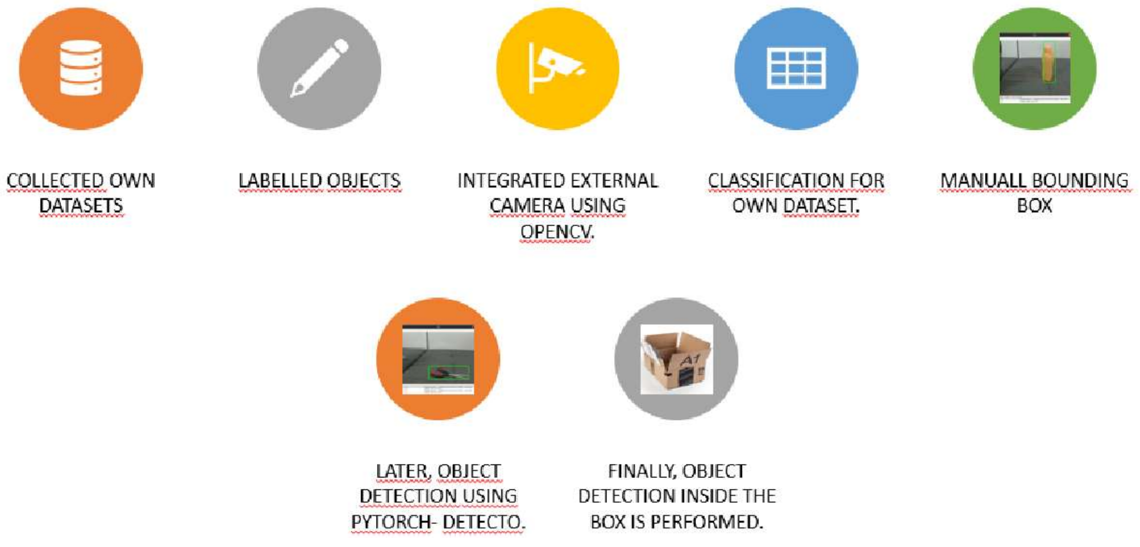

COLLECTED OWN DATASETS
LABELLED OBJECTS
INTEGRATED EXTERNAL CAMERA USING OPENCV.
CLASSIFICATION FOR OWN DATASET.
MANUALL BOUNDING BOX
LATER, OBJECT DETECTION USING PYTORCH- DETECTO.
FINALLY, OBJECT DETECTION INSIDE THE BOX IS PERFORMED.

Figure 17: Overview on modifications

From the implementation of existing model, basic ideas for detection are gained. To implement our use case (boxes returned by customers), the following modifications are done to existing model.

To detect objects inside the box, own data set is created. Just random objects around the study table are selected to implement this use case. There is no specific reason behind for choosing these datasets.

Location information about the objects in images are provided by labelling the objects in images. Data augmentation is done to overcome overfitting. Data pre-processing is done to transform raw data into appropriate form. Since its convenient to use external camera to detect objects inside a box, external camera is integrated using OpenCV library.

The idea for modelling 1 is obtained from reference [30], where classification is done, and bounding boxes are drawn manually. The weights and biases from pre-trained model MobileNetV2 are obtained using transfer learning.

In second modelling, object detection is performed using Faster-RCNN using Pytorch- Detecto which is same as existing model. In addition, the pre-processing of own dataset is done using torchvision.transforms module. Later, annotated images which are saved as xml files are converted into csv files. Further , optimization of hyperparameters is done.

Dataset Collection:

The dataset was created by collecting different types of objects namely gluestick, stapler, bottle, scissor and headset. The images of each class were collected in various brightness, contrasts, and angles. The collected images consist of one or more of the above four classes. A total of 310 images are collected, of which 292 images are in training set and 19 images are in validation set.

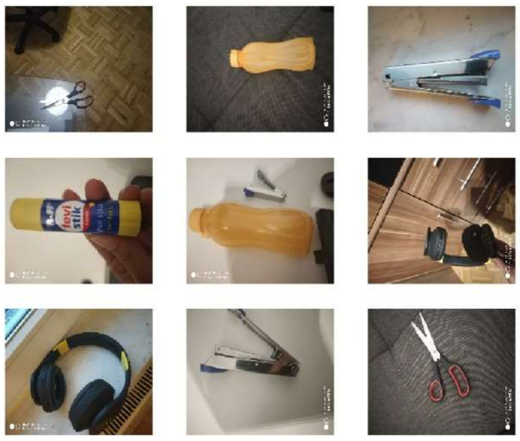


Figure 18: Dataset Collection

There is overfitting in the training phase, so augmentation is done. It increase the diversity of data by which performance of the model could be improved. Overfitting could also be avoided. Normally, deep learning convolutional neural network are trained with large datasets to get efficient results. Since we have small dataset, data augmentation is done to increase the size of training samples. For augmentation of images in our own customized dataset , RandomFlip ('horizontal') and RandomRotation(0.2) is done.

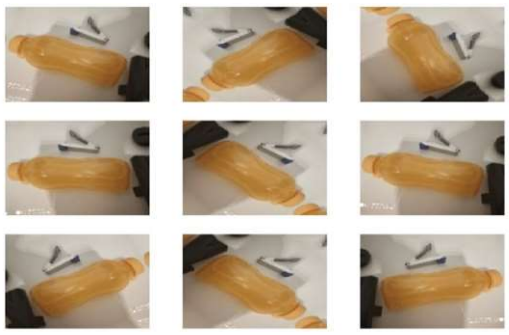


Figure 19: Data Augmentation

Dataset Labelling:

Labelling the objects by drawing a rectangle box manually are called as Annotation. The information about the location along with dimensions could be given to the Model by Annotation. Pre-processing steps such as resizing the image is to be done before performing annotation, because location and dimensions of the object varies after resizing. During my experiments, labelled the images before resizing. After annotation, dimensions have changed. So, the entire dataset was rescaled and once again labelled.

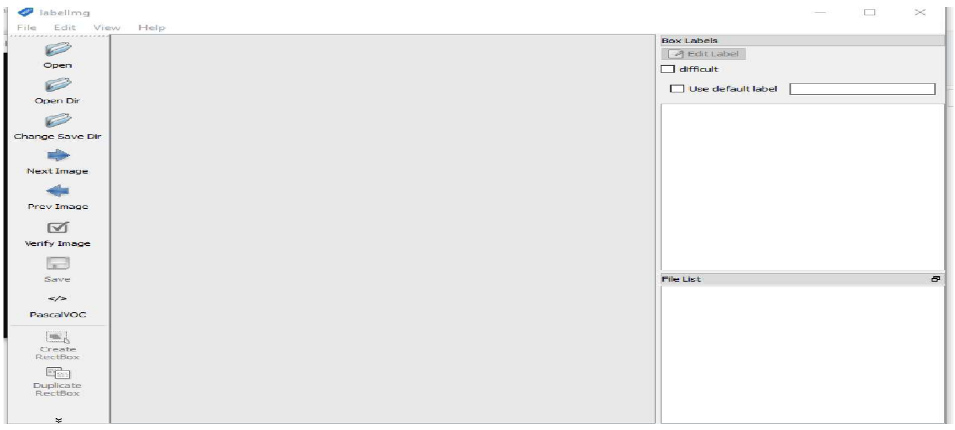


Figure 20: A view on user interface of labelImg tool is given.

The application named labelImg is used for the annotation of gluestick, stapler, bottle, scissor and headset. labelImg consists of several functionalities including creating a rectangle box, labelling objects, and saving it in xml format.

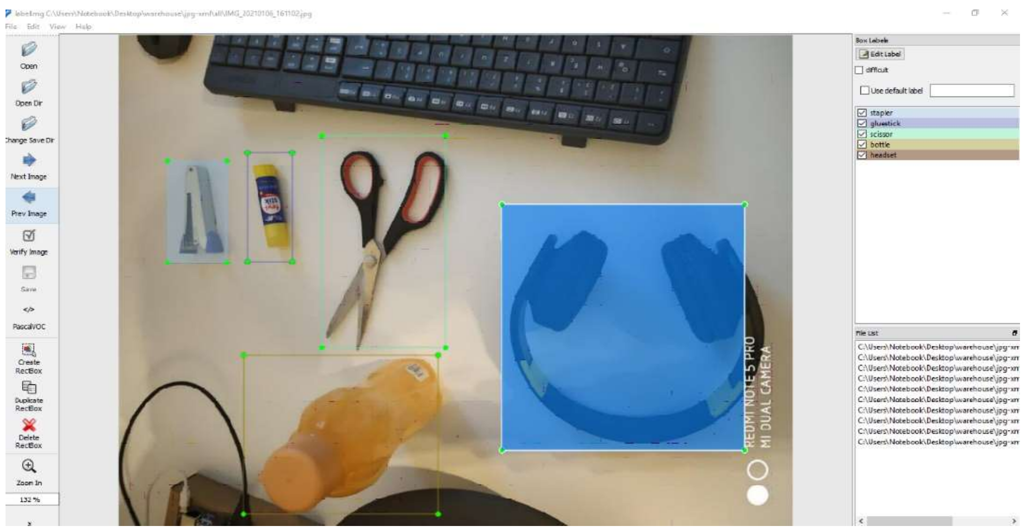


Figure 21: Annotation of own dataset

In figure 21, annotated objects using labelImg tool are displayed. Green rectangles are annotation of objects which are labelled by their classes. Once annotation is done, it is saved as xml files which gives information about the location and dimension of labelled objects.

After the completion of annotation, all generated .xml files from the training sample is converted to train_CSV file and the validation sample is converted to validation_CSV file, which could be further fed to the training of the model.

3.3 Modeling 1- Classification + drawing Manual Bounding box

Configurations:

Dataset: own created with 5 classes(gluestick, stapler, bottle, scissor and headset)

Labelling: labelImg

Training: 292 images, validation: 19 images

Deep Learning Framework: keras

Base Model: MobileNetv2

Real time Camera integration: OpenCV

GPU: Google Colab

Programming Language: Python

Classification:

Data Augmentation- Fliping(horizontal), Rotation is done to avoid overfitting. Pre-trained model Mobilenet v2 (Convolution part is freezed from training, only the new classification part is trained.). To prevent overfitting, dropout layer is added. Final dense layer with activation function 'softmax' is able to classify the 5 classes of customized dataset. For compiling the model, 'Adam' optimizer and loss function 'Categorical cross entropy' (since, more than two classes) are used. For training, learning rate: 0.0001, epochs=20 are used.

Drawing Bounding box:

Manually image prediction bounding box is drawn. Manually non-max suppression is done to suppress less confident bounding boxes.Results are not as expected, i.e, bounding box is not appropriate to the object and in addition, non-max suppression is not working properly.

3.4 Modeling 2

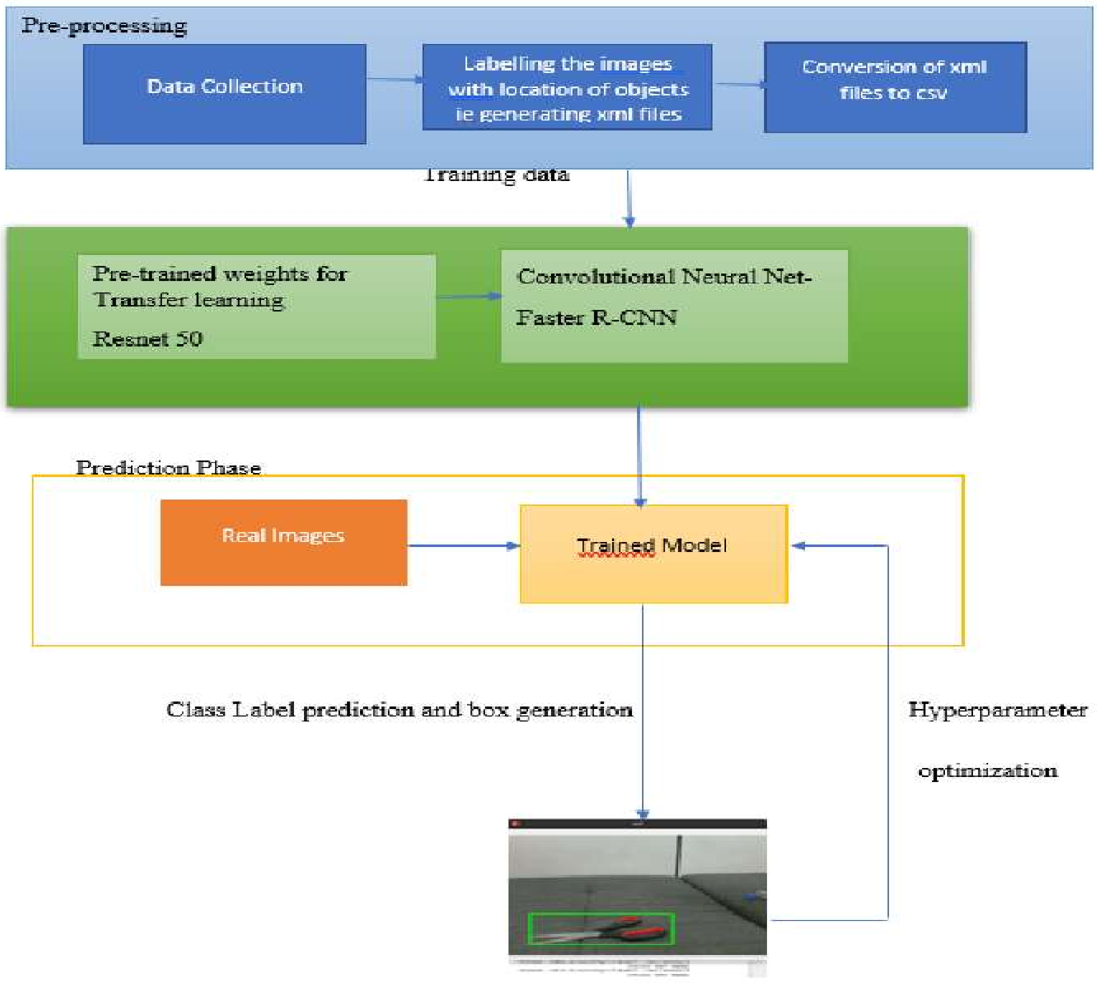

Pre-processing
Data Collection
Labelling the images with location of objects ie generating xml files
Conversion of xml files to csv
Training data
Pre-trained weights for Transfer learning
Resnet 50
Convolutional Neural Net-
Faster R-CNN
Prediction Phase
Real Images
Trained Model
Class Label prediction and box generation
Hyperparameter optimization

Figure 22: Implementation workflow

Data Pre-processing:

Pre-processing of input images are done with the help of transforms module present in Torchvision module. Resize the image to 256*256 pixels, crop the image to 226*226 pixels about the centre, image is converted to PyTorch Tensor data type and the normalization is done. Python Imaging Library (PIL) offers many procedures for image manipulation. So, the input images are transformed to PILImage. Later, some samples with bad brightness and images with noise were removed.

Framework:

Pre-trained models such as Resnet 50 are trained on large datasets like ImageNet. Instead of training from scratch, weights, and biases of pretrained Resnet50 model is used. This model takes input as image and predict its class.

Model Inference process includes reading the input image, performing transformations on image such as resize, normalization, then use the pre-trained weights to find out the output vector and later, predictions are displayed based on scores obtained.Various changes have been made to the default configuration Faster R-CNN provided by PyTorch API named detecto.

Configurations:

Dataset: own created

Classes: 5 (gluestick, stapler, bottle, scissor and headset)

Labelling: labelImg

Training: 292 images, validation: 19 images

Preprocessing: torchvision.transforms

Deep Learning Framework: PyTorch

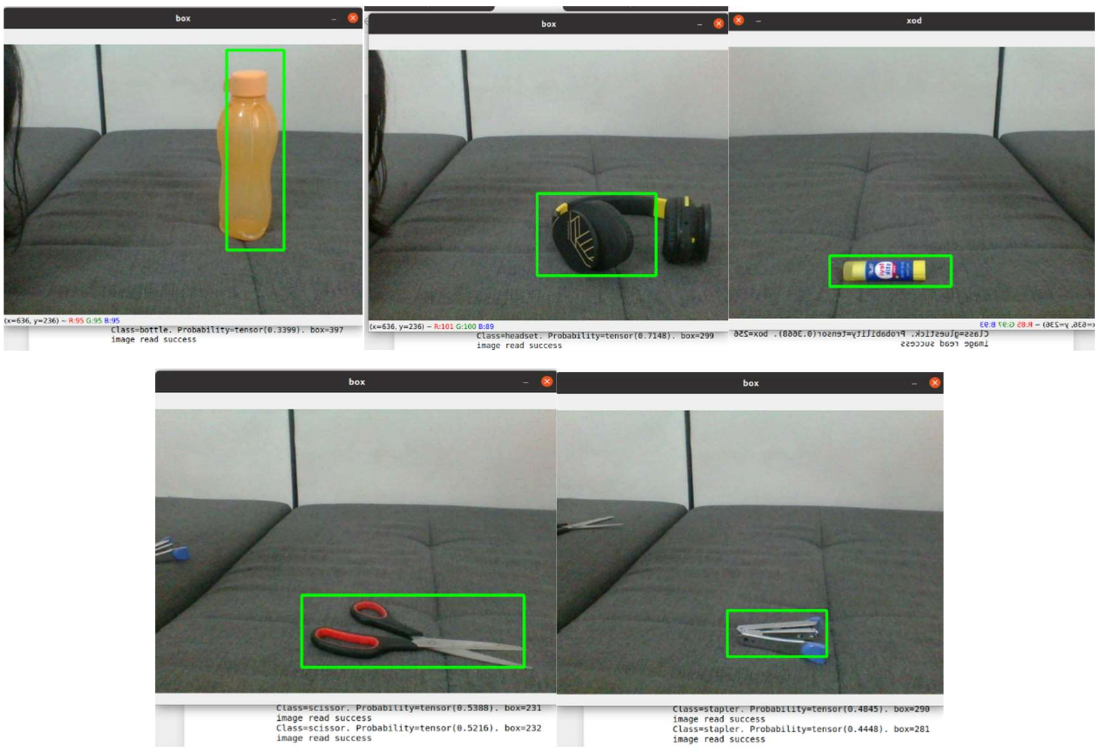
Object detection Model: Faster R-CNN model ResNet50(detecto)

Real time Camera integration: OpenCV

Programming Language: Python

The model is trained with training samples from objects such as glue stick, stapler, bottle, scissor and headset. Training samples are divided into training dataset and validation data set. The train dataset is used to extract the significant features and the validation dataset is used to evaluate the features and to calculate the losses.

Results from own customized models out-of-box:


box
box
xod
box
box
(x=636, y=236) ~ R:95 G:95 B:95
(x=636, y=236) ~ R:101 G:100 B:89

```
Class=bottle. Probability=tensor(0.3399). box=397
image read success
```

```
Class=headset. Probability=tensor(0.7148). box=299
image read success
```

```
Class=scissor. Probability=tensor(0.5388). box=231
image read success
Class=scissor. Probability=tensor(0.5216). box=232
image read success
```

```
Class=stapler. Probability=tensor(0.4845). box=290
image read success
Class=stapler. Probability=tensor(0.4448). box=281
image read success
```

Figure 23: Implementation of customized model

The above figures show results of object detections (out-of-box- real time) on own customized datasets. Class labels are predicted correctly as bottle, headset, glue stick, scissor and stapler. The confidence value of prediction boxes is quite good enough.

Results from own customized models inside-box:

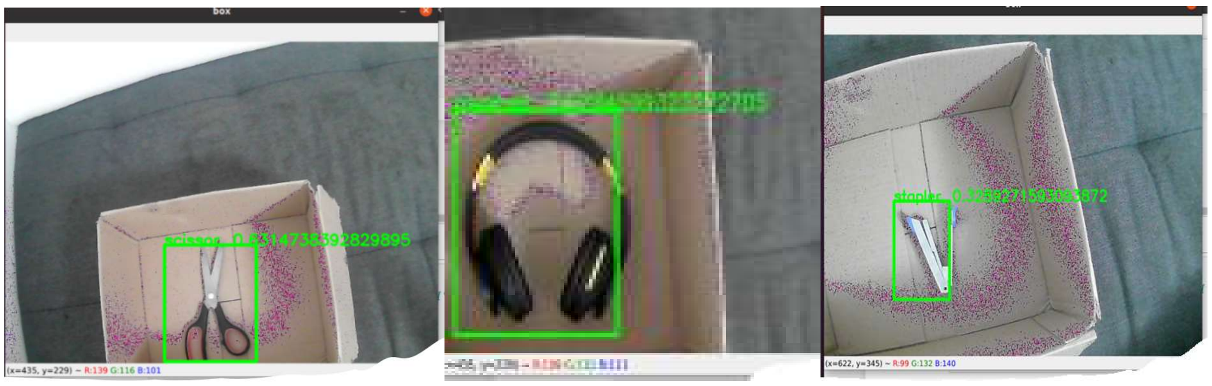
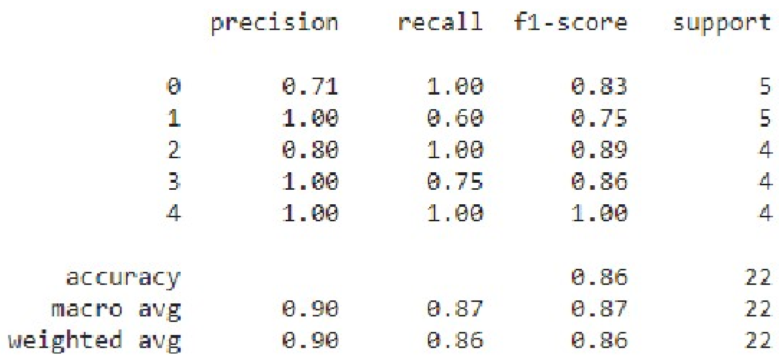

box
scissor 0.8314738392829895
stapler 0.3258271593093872
(x=435, y=229) ~ R:139 G:116 B:101
(x=622, y=345) ~ R:99 G:132 B:140

Figure 24: Implementation of customized model inside box

Finally, the detection of objects inside the box on real time is implemented successfully. Class labels are identified correctly along with bounding boxes. The confidence values of predictions are considerable.

3.5 Performance Analysis

Classification Report for customized dataset:



```
              precision    recall  f1-score   support

           0       0.71      1.00      0.83         5
           1       1.00      0.60      0.75         5
           2       0.80      1.00      0.89         4
           3       1.00      0.75      0.86         4
           4       1.00      1.00      1.00         4

    accuracy                           0.86        22
   macro avg       0.90      0.87      0.87        22
weighted avg       0.90      0.86      0.86        22
```

Figure 25: Classification Report on Model 1

Classification report is about the summary of main classification metrics. The percentage of correct classification performed on model 1 is 86%. Precisions (accuracy of positive predictions) are almost above 70% in all the classes. Recall (the fraction of correctly identified positives) are above 60% in general. F1 scores (mean of precision and recall) are above 0.75 which is pretty good. Support is the number of occurrences of given class in test set. We could notice that test set is imbalanced as the number of occurrences of each class in test set differs.

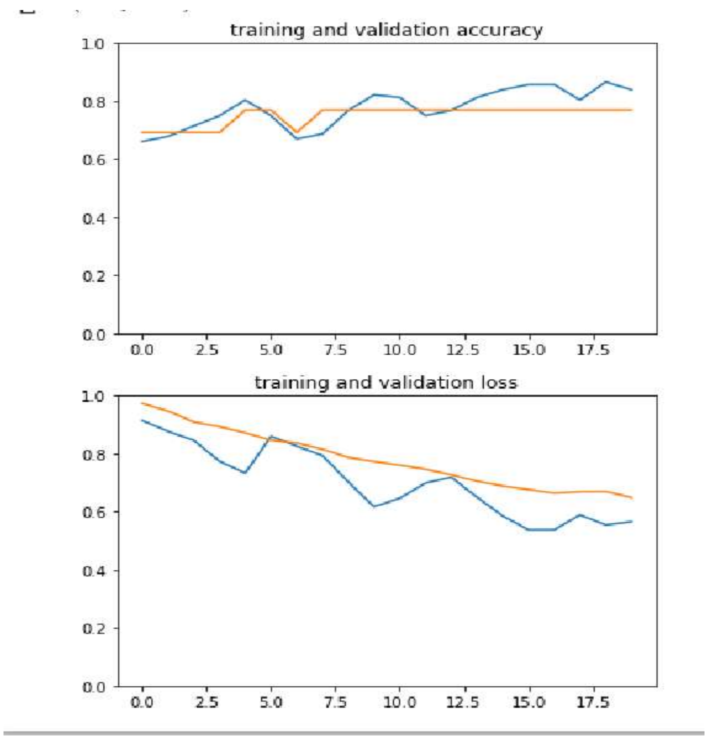
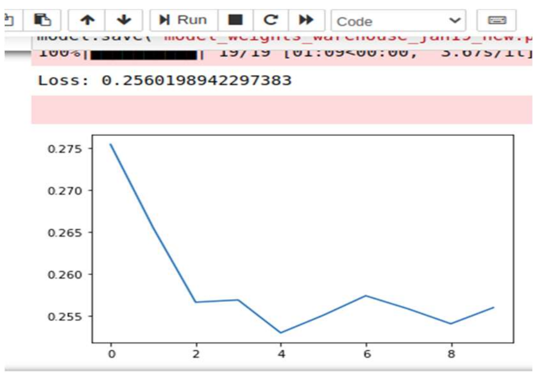


Figure 26: Accuracy and Loss curve on Model 1

First graph shows the curves of training and validation accuracy over 20 epochs. Here, training accuracy is gradually increasing on increasing the number of epochs. Test accuracy remained stable after epoch 7. The larger the gap between training and validation accuracy, the higher the overfitting. On seeing accuracy graph, we could notice little overfitting.

First graph shows the curves of training and validation loss over 20 epochs. Here, training loss keeps decreasing on the whole. The validation loss also gets lower for each epoch. Since model's error on validation set is higher than model's error on training set, it seems that model is getting overfitted.

Loss curve



```
model.save('model_weights_warehouse_jan19_new.p
100%|██████████| 19/19 [01:09<00:00,  3.67s/it]
Loss: 0.2560198942297383
```

Figure 27: Loss Analysis Model 2

The above graph shows loss curve against validation dataset over 10 epochs. On the analysis of above curve, the validation loss has reduced to considerable extent. Though decrease in validation loss is good, it seems that the model learns only on the increase of epochs. So, it seems to have overfitting.

4 Conclusion

Using Deep Convolutional Neural Networks, our use case is implemented, and results are evaluated. Transfer learning helps in adopting for new domains. Object detection using existing model is executed successfully. The results are pretty amazing(out-of-box). Later, Object detection with own customized model is performed. In modelling 1, the results of Classification are as expected but the results of manually drawing bounding box around the objects, are not as expected. In modelling 2, the results of object detection model using PyTorch Library are good (out-of-box). Finally, Object detection inside the box was performed, the model is able to adopt to new objects and the results are considerable.

Future Work:

The results look little promising provided with the extremely small set of training samples. However, there are possibilities to improve the model. It can be improved further with other software platforms such as TensorFlow which allows incomparable GPU computation and rapid prototyping. Model could be further improved by increasing the training samples and by optimizing the hyperparameters. Other object detectors such as SSD, YOLO could also be tried to improve the performance.

Declaration:

"I hereby declare that this seminar report is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references."

[illegible]

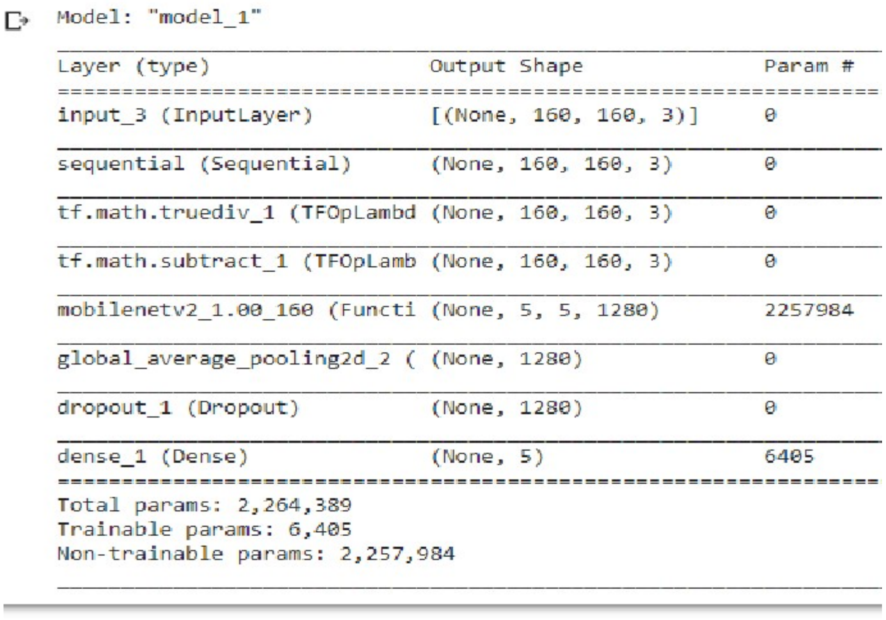
References

1. A Survey of Deep Learning-based Object Detection Licheng Jiao, Fellow, IEEE, Fan Zhang, Fang Liu, Senior Member, IEEE, Shuyuan Yang, Senior Member, IEEE, Lingling Li, Member, IEEE, Zhixi Feng, Member, IEEE, and Rong Qu, Senior Member, IEEE. arXiv:1907.09408v2 [cs.CV] 10 Oct 2019.
2. Investigations of Object Detection in Images/Videos Using Various Deep Learning Techniques and Embedded Platforms—A Comprehensive Review Chinthakindi Balaram Murthy 1,†, Mohammad Farukh Hashmi 1,† , Neeraj Dhanraj Bokde 2,† and Zong Woo Geem 3, 3280; doi:10.3390/app10093280, Appl. Sci. 10.2020.
3. Object Detection with Deep Learning: A Review Zhong-Qiu Zhao, Member, IEEE, Peng Zheng, Shou-tao Xu, and Xindong Wu, Fellow, IEEE. arXiv:1807.05511v2 [c.CV]2019.
4. https://ruder.io/transfer-learning/
5. https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview
6. https://www.pyimagesearch.com/2018/05/14/a-gentle-guide-to-deep-learning-object-detection/
7. https://github.com/jeffheaton/t81_558_deep_learning
8. https://towardsdatascience.com/transfer-learning-using-differential-learning-rates-638455797f00
9. https://www.youtube.com/watch?v=8czpAtZZ8qo&ab_channel=MajaaMatrix

10. https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852
11. https://de.mathworks.com/solutions/image-video-processing/object-recognition.html
12. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
13. en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png
14. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
15. https://www.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html
16. https://www.researchgate.net/figure/An-overview-of-the-ten-pre-trained-networks-architecture-used-in-this-study-In-each_fig1_341042851
17. https://www.datacamp.com/community/tutorials/object-detection-guide
18. https://livebook.manning.com/book/grokking-deep-learning-for-computer-vision/chapter-7/52
19. https://www.learnopencv.com/selective-search-for-object-detection-cpp-python/
20. missinglink.ai/guides/neural-network-concepts/instance-segmentation-deep-learning/
21. https://docs.microsoft.com/en-us/cognitive-toolkit/object-detection-using-fast-r-cnn-brainscript
22. https://machinelearningmastery.com/object-recognition-with-deep-learning/
23. https://www.researchgate.net/figure/An-overview-of-the-ten-pre-trained-networks-architecture-used-in-this-study-In-each_fig1_341042851
24. https://github.com/yehengchen/Object-Detection-and-Tracking/blob/master/img/yolo_vs_rcnn.png
25. https://analyticsindiamag.com/how-the-deep-learning-approach-for-object-detection-evolved-over-the-years/
26. https://www.dlology.com/blog/recent-advances-in-deep-learning-for-object-detection/
27. datascience.stackexchange.com/questions/77830/how-do-stacked-cnn-layers-work
28. https://www.researchgate.net/figure/Milestones-of-object-detection-In-2012-the-major-turning-point-was-the-use-of-DCNN_fig1_341099304
29. https://www.youtube.com/watch?v=K0lWSB2QoIQ&ab_channel=PythonEngineer
30. https://medium.com/@sonish.sivarajkumar/image-classification-and-object-detection-c9803f854923
31. https://detecto.readthedocs.io/en/latest/api/visualize.html

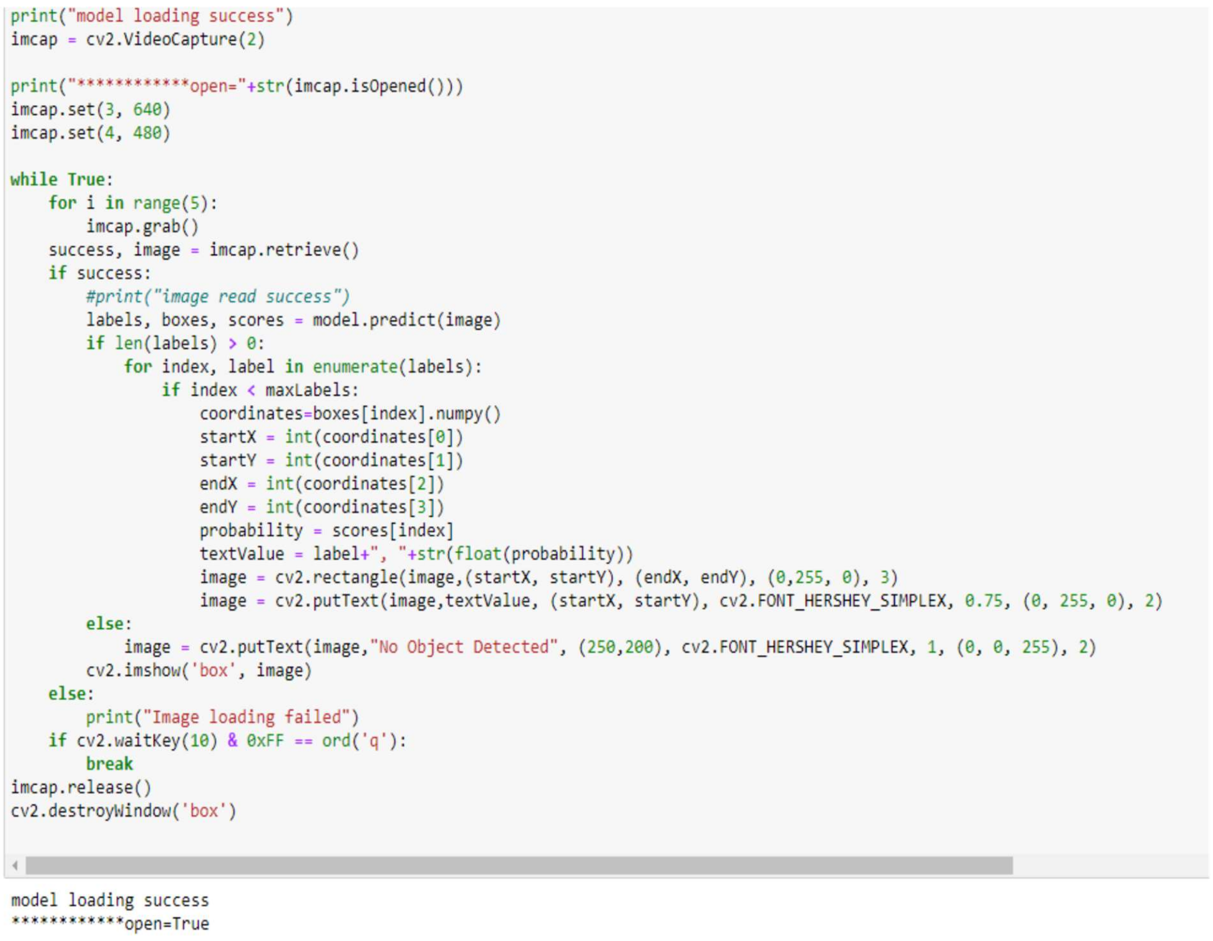
Appendix

Model 1 Summary



```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 160, 160, 3)]     0
_________________________________________________________________
sequential (Sequential)      (None, 160, 160, 3)       0
_________________________________________________________________
tf.math.truediv_1 (TFOpLambd (None, 160, 160, 3)       0
_________________________________________________________________
tf.math.subtract_1 (TFOpLamb (None, 160, 160, 3)       0
_________________________________________________________________
mobilenetv2_1.00_160 (Functi (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d_2 ( (None, 1280)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1280)              0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 6405
=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
_________________________________________________________________
```

Integrating- camera:



```
print("model loading success")
imcap = cv2.VideoCapture(2)

print("************open="+str(imcap.isOpened()))
imcap.set(3, 640)
imcap.set(4, 480)

while True:
    for i in range(5):
        imcap.grab()
    success, image = imcap.retrieve()
    if success:
        #print("image read success")
        labels, boxes, scores = model.predict(image)
        if len(labels) > 0:
            for index, label in enumerate(labels):
                if index < maxLabels:
                    coordinates=boxes[index].numpy()
                    startX = int(coordinates[0])
                    startY = int(coordinates[1])
                    endX = int(coordinates[2])
                    endY = int(coordinates[3])
                    probability = scores[index]
                    textValue = label+", "+str(float(probability))
                    image = cv2.rectangle(image,(startX, startY), (endX, endY), (0,255, 0), 3)
                    image = cv2.putText(image,textValue, (startX, startY), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)
        else:
            image = cv2.putText(image,"No Object Detected", (250,200), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        cv2.imshow('box', image)
    else:
        print("Image loading failed")
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break
imcap.release()
cv2.destroyWindow('box')
```

```
model loading success
************open=True
```

Additional figures:

Two stage Detectors[26]

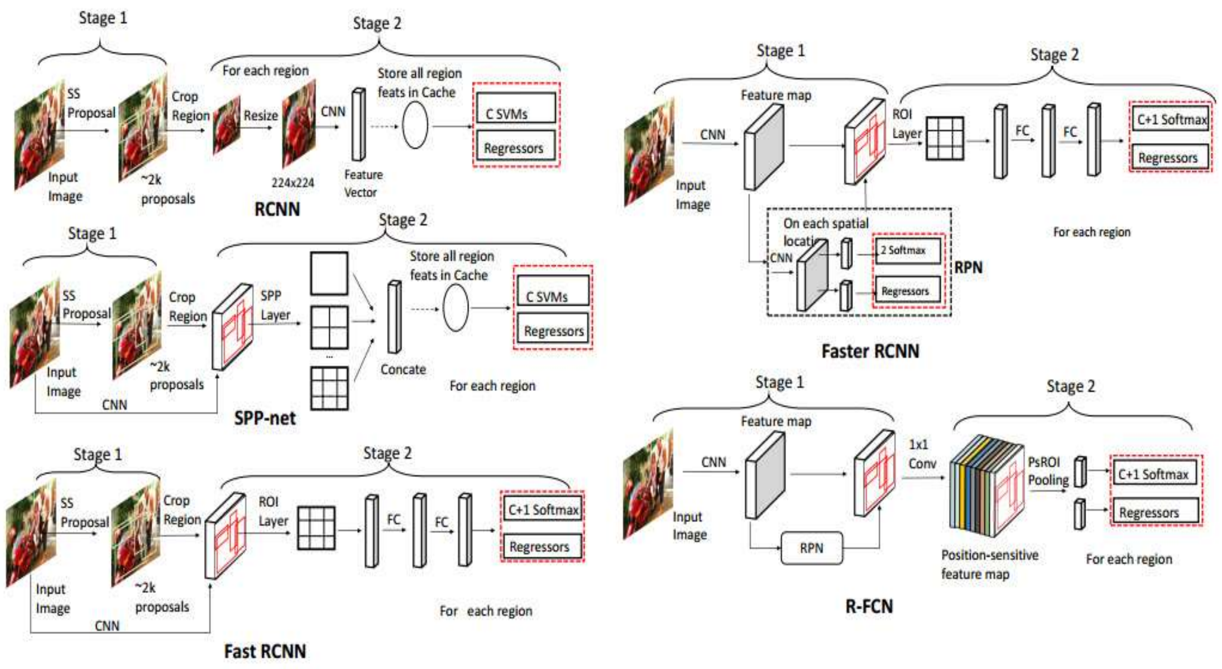


Figure 4: Overview of different two-stage detection frameworks for generic object detection. Red dotted rectangles denote the outputs that define the loss functions.

One-stage Detector[26]

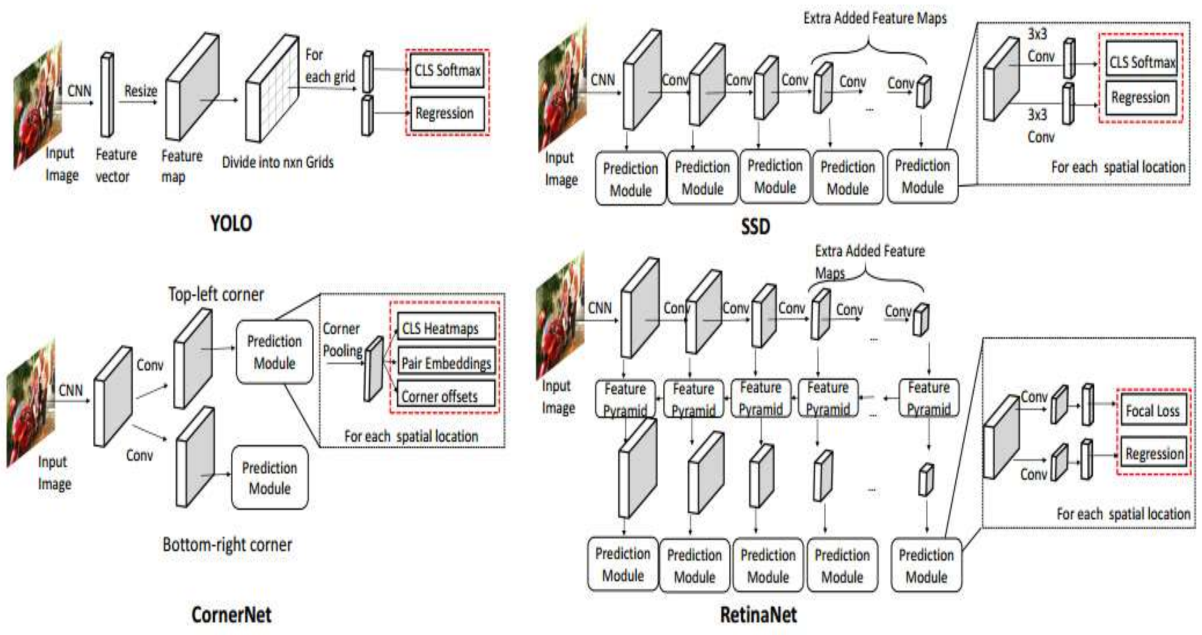


Figure 5: Overview of different one-stage detection frameworks for generic object detection. Red rectangles denotes the outputs that define the objective functions.

Backbone Architecture

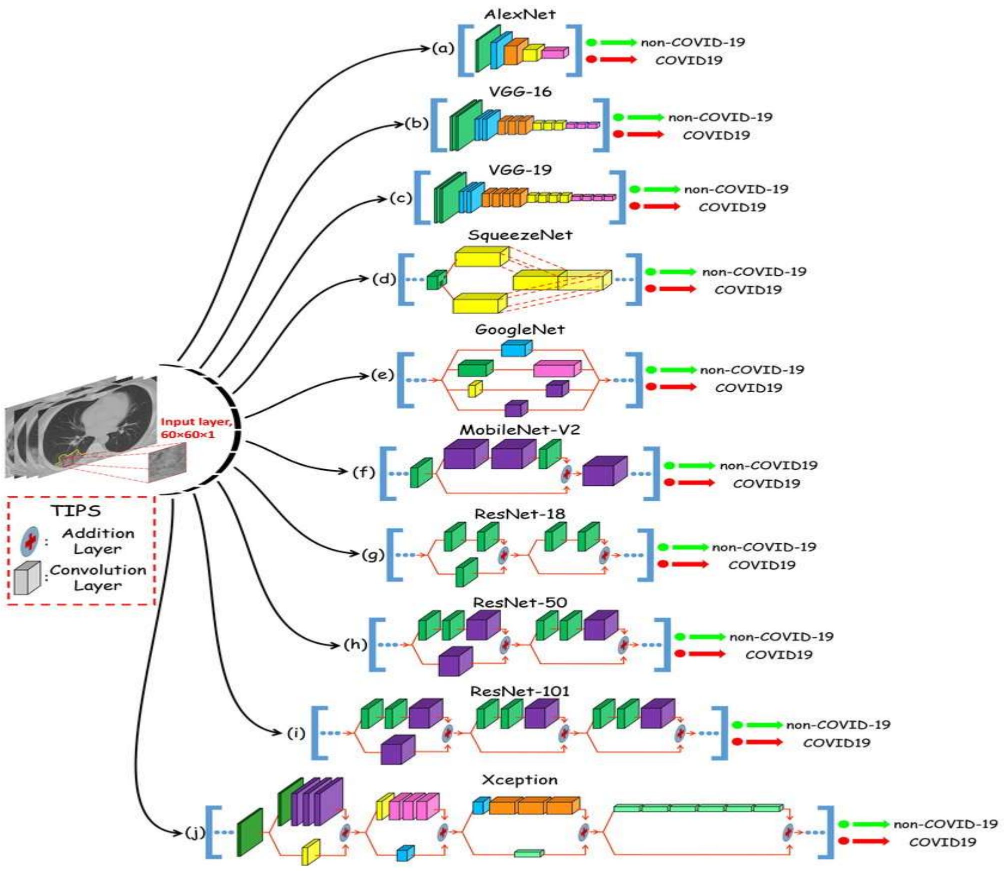


Figure 28: Overview on existing open source Pre-trained models[23]

RCNN:

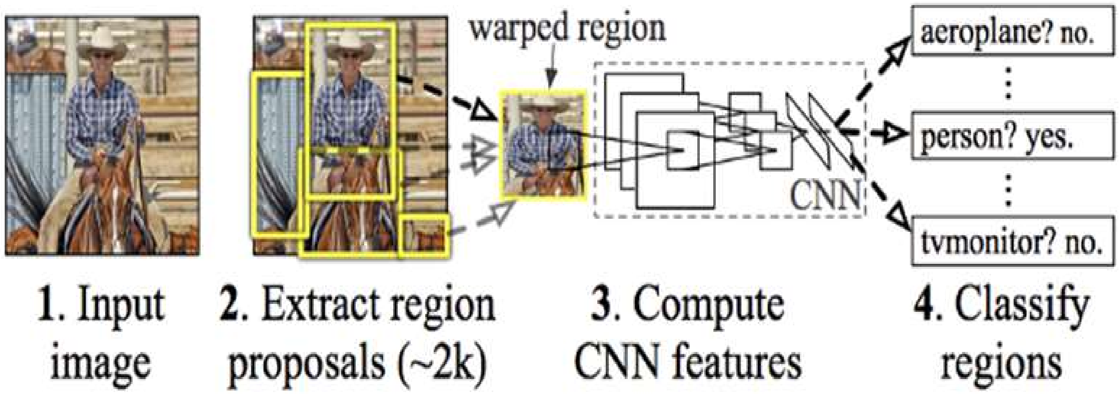


Figure 29: RCNN model [25]

Existing Model Implementation:

```
from detecto import core, utils, visualize
from detecto import core, utils
from torchvision import transforms
import matplotlib.pyplot as plt
image = utils.read_image('fruit.jpeg')
model = core.Model()
labels, boxes, scores = model.predict_top(image)
visualize.show_labeled_image(image, boxes, labels)
```

Model 1:

Data Augmentation:

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
  tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
])
```

Preprocessing in keras:

```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
```

Model 1:

```
        base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                include_top=False, weights='imagenet')
        inputs = tf.keras.Input(shape=(160, 160, 3))
        x = data_augmentation(inputs)
        x = preprocess_input(x)
        x = base_model(x, training=False)
        x = global_average_layer(x)
        x = tf.keras.layers.Dropout(0.2)(x)
        outputs = prediction_layer(x)
        model = tf.keras.Model(inputs, outputs)
```