



FAKULTÄT FÜR
INFORMATIK

# Master Thesis

# Iconography in Christian Historic Artwork

Author:
Syed Abdullah Rizvi

Magdeburg, 31.03.2022

Supervisor
M.Sc. Erasmo Purificato

Professor
Dr.-Ing. Ernesto William De Luca

Digital Transformation and Digital Humanities Group
Department of Computer Science
Otto-von-Guericke University
Magdeburg, Germany

# Abstract

Interpreting the underlying semantic information in the art pieces with their context is very essential for understanding the hidden meaning in them. However, even for professionals, analyzing the scene is challenging due to the incredibly intricate and sophisticated portrayal of these artworks. Recently, many researches have established the viability of using AI methods for these purposes. In the light of this, this work uses Object Detection models of YOLOv5 and Faster R-CNN used in Computer Vision domain of Machine Learning to make an attempt on iconography of paintings of Holy Mary. The results showed that the models were able to predict Holy Mary; however, an explainable prediction of the results using CAM was not completely achieved.

# Acknowledgment

Prima facie, I am thankful to God for providing me with the health and well-being required to finish this thesis.

Naturally, I owe gratitude to my research supervisor, Mr. Erasmo Purificato, from the Digital Transformation and Digital Humanities (DTDH) Group from the Faculty of Informatik (FIN) at the OVG Universität. Without his guidance and unwavering commitment to every step of the way, my thesis would never have been completed. I would want to express my heartfelt gratitude for your support and understanding throughout this time.

I would also want to express my thanks to Professor Dr.-Ing. Ernesto William De Luca of the same DTDH Group at OVG Universität, for providing me with the chance to work on and for overseeing this thesis.

I also place on record, my sense of gratitude to everyone, who directly or indirectly, have lent their hand in this work.

Most significantly, none of this would have been possible without the support of my family. To my loving parents and supportive sisters, as well as to my beautiful and understanding companion. I will forever be grateful. This thesis stands as a testament to your unconditional love and encouragement [1]

# Contents

# Section 1: Introduction

The word “iconography” comes from two Greek words, *eikon* (meaning “image”) and *graphe* (meaning “writing”). Together we get “image-writing,” so the word “iconography” In this case, expresses the concept that a picture can tell a story. It is actually more difficult to grasp an image's iconography than to understand its subject matter, since it entails knowing the unique culturally produced symbols and motifs present in a piece of art that might assist us in identifying its subject matter. It is necessary to be knowledgeable of the symbols' culturally unique meaning in order to fully understand them when they are depicted in an artwork.

## Section 1.1: Motivation

A research into machine learning applications in iconography led to the conclusion that iconographical analysis of Christian artworks have used image classification techniques on full images of Christian saints. The motivation then arise for this work was to make this task an object detection problem by making use of a current state-of-the-art models (SOTA) that would train on key objects that appear regularly in pictures of Christian artworks and doing inference on them. An explainable AI technique like CAM was then to be used to see the relevancy of the results.

## Section 1.2: Aim of this Thesis

Goal of thesis was to do iconographic analysis on Christian saint Holy Mary using object detection models from Computer Vision techniques of Machine Learning. The expectation was to make a model that could detect if the picture is of Holy Mary or not based on certain key regions like her clothes.

## Section 1.3: Thesis Outline

After introducing the topic of this thesis in Section 1, Section 2 focuses on Related Work and researches done in this field. Afterwards, Section 3 is discussing Methods used, meaning models that this thesis used, their architectures and performances. Section 4 then writes about Implementations to the training experiments as well as their Results. Section 5, then concludes this thesis.

# Section 2: Related Works

There are numerous ways to connect an artwork to its meaning. The eras and audiences for whom it was designed may influence the artwork. Therefore, Iconography can tell the variety of objects that comprise the represented picture, their personality and mutual connections, and inevitably their possible symbolic definition. It can also tell about artists influences on their arts. Therefore, iconography plays an essential part in historical paintings. For this, the iconographic argument constantly relies on accumulating historical data in relation to a particular artwork.

The use of images to transmit religious beliefs and ideas, as well as to illustrate religious events, is known as religious iconography. Icons here can be little objects or part of regions in a religious artwork that expresses specific or symbolic significance.

Some Examples of Religious Iconography [2], [3] are:

1. The Virgin Mary is typically shown in a flowing blue garment/robe. Heaven and her spirituality is shown through this. In addition, her purity is symbolized by Lilies in a vase with a bowl or urn of water.
2. The cross has been a religious symbol for Christianity since the second century. Crucifix which is a cross with Jesus on it, symbolizes Catholicism.
3. The Holy Spirit is frequently depicted as a dove. This is based on the account of Christ's baptism, when the Holy Spirit descended like a dove from heaven. It may also be used to signify a person's soul.
4. The fish emblem, which originates from the Greek term ichthus, is frequently used to symbolize Jesus Christ. The image of a lamb is occasionally used to represent Jesus and/or his love. This has to do with the fact that he is referred to as God's lamb.
5. Religious figures such as gods, saints, prophets, and martyrs are sometimes memorialized in sculptures, which serve as icons for individuals who practice the faith with which they are linked.
6. In religious iconography, divine creatures are frequently shown as human beings, but demons or evil spirits are depicted as threatening animals.

7. In religious iconography, images of books and tablets are used to represent God's message.
8. In religious art, a halo, which is a circle of light encircling a figure or their head, is sometimes used to symbolize a holy person or saint. Flames, known as Mandorla in Asian religious art, are wrapped around the torso or head.
9. Many churches feature stained glass windows that communicate stories about their beliefs through visual images. The Stations of the Cross are frequently shown in these windows in Catholic churches.

With recent advent of Machine Learning and Computer Vision tools in particular, it is now possible to recognize objects in pictures and videos. However, in the past, historical artwork domain remained one of the field in which difficulties arise due to the availability of lesser data. This problem gets more aggravated due to presence of even lesser annotated datasets. Researchers in recent years now have at their disposition a wide range of pictures to evaluate the influences throughout artworks and authors. The most noticeable initiative here is the **Iconclass system** [4]**,** which gives over 28k category types for ten top-level classifications of pictures and it is significantly used by art historians and collection managers. Another, widely used dataset is **IconArt** [5] and most recently, **ArtDL** [6] project has collected a substantial amount of data from various sources for researchers to apply Computer Vision techniques. They have also provided a neural network classifier trained on ImageNet and have use it for iconography of the saints present in ArtDL dataset.

In recent years, many researchers like E. J. Crowley and A. Zisserman [7] , have applied Transfer Learning for implementing deep learning techniques on artworks such as paintings.
Others like [8] have use time context to improve object detection in paintings for Iconography purposes. Some have detected people in artworks [9] and genre and style of artwork [10]. Gonthier et al. [11] used residual networks [12] on the image of Saint Sebastian and the Jesus. A similar concept of iconography using CNNs was also demonstrated by Madhu P. et.al in [13]. However, Federico Milani and Piero Fraternali [6] did one of the most tremendous work in this field. Not only had their dataset ArtDL, as mentioned in previous section, one of the best for Iconographic studies, but their comprehensive research also categorizes many of the research papers written with regard to this subject. Iconography in specific saints from Christian historic artwork was also done by [14] where they train a Convolutional Neural Network (CNN) on bodies of Mary and Gabriel. Another

work on iconographic analysis using Milani et.al dataset and with a different approach was done by Madsen n.d. [15]. Together with the [6] and [13] these two papers forms the basis of this thesis. Moreover, Class Activation Mapping [16] that suggests specific regions of image, which the model is predicting upon, was also attempted.

# Section 3: Methods

This section will give a theoretical background on the models used in this thesis. First, a brief background of object detection algorithms will be presented, Afterwards, working and architecture of Region based Convolutional Networks (RCNN) [17] family of object detectors will be discussed. Then, the Yolo Only Look Once (YOLO) [18] family of object detectors will be discussed. Finally, the method for explaining model results, Class Activation Mapping (CAM) will be presented.

## Section 3.1: Background

Object detection algorithms are extremely powerful and may be applied across a wide range of applications. Now, it is a prominent topic in computer vision research, and it has been improving for years with the inclusion of new boosted techniques that enhance precision and speed. An image consists of a combination of objects and location. The process of detection is the identification of the object, whereas classification is the categorizing of the object based on previously determined classes. Object detection combines these two tasks [19].

With the advent of CNNs, a number of highly efficient architectures have been made in the domain of computer vision. Earliest architectures like DPM (Deformable Parts Model) were using a sliding window on the images that were input to them. Few years back, Region based CNN was developed and was an instant hit. They work by extracting probable regions. R-CNN employs an area proposition network. A classifier design is then used to categorize the extracted characteristics. Some issues in R-CNN's led to the creation of advanced R-CNN extensions known as Fast R-CNN and Faster R-CNN. Although they were an advancement on the previous mode, however still they were difficult and time-consuming to master since many aspects had to be learnt independently. The invention of YOLO (You Only Look Once) [18] corrected these flaws. In contrast to prior designs of object detection algorithms, YOLO merely looks at the image to comprehend where the item is as well as what class it belongs to. Because of its simple manner of detection, YOLO is extremely rapid; yet, it is less accurate than R-CNNs. The high frame rate of YOLO allows for live detection, with a base version operating at 45 frames per second and a somewhat faster version running at

roughly 150 frames per second, respectively. The YOLO architecture tackles the challenge of object identification using regression analysis. Different versions of YOLO have been developed, with the first being YOLOv1 and the latest being YOLOv5.

## Section 3.2: Deformable Parts Model

The deformable components model separated a picture into a number of sections using a typical sliding window approach, which was then use as an input directly into a classifier [15]. The whole method is detailed in the activities mentioned below.

1. To begin, a picture containing objects is given.
2. The image is then separated into equall regions as shown in figure 1.
3. Afterwards, each section is then treated as a distinct full picture.
4. A convolutional neural net is then trained on all of the individual photos to classify them.
5. Once all areas of objects have been assigned a class, whole picture with those objects is then set up using the divided images.

While this strategy is easy, the model's learning process is complicated by the different positions of objects in the photos and the aspect ratios of objects. It is possible that certain objects can/will take up a large portion of the image while others will take up a little portion. Additionally, the activity requires a variety of objects of varying sizes and shapes. This would need the image to be segmented into a large number of sections, this would can make the process of learning and detection exceedingly sluggish and computationally intensive [20].

**Fig. 1:** Image divided into many sub-images as a result of a sliding window [21]

## Section 3.3: R-CNN (Region based CNN)

Ross Girshick and colleagues from the University of California, Berkeley released a paper titled "Rich feature hierarchies for accurate object detection and semantic segmentation" in 2014[21] Ross introduced a novel technique for object identification in this study, which was developed to overcome the limits of prior sliding window-based versions. This technique was dubbed R-CNN (Region-based CNN) due to the fact that it combined convolutional networks with a region proposal component. This method resulted in the first large-scale and practical implementations of deep learning-based object identification [22]. The model is divided into three major modules:

1. Region Proposal: A method known as Selective Search is used to exclude 2000 regions. Additionally, these are called bounding boxes.
2. Function Extractor: Those regions are then sent through a convolutional network to remove any remaining features.
3. Classifier: They are categorized into one of the classes using linear SVM[22]

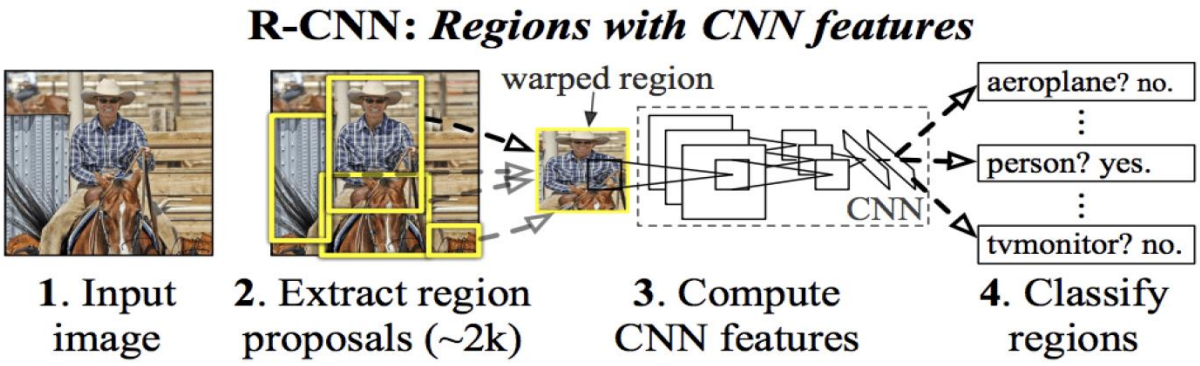



**Fig. 2:** R-CNN model architecture [22]

The complete procedure with R-CNN for detecting objects may be summarized as follows:

1. As was the case with the preceding algorithms, a picture is used as the input.
2. Selective Search approach is used for extracting interesting regions from that image. It focuses on the four components: textures, colours, scales, and enclosure. Selective Search makes note of trends in a picture and suggests areas of interest based on these. In all, the proposal network identifies 2000 regions of potential relevance.
3. The area of interest is then warped and reshaped in accordance with the input of the convolution network.
4. The subsequent phase deploys a convolutional network, AlexNet Deep CNN, and then a layer which is connected fully. CNN extracts features from potential regions of interest and then generates an element vector with a size of 4096.
5. This vector is then classified by a SVM based on the objects according to their exclusive regions of interest.
6. The same output vector is then used to construct bounding boxes for the regions of interest through linear regression. [21]

This is a high-level summary of the CNN object identification technique using regions. Despite its effectiveness, the total detection procedure is incredibly time consuming due to the architecture's three models. The primary disadvantage of this architecture is the extraction of

2000 areas of interest, as it takes a significant amount of time to educate a convolutional network on 2000 regions and also to detect the items included inside them. As a result, the model may be unsuitable for use in real-world applications. [21]

## Section 3.4: Fast R-CNN (Fast Region based CNN)

Fast R-CNN (Fast Region based CNN) were developed by same authors to minimise the limitations of the previous version [23]. This technique is identical to R-CNN, however it contains the first component is a pre-trained deep convolutional network instead of region proposal. This way the process becomes fast. The input picture is passed into a pre-trained CNN, and the generated maps are then used to identify regions of interest. This approach introduces in fact a pooling layer that is used to restructure the areas of interest captured by the fully connected layer [21] [23]

The following steps outline the complete training process:

1. The picture is sent into a pre-trained convolutional neural network to obtain the whole image's feature maps.
2. Using the same Selective Search approach as in R-CNN, the recovered feature maps are then employed as regions of interest.
3. Each extracted region is then sent through a Region of Interest pooling layer to reshape it to the precise size required for fully connected layer input.
4. Then three fully connected layers are used for object detection. The second one is a Softmax to identify objects and another with a linear regressor to detect the coordinates of the bounding box. [21], [23]

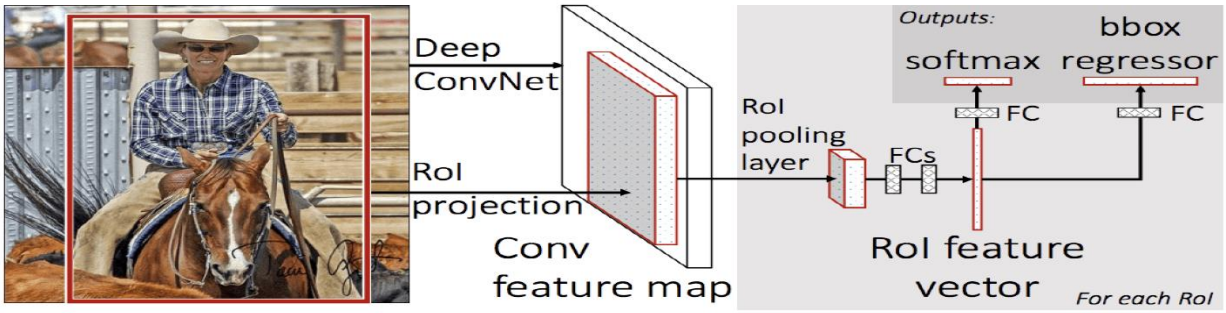



**Fig. 3:** Fast R-CNN model architecture [21], [23]

This is how Fast Region-based CNNs identify objects. Fast R-CNN is quicker in terms of training and detection since the CNN is trained once on the entire picture rather than on 2000 areas. However, even with being faster than normal R-CNNs, this Fast R-CNNs still was significantly slow, since it relied on Selective Search, and it is a time-consuming and slow-moving approach. Fast R-CNN reduces detection time to roughly 2 seconds, which is a significant improvement over R-CNN [23].

## Section 3.4: Faster R-CNN (Faster Region based CNN)

In 2016, Shaoqing Ren et al. published a study titled "Faster R-CNN: Towards Real-Time Object Detection using Region Proposal Networks." [24] This newer version of R-made adjustments to address the slowness of prior ones. Except for the region proposal network, the model architecture is quite similar to that of Fast R-CNN. Earlier versions relied on Selective Search for the proposal network, whereas the Faster R-CNN utilized Neural Networks for the area proposition network.

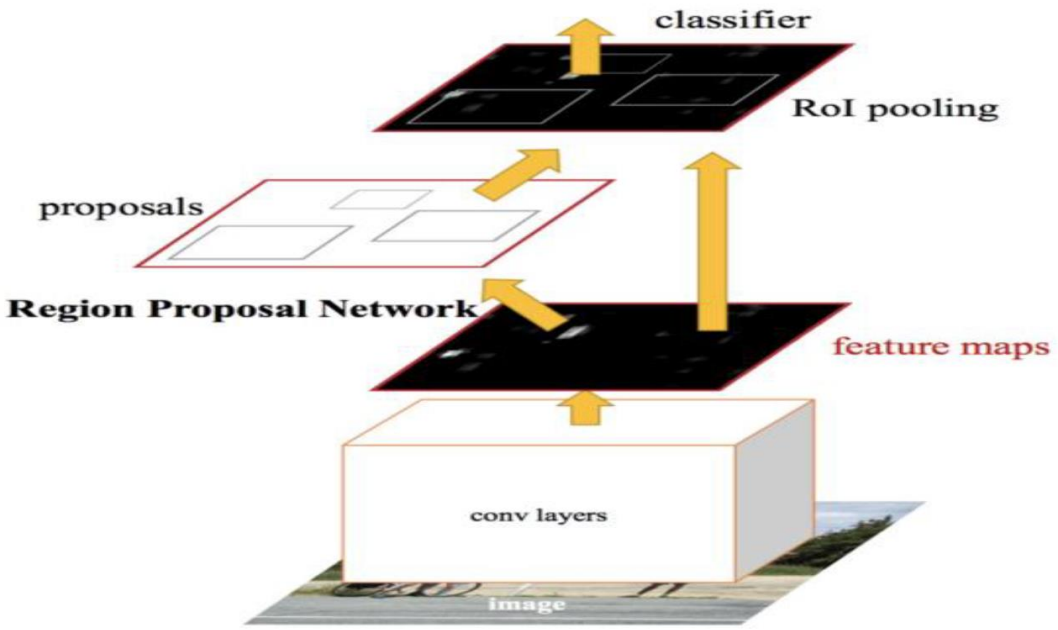



**Fig. 4:** Faster R-CNN design [24]

The substitution of a region proposal network for a Selective Search strategy resulted in a significant reduction in training time and also in training efficiency. The network design as a whole is composed of two primary components: the region proposal network (RPN) and the Fast R-CNN. [21]

The following actions outline the training of Faster R-CNN:

1. As with Fast R-CNN, an image is fed into a convolutional neural network to extract the feature maps
2. Afterwards, RPN generates regions based on the object's probability score and bounding box coordinates. The region proposal network selects an anchor point by the use of a sliding window on the image. Each window has k anchor points of varying shapes and sizes. The number of anchor boxes used is determined on the size and aspect ratio of the photos. The region proposal network is a convolutional neural network with a kernel diameter of 3x3, followed by two 1x1 convolutional layers for predicting the object's score and also the bounding boxes.[25]
3. As seen in Figure 4, region suggestions are then passed via a ROI pooling layer to equalize the size of all proposals for the fully connected layer input.
4. Following that, a Softmax layer predicts the object's class. Afterwards, a fully connected layer uses linear regression to predict the bounding boxes.

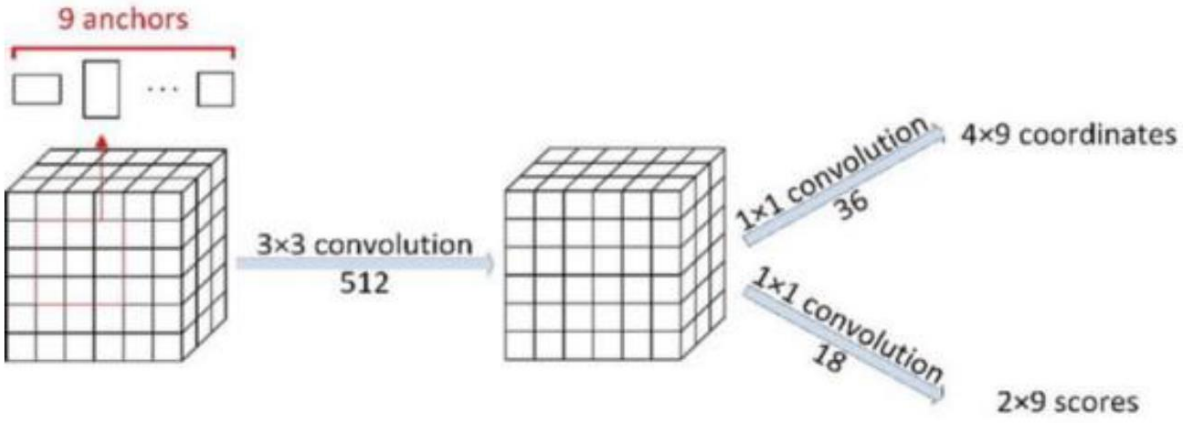



**Fig. 5:** Region Proposal Network of Faster R-CNN [25]

RPN was successful in lowering the training time associated with the Selective Search technique. In addition, the feature of RPN to interface with any type of item identification makes it a particularly valuable network. The faster R-CNN approach was likewise a region-based detection technique, scanning for objects in various areas of the picture one by one. The overall model was designed in such a way that multiple modules were ran one by one, implying that the performance of the following module can change due to the output of the preceding module. These problems were covered in Faster R-CNN. It is one of the most advance and fastest algorithm in all of the R-CNN versions. In terms of object detection speed and accuracy, the algorithm achieved on the best performance in current times [21], [24].

## Section 3.5: YOLO (You Only Look Once)

The methodologies described above made the task of object detection in two stages. To resolve the problem, the first step was to identify the areas of interest in the provided picture, and the second step was to categorize those regions of interest into the appropriate classes. Despite the improvements, this whole procedure was still too sluggish for real-time object identification applications. Because of this constraint, Joseph Redmon and colleagues developed a unified one-stage approach in 2016, which they named YOLO (You Only Look Once) [18]. YOLO means that you only look at the image once, rather than looking at several portions of the image to find the item. When a picture is taken in its entirety, the YOLO model uses a single neural network to predict item classes as well as bounding boxes in real time. In contrast to earlier algorithms, YOLO approached the identification of objects as a regression challenge, rather than as a classification problem that required the employment of a classifier. As a result, using this approach, the model could be trained incredibly quickly, and real-time pictures could be processed at 45 frames per second with the basic design [18].

YOLO was made in Darknet, which is a framework for deep learning developed in the C and Cuda programming languages. Even with the fact that it is very quick, YOLO has a much superior mean area under the curve (mAP) than various object identification methods, including Deformable Parts Model and also R-CNN. When compared to other cutting-edge algorithms, YOLO produces more

bounding box inaccuracies, although it is very uncommon to identify any objects when there are none. It also has excellent generalization capabilities [18].

YOLO's training technique is uncomplicated. To begin, the characteristics of a picture are utilized as input to the model, which is then refined. Following that, the model divides the whole picture into a S x S grid, where S may be any value between 1 and 100. Detecting the item whose centre falls inside each grid cell in the picture is the responsibility of each grid cell in the image. A vector including is predicted for each grid cell in the grid. This vector contains confidence score, bounding box, and class likelihoods for each grid cell.

Five values are assigned to each bounding box: the confidence, the elevation, the width, and the height. The confidence score indicates how certain the model is that a specific bounding box contains one or more items. The overlap between predicted and ground truth bounding boxes, is used to calculate the confidence score and this statistic is called the IOU (Intersection over union). When the training starts, grid cells containing object centers receive confidence scores of 1, with the rest of the values in the vector remaining unchanged. Other grid cells with no objects in them, as well as a vector of zeros, have a confidence score of zero. The value of C will be 10 for a dataset that has 10 classes as objects. There is can only be one class in the grid cell vector, regardless of how many bounding boxes are selected [18].

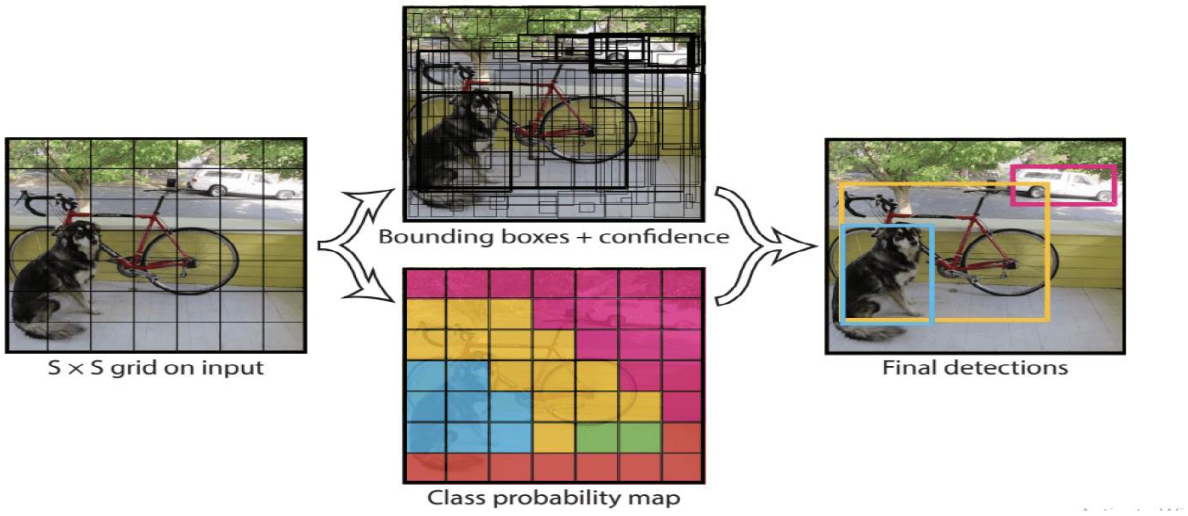


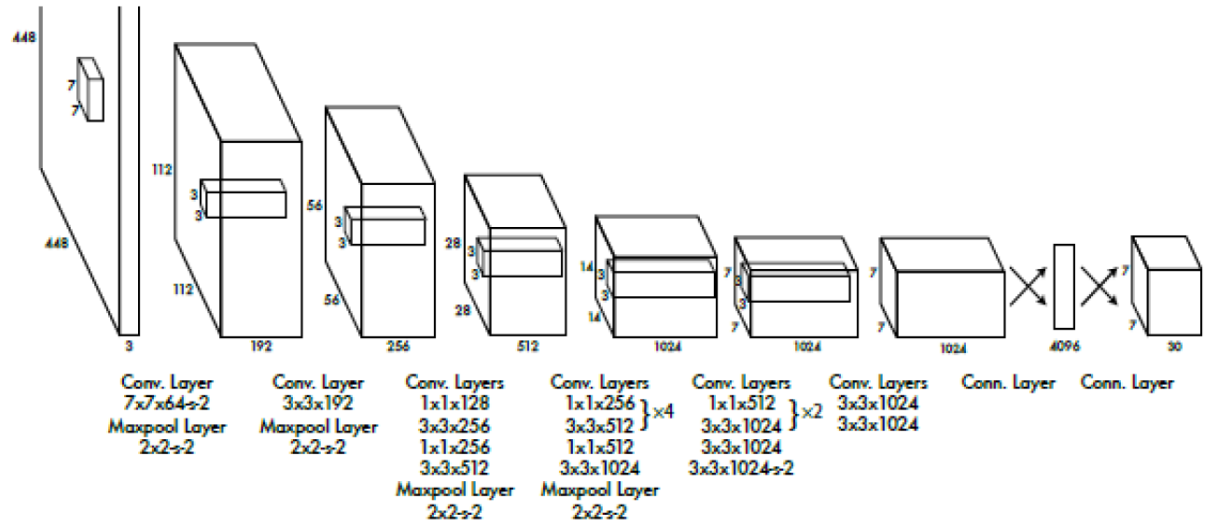

**Fig. 6:** YOLO model prediction [18]

## Section 3.5.1: Design

Figure 7 depicts the model design described in the original paper presented, which has a total of 26 layers in its entirety. The design has twenty-four convolution layers in start which are utilized for extraction, while two fully connected layers are used to forecasting are located in last for the output vectors. Apart from the fact that 1 x 1 layers followed by 3 x 3 convolution layers are used in lieu of inception modules, the architecture is quite similar to that of GoogLeNet. The smaller-sized form of YOLO contained nine convolution layers, as opposed to the basic model's 24 layers, with the rest of the parameters being the same as they were before. On ImageNet dataset with 1000 classes over almost a week, the very first 20 convolution layers were pre-trained, with a conventional pooling layer in the middle and a totally linked layer at the end. A predictive accuracy of 88 percent was attained by the pre-trained network on the validation set. In the preliminary training, a photograph with a resolution of 248 by 248 pixels was used as an input. Concatenation of 20 pre-trained convolution layers was used for detection, which included 4 convolution layers and also 2 fully connected layers with variable weights, all of which were concatenated at the conclusion of the 20 convolution layers. Additionally, the image size was raised from 248 × 248 to 448 x 448. The output layer makes use of a linear activation function and generates an output consisting of a bounding box coordinates, confidence score, and class probabilities, among other things. In addition to the coordinates, height and size of the bounding boxes are normalized. With the exception of the last layer, the leaky relu activation function is utilized for all layers. A total squared error in the outcome is used to optimize the model, which is readily optimized due to its simple design. The difficulty with the sum of squared mistakes, on the other hand, is that it gives the same significance to both the bounding box and the classification errors. Furthermore, for grid cells that do not include any objects, confidence scores trend in the direction of zeros, which has the effect of subduing grid cells that do contain items. In order to solve this, the loss feature was modified to penalize bounding box predictions more severely, as well as confidence scores for bounding boxes without objects being penalized considerably less severely. Aside from that, the sum of squared errors gives equal weights to errors for both big and small bounding box boundaries. This is in advance calculated by square rooting the bounding box elevation as well as the width forecast. YOLO predicts the same number of bounding boxes for each cell in the grid as it does for the previous cell. Throughout the training process, one bounding box predictor is assigned to the task of predicting the object, and this predictor has a much better present intersection than the union with the ground truth box. This

assists bounding box predictors in becoming more accurate in their predictions of diverse spatial objects [18]

**Fig. 7:** YOLO model architecture [18]




## Section 3.5.2: Training

The image was divided into a grid of seven by seven pixels on the paper. The paper selected two bounding boxes B which predicted two bounding boxes as well as a single set of class probabilities. A further consideration is that the model was trained using the PASCAL VOC dataset, which has 20 discrete classes, hence the class probabilities C were set to 20. Each grid cell anticipates a 30-value vector, two 5-value bounding boxes, and a total of 20 class probabilities. Regularization and augmentation were used to avoid overfitting on training images in order to prevent overfitting. Translation and random scaling were applied to the original photographs in order to enhance them. The centers of items in an image may lie near many cells, causing YOLO to project multiple bounding boxes for a single item when the centers of objects in a picture are near multiple cells. In order to reduce repeated detections, the method of non-maximal suppression is used. A single object's overlapped bounding boxes are detected using the IOU threshold, which is used in this case. It is decided which boundary box to use if numerous bounding boxes with a greater IOU limit overlap [18]. The bounding box with the greatest confidence score is chosen in this case.

## Section 3.5.3: Limitations

When it comes to object prediction, one of the major limitations of YOLO is that it can only anticipate one thing per grid cell. This inhibits the model from recognizing objects with centers in the same grid cell that are near to each other. As a consequence, the model has difficulty distinguishing between small elements in the image that look close to one another. Given that YOLO derives the box coordinates from the object's properties, it has difficulty detecting objects with unusual aspect ratios. Some of the most important properties in training are also lost because of the several down-sampling layers used in the model's construction. For the purpose of computing weights errors, YOLO treats both large and small bounding boxes in the same way. This is a concern because small box errors are significantly more expensive in terms of IOU than big box errors. Small box errors are therefore more difficult to detect [18]

| Real-Time Detectors | Train | mAP | FPS |
|---|---|---|---|
| 100Hz DPM [30] | 2007 | 16.0 | 100 |
| 30Hz DPM [30] | 2007 | 26.1 | 30 |
| Fast YOLO | 2007+2012 | 52.7 | **155** |
| YOLO | 2007+2012 | **63.4** | 45 |
| Less Than Real-Time | | | |
| Fastest DPM [37] | 2007 | 30.4 | 15 |
| R-CNN Minus R [20] | 2007 | 53.5 | 6 |
| Fast R-CNN [14] | 2007+2012 | 70.0 | 0.5 |
| Faster R-CNN VGG-16[27] | 2007+2012 | 73.2 | 7 |
| Faster R-CNN ZF [27] | 2007+2012 | 62.1 | 18 |

**Fig. 8:** YOLO comparison with other detectors on VOC [18]

### Section 3.5.4: Performance

YOLO performed better than all other real-time detection algorithms, with a mean absolute performance (mAP) of 63.4 and an inference time of 45 fps (frames per second). Fast and Faster R-CNN, on the other hand, were more accurate than YOLO for non-live detections with a time of fewer than 10 frames per second [13].

## Section 3.6: YOLOv2

After releasing YOLOv2, which featured architectural improvements to eliminate the initial work limitations, the same authors published YOLOv3 in 2017 [26]. When it came to pre-training, YOLOV2 used a bespoke darknet-19 architecture, which consisted of nineteen convolution layers and five max pooling layers between them. Later, for object identification, six layers were concatenated at the end of the design, resulting in a 30-layer architecture for YOLOv2. It was proposed in YOLOv2 to leverage the concept of anchor boxes, which had previously been used in Faster R-CNN, to assist the users in identifying small objects that occurred in groups of users. Instead of the completely connected layer utilized by YOLO to anticipate the outcome, YOLOv2 relies on support boxes to create the final result. It predicts up to B bounding boxes for each grid cell by using B anchor boxes, and for each bounding box, a set of class probabilities is generated, enabling the model to predict up to B objects for each grid cell. A significant difference between YOLOv2 and the previously mentioned Faster R-CNN is that it applies k means clustering on training photos to locate anchor boxes with improved prior fit. Because the image size has been reduced significantly at the time of production, YOLOv2 originally had difficulty detecting little objects in an image at first. As a result, it was difficult for the model to preserve fine-grained traits as more and more layers were added. When it came to dealing with this, the model made use of a method known as identity mapping was used. It was an outstanding upgrade to the YOLOv2 model, enabling it to better converge and generalize as a result of batch normalization. For the purpose of

eliminating dropout layers and reducing overfitting, batch normalization was performed after all convolution layers [26].

To begin the training, the darknet-19 model was trained for 160 epochs using the ImageNet classification dataset with an input image size of 248 by 248 pixels. The optimizer employed Stochastic Gradient Descent (SGD) having a learning rate of 0.1, and was implemented in Python. Cropping, zooming, rotating, changing the exposure and other augmentation procedures were also used throughout the training process to enhance the images. The same model was fine-tuned for classification with image size of 448 by 448. The last convolution layer was six convolution layers with random weights for object identification, and the size of the input image was 416 x 416 pixels. Three 3 × 3 convolution layers were utilized, followed by three 1 x 1 convolution layers, with the number of filters varied based on the length of each grid cell's vector. The number of filters employed in the last three 1 x 1 convolution layers varied based on the length of the vector for each grid cell. Using Pascal VOC with 20 classes as the assessment dataset, the outcome of the model architecture reduces the original 416-pixel image to 13 x 13 pixels. Using 5 Anchor boxes, the number of bounding boxes B was determined to be 5. Each box had five values, including coordinates and a confidence score, and was designated as B. As a consequence, each grid cell has a vector of length 125. A result of this was that each of the one-by-one convolution layers had 125 filters, and the final output of the model was thirteen by thirteen by 125 pixels. The VOC dataset was used to train the model over a period of 160 epochs. It was determined that the learning rate should be 0.001, and that the momentum and weight decay should be set to 0.9 and 0.0005, respectively, during the first 60 epochs. It took 60 to 90 epochs for the learning rate to reach 0.0001, and it took 90 to 160 epochs for the learning rate to reach 0.00001. During training, the same augmentation methods that were used in YOLOv2 were used. The results were similar.

YOLOv2 was lightning quick showing a 45 frames per second inference time and accuracy that was state-of-the-art at the time. While algorithms such as SSD and RetinaNet surpassed YOLOv2 in terms of mean absolute performance (mAP), this was not the case a few years later. YOLOv2 lacked critical skills detecting smaller objects was not very good [26]

| Detection Frameworks | Train | mAP | FPS |
|---|---|---|---|
| Fast R-CNN [5] | 2007+2012 | 70.0 | 0.5 |
| Faster R-CNN VGG-16[15] | 2007+2012 | 73.2 | 7 |
| Faster R-CNN ResNet[6] | 2007+2012 | 76.4 | 5 |
| YOLO [14] | 2007+2012 | 63.4 | 45 |
| SSD300 [11] | 2007+2012 | 74.3 | 46 |
| SSD500 [11] | 2007+2012 | 76.8 | 19 |
| YOLOv2 288 × 288 | 2007+2012 | 69.0 | 91 |
| YOLOv2 352 × 352 | 2007+2012 | 73.7 | 81 |
| YOLOv2 416 × 416 | 2007+2012 | 76.8 | 67 |
| YOLOv2 480 × 480 | 2007+2012 | 77.8 | 59 |
| YOLOv2 544 × 544 | 2007+2012 | **78.6** | 40 |

**Fig. 9:** YOLOv2 comparison with other detectors on VOC [26]

## Section 3.6.1: Performance

With a detection period of 40 frames per second, YOLOv2 had the highest mAP of 78.6 out of all detectors on PASCAL VOC, outperforming all others. YOLOv2 using a 288 by 288 input image was the fastest, with a frame rate of 91 frames per second and a mean acquisition time of 69 milliseconds[26]

## Section 3.7: YOLOv3

The YOLOv3 model, which was a newer edition of the YOLO verisons [27]. It was built in 2018 by Joseph Redmon and his co-workers. The approach was created to increase accuracy while remaining as quick as feasible without sacrificing speed. [28]. When YOLOv3 was pre-training on the ImageNet 1000 class dataset, it used an original 53 layer network from Darknet, which was built from the ground up. After that, the network was enlarged by 53 layers to accommodate the object detection task, giving YOLOv3 a total of 106 layers in total. However, despite the fact that YOLOv3 is slower than YOLOv2, it still processes photographs at a rate of 30 frames per second due to its intricate and vast design.[27]

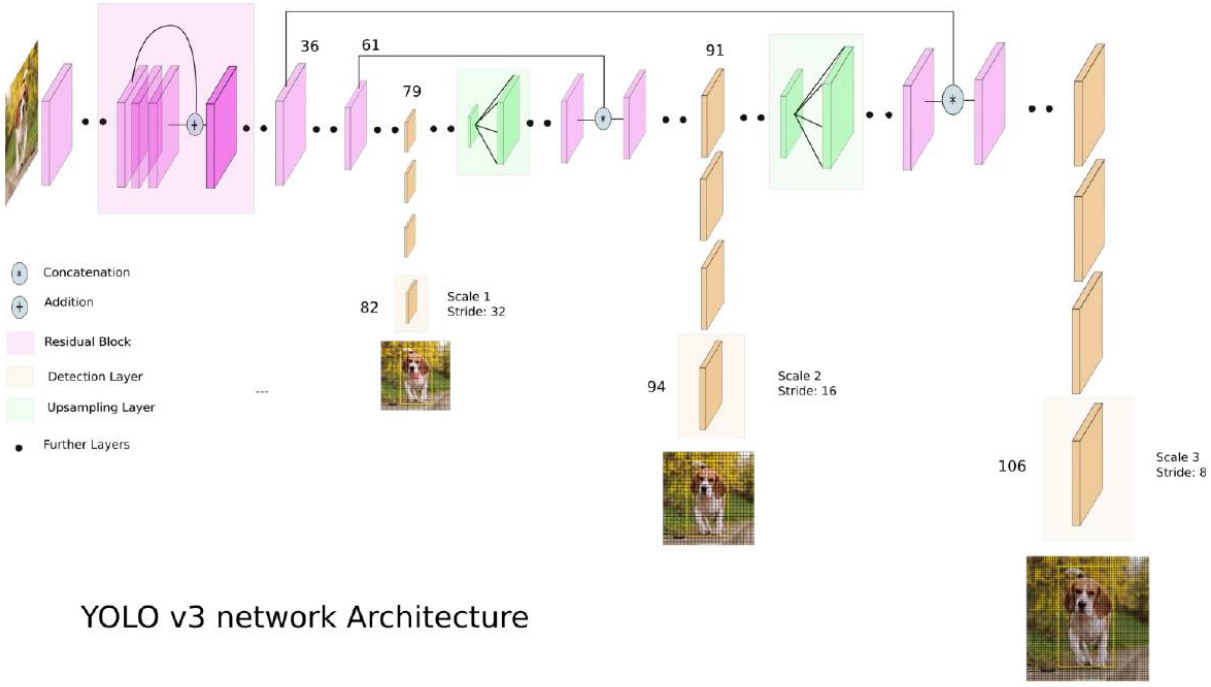



**Fig. 10:** YOLOv3 model architecture

Moving further into the network, YOLOv3 used upsampling layers in the network architecture to ensure that the required features were maintained. One of the most significant improvements to the YOLOv3 was that in could detect three different sizes of objects, as seen in figure 8. As YOLOv2, a 1 x 1 layer of convolution is used to construct the final result, along with a number of filters that are dependent on the parameters and dataset. There are three 1 x 1 convolution layers applied to three different sized feature maps, which are applied at three separate places in the network to detect at three different scales. The YOLOv3 model was evaluated on the COCO dataset, which had 80 classes, in the research that was proposed [27], [29]

The picture size is reduced by factors of 32, 16, and 8. It is possible to identify large objects using the output of size 13 by 13, medium objects using the output of size 26 by 26, and tiny objects using the output of size 52 by 52. In addition to the three anchor boxes on each scale, a total of nine anchor boxes were used. K means is used to generate anchor boxes by clustering with and then arranging them in descending order of importance. The largest of those three anchor boxes are used in the 13 x 13 grid, the next three are used in the 26 x 26 grid, and the smallest three are used in the $52 \times 52$ grid [22], [23].

Cross entropy, error is utilized instead of the sum of squared error that was used in previous versions of YOLOv3. To anticipate each box's confidence score and set of class probabilities, YOLOv3 used logistic regression activation rather than Softmax, as opposed to the Softmax method. As a consequence, it was possible to forecast many classes for a single item based on the class likelihood score [22], [23].

## Section 3.7.1: Performance

In a comparison to other state-of-the-art detectors at the time, such as RetinaNet, YOLOV3 achieved a mean absolute performance (mAP) equivalent to that of those other detectors, but with an inference time of just 22ms, which was much faster than other detectors. Yolov3 had the greatest mean absolute probability (mAP) of 57.9, and it had an input network size of 608 x 608 pixels. [27]

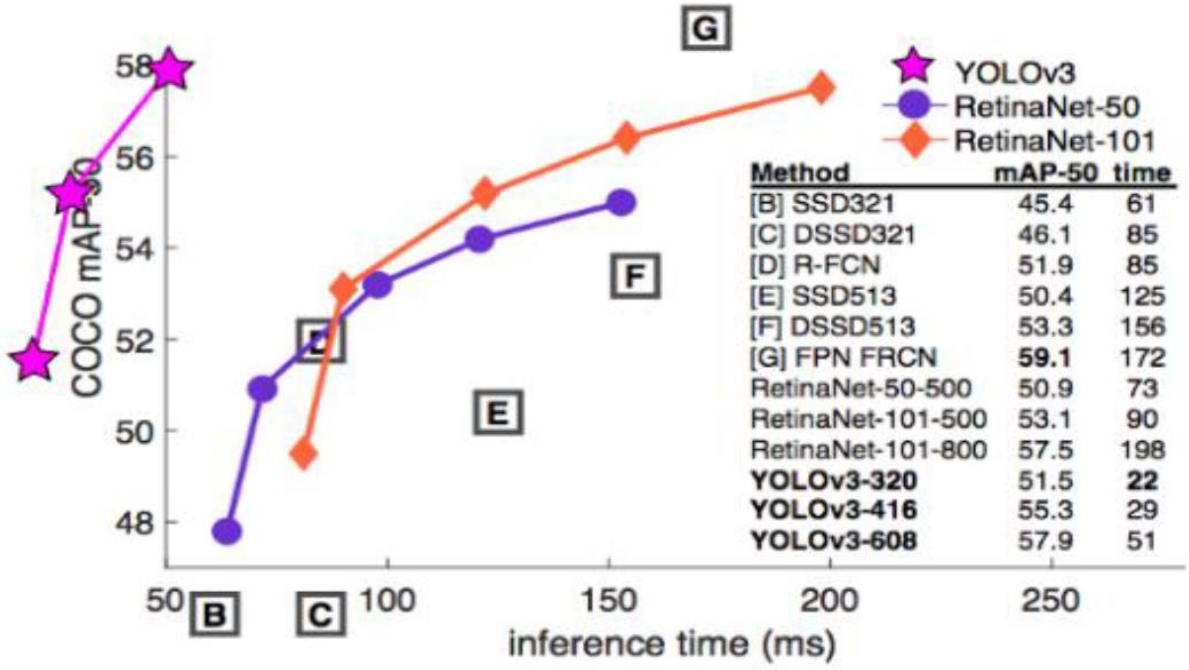



**Fig. 11:** YOLOv3 comparison with other detectors on COCO

## Section 3.8: YOLOv4

In his study published in 2020, Alexey Buchkovskiy proposed the term "YOLOv4" [30]. Most algorithms used prior to YOLOv4 needed bigger batches of data for training, which was a difficult process to do with a single GPU. YOLOV4 is an object detection technique that uses just a single GPU for training and only little batches of data.

Yolov4 is incredibly quick and has incorporated certain universal features as Mish activation, Mosaic augmentation, cross stage partial connections, CIoU loss, and other similar characteristics to achieve high precision. The YOLOv4 model architecture is comprised of three major components:

1. Backbone: Object detection tasks are fine-tuned using the backbone model, which has been pre-trained on datasets such as ImageNet. Because of this, the feature maps obtained in these pre-trained models are very valuable in later layers of the network, which are employed for object detection. There are a variety of YOLOv4 backbone models available. Some of these backbones do better at classification while others are good for object detection task. For example, CSPDarknet53 is more better for object detection tasks. It is necessary to have a larger size input for the backbone model in order to be able to train the tiny object detection more effectively [30]
2. Neck: Second, the neck portion of the model is made up of layers that are located in the midst of both backbone and head networks. Neck performs the function of a feature extractor, extracting features from various areas of the backbone network. YOLOv4 also uses Spatial Pyramid Pooling (SPP) to expand the receptive field. It enhances the performance by increasing the receptive field. YOLOv4 employs a modified PAN that concatenates data rather than adding it all together. PAN employs two approaches: bottom-up and top-down[30]
3. Head: YOLOv3 and YOLOv4 both architectures have same head, except that anchor boxes are employed for purpose of detection, at three distinct scales instead of only one. Using varied scales may aid in the detection of objects of varying sizes. [30]

In addition, two approaches, Bag of freebies (BOF) and Bag of specials (BOS), are included in YOLOv4 in order to increase the effectiveness of the YOLOv4 object detector. With various characteristics, a bag of freebies was created to aid in the advancement of model training. One of the features of BOF is data augmentation, which allows the model to generalize more effectively on previously unknown data. The use of various image augmentation methods, such as rotation and saturation changes as well as hue changes, is common, and the same techniques are used to create bounding boxes. New augmentation strategies, such as cut-off and random erasing, have also been shown to increase the performance of convolution networks. Regularization strategies other than dropout include DropBlock and DropConnect, both of which have shown to be quite effective in avoiding overfitting on training data. DropBlock is used to maintain regularity in the YOLOv4 backbone network. YOLOv4 has BOF also has a characteristic of CIOU. It calculates the loss of bounding box in terms of the distance between the centre points of the ground and the predicted, the aspect ratios of the ground and predicted, and the overlap between the ground and predicted. The CIoU aids in the improvement of accuracy and convergence of the model. Bags of Specials incorporates inference-related elements that help to enhance overall performance. Among the BOS networks that are used to increase the receptive field and improve feature learning are the SPP, RFB, and ASPP. A modified variation of the Spatial Attention Module is also used by YOLOv4 to build an improved feature map, which is another aspect of the program. In addition, YOLOv4 makes use of a Mish activation function to increase test accuracy. [30]

Another feature that is the Mosaic augmentation, is also included as a characteristic that is utilized in YOLOv4, which is a video that blends four photos into a single frame and uses mosaic augmentation to do this. This assists the model in selecting small batch sizes and expedites the training process. The Self-Adversarial Training feature is divided into two levels. Instead of using a filter in the first step, the network refreshes the picture data. Second, the model is trained on the new changed picture in order to recognize objects in the third step. [30]

## Section 3.8.1: Performance

On the COCO dataset, YOLOv4 achieved an AP and AP50 of roughly 44 and 66, respectively, which was on par with other state-of-the-art detectors (SOTA), with an inference time of more than 30 fps, which was much quicker than other detectors. [30]

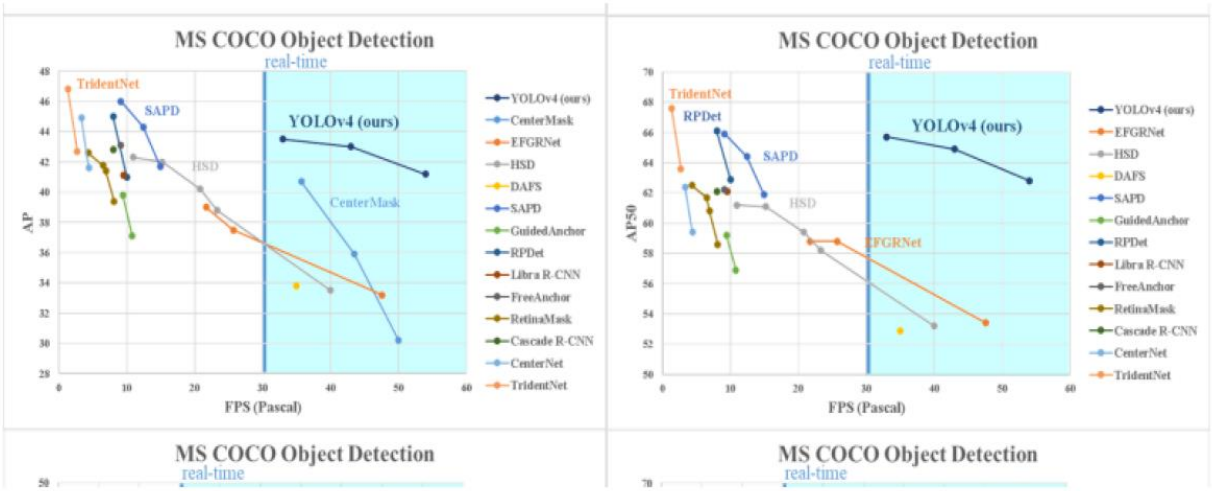



**Fig. 12:** YOLOv4 comparison with other detectors on COCO [30]

## Section 3.9: YOLOv5

YOLOv5 is the most recent version of Glenn Jocher's YOLO family of models, having been released just two months after the previous edition, YOLOv4. If we look at the model's architecture, YOLOv5 is similar to YOLOv4 in that it uses the same backbone, neck, and head as the previous model. Yolov5 is implemented in Pytorch [31] rather than the Darknet framework, which is more efficient. In contrast to previous versions, YOLOv5 makes use of the a.yaml [32] configuration file for configuration. It has number of different versions, which include the YOLOv5s, YOLOv5l, YOLOv5m, YOLOv5n, and YOLOv5x, which are all different models. In terms of size, the smallest YOLOv5 model is the YOLOv5s [25], while the largest is the YOLOv5x [33]

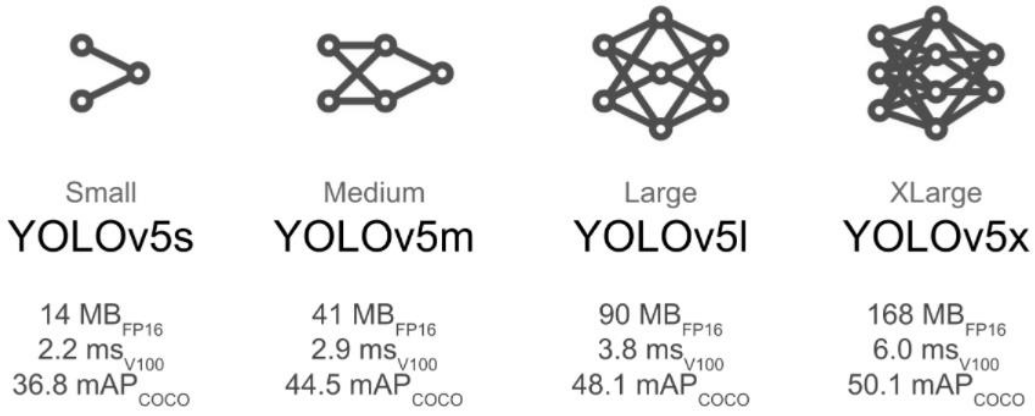



**Fig. 13:** YOLOv5 versions comparison

## Section 3.9.1: Performance

On the COCO dataset, YOLOv5x was able to get an AP that was very near to that of the SOTA EfficentDet, and with a time of less than ten milliseconds. The YOLOv5x algorithm, on the other hand, has an AP of over 50 on COCO and a 8ms inference time [33].

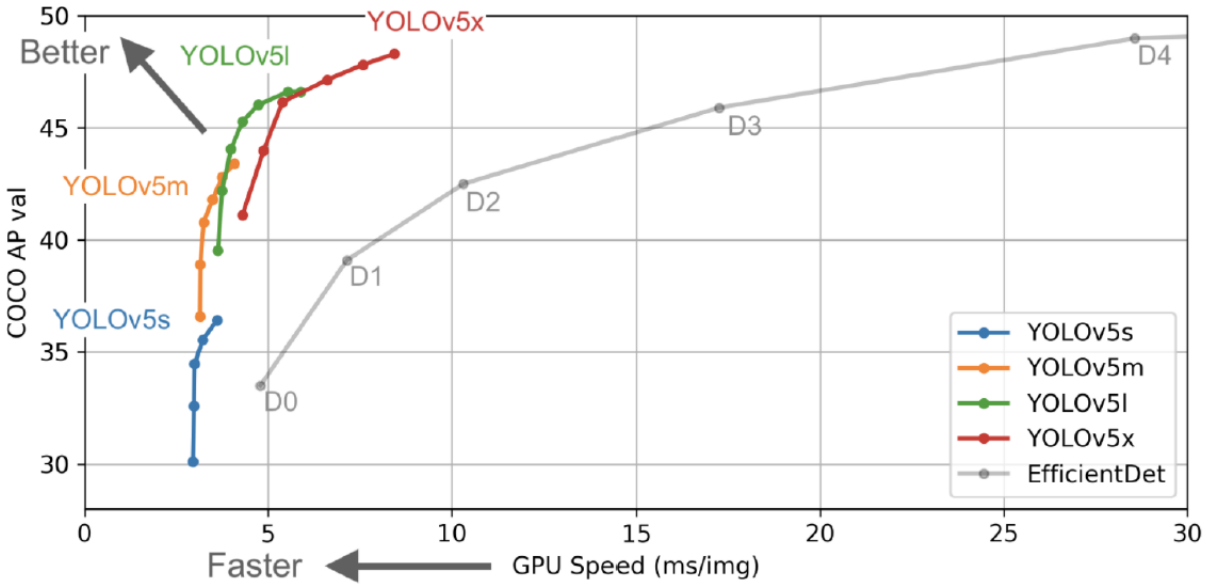



**Fig. 14:** YOLOv5 comparison with other detectors on COCO [33]

## Section 3.10: Class Activation Maps

A side effect that occurs is the loss of inference transparency because of the large number of parameters in a neural network. Because it is impossible to determine the grounds for a decision taken by a neural network, it is difficult to determine whether the prediction was correct or the network is inferring incorrectly on the wrong foundation. As an example, if a model has been created and trained on pictures of Jesus Christ. Most of the times, Jesus is represented as crucified in paintings. In this situation, the model may activate only based on the crucifixion, rather than because of Jesus. This raises the question of whether the model is capable of recognizing Jesus even if he is not in the image of the crucifixion. Class activation mapping (CAM) [16] is a technique, which produces heatmaps on the basis of inferential grounds to aid in such cases. The heat maps may be used in combination with the input image as an overlay to provide additional information of where the model concentrates for its prediction. So in essence, it is a visualization of input regions, that can be understood by humans, which display the model prediction on the images.

# Section 4: Implementation And Evaluation

The current section relates to experimental setup and the parameters related to dataset and its processing as well as the models used for training and inference. A high-level summary of the data is first given. Secondly, a discussion of data pre-processing will be provided. Afterwards, the implementation setup of the models will be discussed. Lastly, results of both of the models' predictions will be discussed at the end of this section along with CAMs.

## Section 4.1: Data

Milani et al. published the ArtDL paintings data set, which was used in this thesis as a subset [6]. It includes 42.479 paintings representing Christian Saints which were taken from IconClass [4]. The bulk of the images were made during Europe's Renaissance. Murals, frescoes and canvas paintings are among the many different art styles and materials present in the data collection. The paintings are approximately 60% in color and 40% in grayscale. The 10 saints in this dataset are: **Virgin Mary, Anthony of Padua, Saint Dominic, Francis of Assisi, Saint Jerome, John the Baptist, Paul the Apostle, Saint Peter, Saint Sebastian and Mary Magdalene.** Each picture has at least one representation of one of these saints, although it may contains many saints. If a picture portrays a number of saints, just one saint is selected to serve as the annotation for the artwork. The annotations by Milani et. al are labels that indicate the presence of a certain saint in the picture.

The dataset contained most number of both colored and grayscale images of Saint Mary at about 19,399 [6] and iconographic analysis on her paintings is possible due to some key features that most of her paintings and her other artworks contains. As, discussed in **Sec 2** about iconography of

Christian religion, some of the most notable features of Virgin Mary's iconography [2][3] in the ArtDL dataset were found to be:

1. Robe that Holy Mary is wearing. The dataset contained huge amount of paintings of hers with various possibilities of colors of her robes and dresses:
    a. Blue robe with red dress underneath
    b. All blue robe and dress
    c. Red robe with blue dress underneath
    d. All red robe and dress
    e. Shade of blue could range from very light almost whitish blue to dark or even blackish-blue
    f. Apart these, there were a minority paintings of Holy Mary with dresses in different colors like Golden and White.


**Fig. 15:** Sample images from the ArtDL dataset [6]

2. A Holo or circle on or over the head of Holy Mary is also found in almost all of the images indicating divine nature of her being. The Holo could either be filled or un-filled, above or behind the head as shown in first and third images in top row and middle image in second row of figure 14.
3. Another interesting key point that was observed was Virgin Mary's scarf. Mostly, it was white with shades ranging from transparent to fully opaque white. At times, it could also be the part of the red robe that covered her body as shown in first image from right in the top row in figure 14.
4. Lastly, Holy Mary, most of the times is shown together with Baby Jesus.

All four of these features were considered in this thesis to perform iconographic predictions upon and they were annotated using bounding box annotations on Roboflow website. As, the robe (both blue and red colors), scarf and the baby Jesus were all mostly together spatially, so one bounding box was made for these three features. However, Holo were separately annotated whenever they occurred visibly enough in the portraits as sometimes holos were very weakly drawn with just a faint non-continuous line.

From about 10,000 colored portraits of the Saint Mary's dataset, 1000 images were taken for the work of this thesis. After their annotations, they were then preprocessed and augmented as discussed in the next section.

## Section 4.2: Pre-processing

Recently, many computer vision solutions have sprung up providing developers with various useful applications in this field. Roboflow [34] is one such solution, which provides, among other options, to load your custom dataset, check its heath, do annotations and preform various preprocessing and augmentation steps and export it in commonly used formats like Tensorflow TFRecord [35] and YOLO Pytorch [36]. This saves important time when making and experimenting with a computer vision model and make the process a bit more straightforward.

1. In first step, all the images were split into train, validation and test sets, a generally accepted way of choosing these sets are 80% for training and 10% for each of the validation and test sets. Therefore, The selected 1000 images were split into following sets:

    - Train set: 840 images
    - Validation set: 100 images
    - Test set : 100 images

2. Secondly, they were then resized to 640x640, as resizing an image is necessary for conforming it to the dimensions of a neural network's input layer. However, the model we will be using also does this itself. Resizing to a lower size also decreases processing costs and training time.
3. Finally, Auto-Adjust Contrast was used for boosting the contrast. It does so on the image's histogram which also improves normalization and line detection in different lighting conditions. As we were dealing with images with some low contrast images and there were segments of images with saturated contrast, so flattening the contrast of the image using preprocessing is a recommended step. Also, edges become clearer as neighboring pixel differences are exaggerated [37]

Augmentation is used to create an altered training data based on existing examples. This results in improved model performance, especially on small datasets [38]. Therefore, after the initial pre-

processing step, five types of augmentations steps were also done on the dataset right from the Roboflow's working environment:

1. **Hue**: Between -25° and + 25°

   This augmentation technique is used to randomly alters the colour channels of an input image. This causes a model to consider more colour schemes for all the objects and scenes in input images. As mentioned earlier in this section, that images of Saint Mary had varying degrees of blue and red shades, so applying just a small 25 ° Hue augmentation will surely help the model in learning those differing shades better.

2. **Saturation**: Between - 25% and + 25%

   Saturation augmentation technique is closely similar to applying hue except that it changes the vibrancy of the image. Therefore, changing and adjusting the saturation of an image can increase your model performance better as portraits made by different paintings of Holy Mary could differ in white-balance and different light settings.

3. **Brightness**: Between - 25% and + 25%

   It Increases the variability of picture brightness in order to make your model more tolerant to variations in lighting and camera settings.

4. **Exposure**: Between - 25% and + 25%

   It also Increases the variability of picture brightness in order to make your model more tolerant to variations in lighting and camera settings.

5. **Noise**: Up to 5% of the pixels

   Noise is the purposeful alteration of pixels in such a way that they seem to be different from what they should have represented. It is done by randomly changing certain pixels to entirely white or completely black. Researchers have considered that the implementing blurring and noise had the most adverse effect on top-1 and top-5 accuracy classification [39], [40].

The images before and after pre-processing and augmentations are shown in below figure. Please note for the sake of this display, sizes and aspect ratios of the original (first row) images had to be altered. The output images (second row) after these initial steps, all had same size as mentioned earlier.

**Fig. 16:** Images of Holy Mary from the ArtDL dataset [6] before and after pre-processing and augmentation steps

After these steps, the dataset was exported from Roboflow's workbench. There are two ways to export the dataset from Roboflow. First is to download the dataset in the required format, i.e. Tensorflow TFRecord, JSON COCO etc. Second way is to get the download code and use it directly into your Jupyter notebook or Terminal via Roboflow pip package.

## Section 4.3: Implementation Setup

For running experimental setups, two different object detection models were selected, YOLOv5 [33] and Faster R-CNN [24]. Both of them are State-Of-The-Art models and their backgrounds, architectures, evolutions and performances had been discussed in section 3, with Faster R-CNN and YOLOv5, being discussed in sections 3.4 and 3.9 respectively.
The setup for implementation of models was on Google Colab [41]. The GPU assigned was Tesla P100 with 16GB of Memory and RAM assigned was of 32GB. All the code was in Python [42].

### Section 4.3.1: Training with YOLOv5

YOLOv5 model [43] is currently one of the most famous and widely used object detection algorithms. This leads to a wide range of learning resources in the every form available [44]. Out of which, Ultralytics's documentation [45] is the best as they were the original developers of this version of the model.

Some major requirements that needs to be installed after cloning the Ultralytics's YOLOv5 repository are:

- Python>=3.7.0
- PyTorch>=1.7
- CUDA/CUDNN
- torch>=1.7.0
- torchvision>=0.8.1
- pandas>=1.1.4
- scikit-learn==0.19.2
- tensorflow>=2.4.1

All of the requirements were easily installed using a requirement text file. Afterwards, Dataset from Roboflow [34] was imported and incorporated into specific train, valid and test folders in Colab by using their python package. The next step was to prepare the dataset yaml [44] file in order for YOLOv5 to be able to access the data. Now comes the part to select which YOLOv5 version needs to be ran. YOLOv5 has five versions namely, YOLOv5n, YOLOv5s, YOLOv5l, YOLOv5m, and YOLOv5x  with each having a different model architecture. Pre-trained weights [46] for each model are accessible in Ultralytics's repository's weights folder. Out of these versions, YOLOv5s and YOLOv5x were selected and trained for this thesis.

YOLOv5s and YOLOv5x both predict three anchor boxes per grid cell at three distinct scales to identify objects of varying sizes. The model was initially ran for 150 epochs. This resulted in an overfit model. Analyzing the train and validation loss function graphs, it was obvious that model started to overfit just after 25 epochs due to size of data being small and advance model architecture of YOLOv5.  Moreover, Transfer Learning was done by using pre-trained Yolov5 weights exported from the YOLOv5 authors' repository [46]. In this experiment, the batch size was selected to be 32 as to be a trade-off between fast and efficient performance of the mode. Other main settings that were utilized for the hyperparameters were:

- Initial learning rate lr0=0.01
- Final learning rate lrf=0.01
- Momentum=0.937
- Weight_decay=0.0005

From all of the YOLOv5 versions, the smaller (YOLOv5s) and the largest one (YOLOv5x) were selected. The results of comparison of both is listed in Appendix A. Here, results of only the one version, i.e.YOLOv5s is discussed.

## Section 4.3.2: YOLOv5s Results

Below is a summary of YOLOv5s, the smaller version that was trained for 25 epochs.

```
YOLOv5s summary: 213 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 2/2 [00:01<00:00,  1.14it/s]
                 all        100        203      0.846      0.889      0.915      0.627
                Holo        100        105      0.713      0.829      0.849      0.596
                Mary        100         98      0.979      0.949      0.981      0.657
Results saved to runs/train/yolov5s_results
CPU times: user 5.16 s, sys: 882 ms, total: 6.04 s
Wall time: 8min 8s
```

**Fig.17** Summary of the trained YOLO v5s showing Precision, Recall, mAP@0.5 and mAP@0.5:0.95

Figure 17 shows most of the basic yet important parameters for YOLOv5s. The model took only 8 minutes to train. Even with such low time to train, the model showed promising results with Precision for 97.9% for class Mary and 71.3% for Holo class. Such difference in both precisions could be answered by the fact that Mary class was trained on a bigger region as compared to Holo class whose region was not only small but also considerably varied too like filled, un-filled on head, above head etc. This is also shown by Recall metric where recall is again comparatively lower for Holo class. Both mAP[1] metrics showed good results for both classes with total mAP for 0.5 being 91.5%

[1] The mAP calculates a score by comparing the ground-truth bounding box to the detected box. The higher the score, the better the model's detection accuracy. It is stated over different IoU (intersection over Union) thresholds, from 0.5 to 0.95. mAP@0.5 means ground-truth and predicted box shares 50% of area.

Moving to metrics plots, figure 18 shows plots for YOLOv5s on our dataset. Considering the top two plot of mAP, there is a steady rise with mAP@0.5 and then the difference is not considerable after 15th epoch. However, we see that mAP for 0.5:0.95 continues to rise. Nevertheless, we cannot let the model train further because of the overfitting that occurs after about 25 epochs (see more about it in Appendix A)

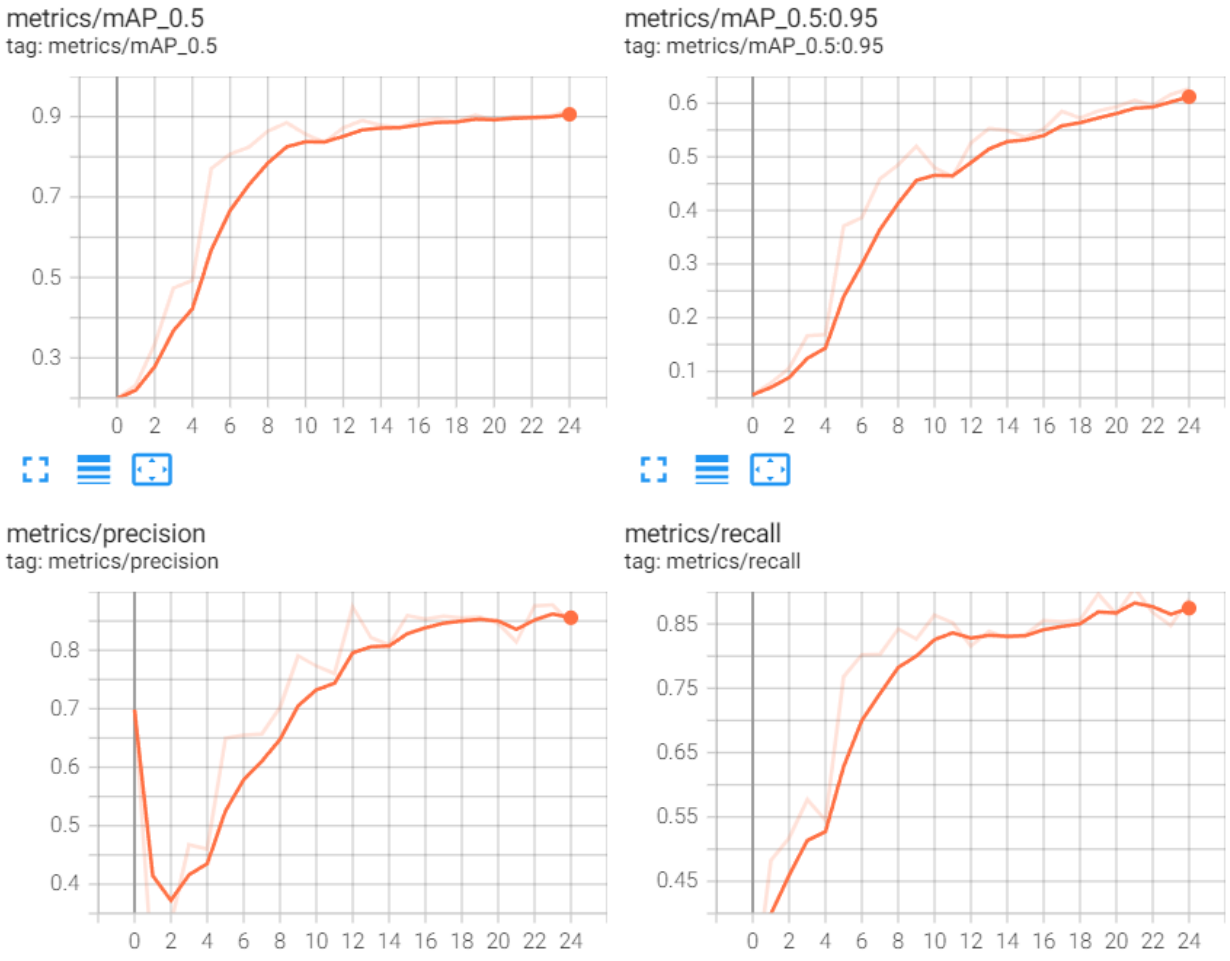



**Fig.18** Plots of the mAP, precision and recall for the trained Yolo v5s

Considering train and validation losses in figure 19, we see the steady decrease in train and validation losses for box, classification and objectness losses. Both train and validation losses decreasing together means that the model is not overfitting and from our initial run for 150 epochs, we know that training the model further will result in overfitting and an increase in validation losses and hence, resulting in bad inference (see more about it in Appendix A)

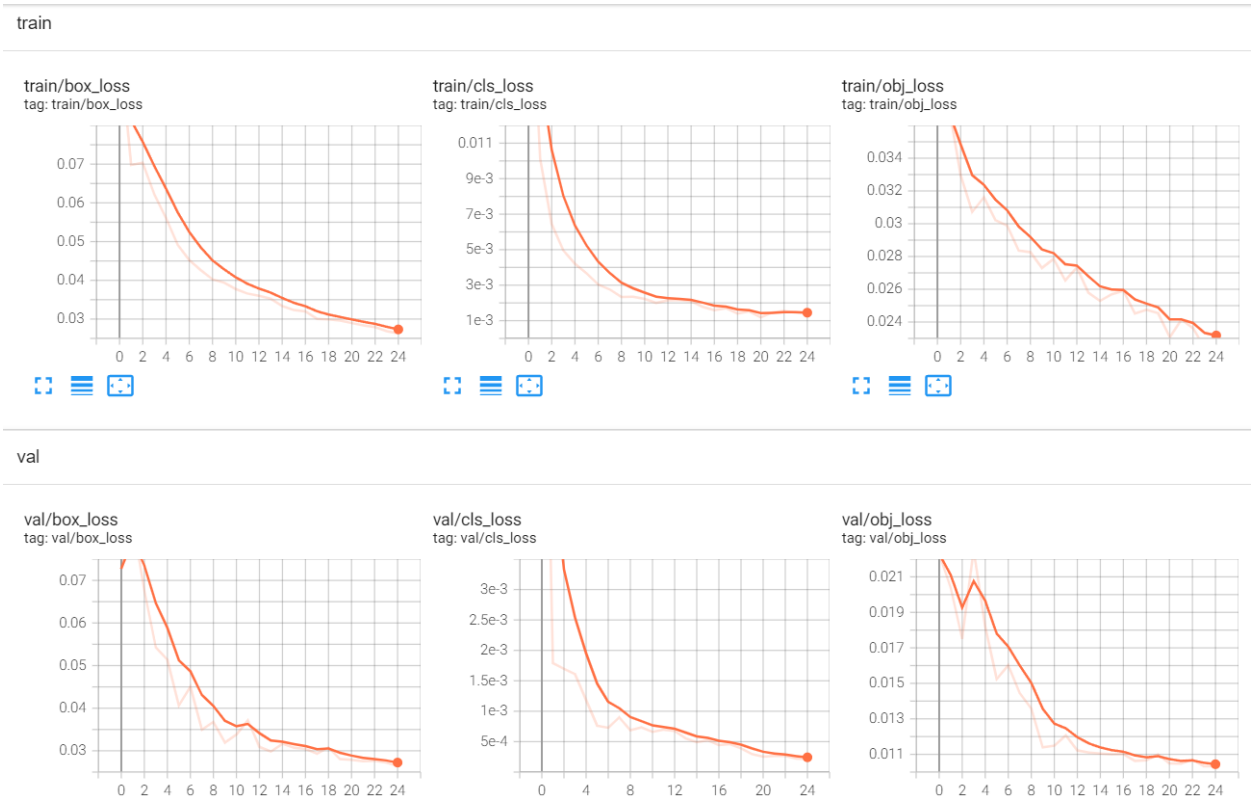



**Fig.19** Plots of the metrics for the trained YOLO v5s comparing train vs validation losses

Finally, we have the inference images as shown in below fig.20. These images were obtained by using YOLO v5s best weights. After the model is trained on images from the training set, the model's best weights are saved and using that YOLOv5 was required to draw the bounding boxes on the test set that the model has not seen.


**Fig.20** Inferenced images from the trained YOLO v5s

An interesting remark that the model was unable to predict Saint Mary in third row middle picture as clearly there is not much red or blue dress shown. However, model was able to predict her in the left-most picture of the same third row even though the image did contained many other people.

## Section 4.3.3: YOLOv5 CAM Results

Class Activation Mapping is generally being done on Image Classification tasks and not object detection tasks [51]. The output of object detection algorithms are generally in a form of dictionaries of bounding boxes, labels and scores. They then do an argmax on scores to find the highest scoring category. Therefore, to get a thermal heatmaps like CAM is not straight forward with them. The code needs to be dived into and altered to get specific outputs from specific layers. For the case of YOLO v5, some very recent experiments were done and included in a repository [48]. There are some issues found when ran on different setups and can require many changes in original code and dependencies to work [48]. This code when used on this thesis dataset, did show some heatmaps. However, the resulting maps are not perfect and cannot be taken into consideration when evaluating YOLO model's results as will be shown in below figures.

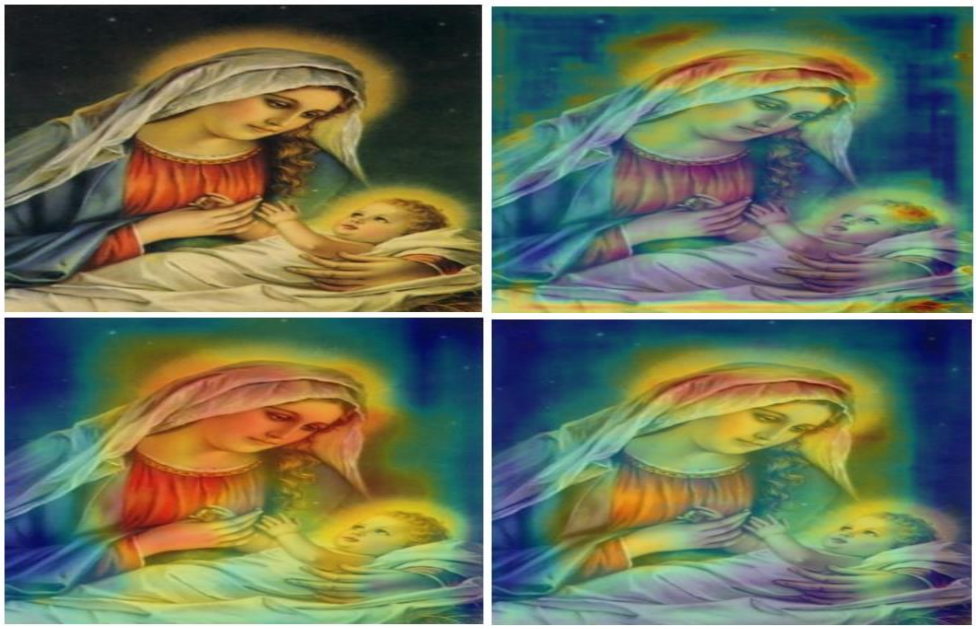


**Fig.21.1** CAM from three different layers on an image. Original image is top left

Although, the images shown in figure 21.1 is of a single original image (top left) with other images being output of different layers of YOLOv5. This CAM result is relatively discernable as activations shown in top right and below right images are concentrated on Holo and blue dress of Holy Mary. However, the heatmap shown on below right image in fig. 21.1 is disperse and not pointing to specific regions of the image. It is also to be noted that this image is one out of many images where

the CAM results are somewhat understandable relative to the objects the model detected. In almost all other images that were tested, CAM results were clearly not accurate (please see fig. 21.2) as the heatmaps were drawn on almost whole area of the image in regards to how it should solely be focusing on specific regions only.

Below are two more sample images with their respective CAMs, which explains why the CAM implementation on YOLOv5 was not achieved properly. The CAMs results are wrong as it shows that they are focusing on various random parts of the image.


**Fig.21.2** Original images and CAM results on them. The CAM has focused (denoted with red) on random parts of these images

## Section 4.4: Training With Faster R-CNN

Faster R-CNN was also implemented using Python using Google Colab. In this case, TensorFlow API [49] was used to get the implemented object detection models directly from Tensorflow Model Garden. It is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection model. Apart from this, model framework was of Keras [50].

As TensorFlow's Object Detection API was used here, there was no need to install dependencies separately. After configuring pipeline, we are ready to select feature extractor model. For that, in Faster R-CNN, mobile net was utilized because of its faster performance

All of the configurations for hyperparameters etc. were configured using a config. file. Afterwards, Dataset from Roboflow [34] was imported in TF Records format [49] because format is used for Tensorflow object detection models. After that, the dataset was incorporated into specific train, valid and test folders in Colab by using their python package. As a base feature extractor, ssd_mobilenet_v1[51] was used. Main reason of using Google's MobileNets was there high speed [52], which it achieves by using Depth-Wise Separable Convolutions [51]. The basic hyperparameters configurations were altered as to compare them to YOLOv5 model. Again, Transfer Learning was done by using pre-trained MobileNet weight exported from the TensorFlow repository. The weights were trained on COCO dataset [53]. Generally, Faster R-CNNs have better accuracies on various datasets than YOLO models [52], [54].

As there is a parameter for step size in Faster R-CNN implementation instead of epochs, a calculation was done to determine the number of step size equivalent to 25 epochs. This calculation gave a number of 781 as step size which equates to 25 epochs on the given dataset of 1000 and batch size of 32 [55], [56]

## Section 4.4.1: Faster R-CNN Training Results

The model used Google's MobileNet [51] as a feature extractor. Figure 22 shows Faster R-CNN's metrics of classification loss, localization loss and Total loss graphs. Please note that the model was trained for 781 steps as mentioned earlier.

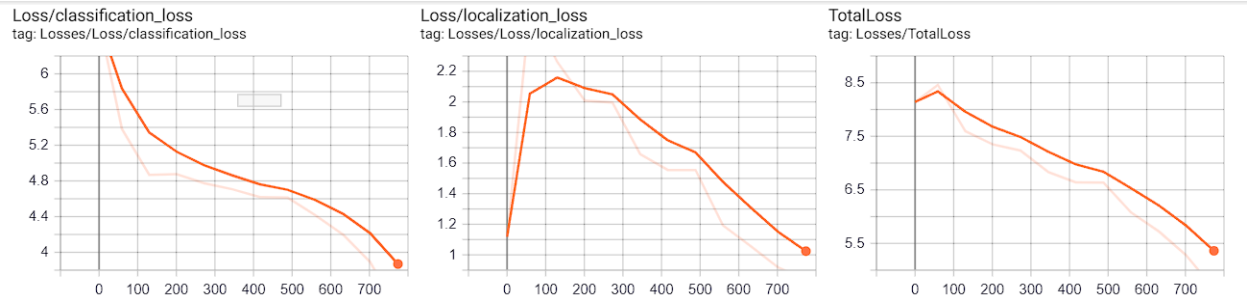



**Fig.22** Plots of Loss functions of Faster R-CNN

One observation was that the Faster R-CNN showed a Total loss of about 4.7%. Legacy issues- like the model only storing values for losses, shown in fig. 22 - prevented a direct comparison between Faster-R-CNN and YOLOv5. However, total time train for FRCNN was a bit higher at 9.5 minutes to YOLO's 8 minutes. The inferred images, which were almost same as for YOLO inferences images are also shown below in fig. 23:

## Section 4.5: Discussion

Quantitative results are shown in previous sections for both models YOLO and Faster-RCNN. Both Holo and Mary classes are balanced with 105 and 98 labels respectively and could be trained without any issue in this regard. The two object detection models (YOLOv5 and Faster R-CNN) chosen for this thesis were because of their high accuracies and fast results. The training models were initially ran for 150 epochs (see the appendix) resulting in overfitted data because of total annotated images being 1000 only. However, with this result, it was possible to see epochs, after which the model would start to overfit. In this case, it was after 25 epochs after validation loss for objects will start to rise with training loss still decreasing. Therefore, new training was done with 25 epochs. Considering figure 17 which summaries YOLO model's training results, it is observed that precision as well as mAP for Holo class is less than that of Mary class, although Holo had higher number of labels. A reason for that could be smaller size of the annotated region of Holo as compare to annotated region of Mary, which is many times higher. Moreover, another reason it seems is that often times, Holos on head of Holy Mary where not prominent and have changing styles; from an unfilled/filled circle, on head vs above head. This is confirmed by the lower recall Holo class has.

In section 4.3.4, CAM results from YOLOv5 output images are presented. Images for fig. 21.1 were the result of CAMs being overlaid on a single image. The output is showing three different layers of the YOLOv5 model but of a same one image. It is clear from this image that although the CAM is focusing on her head area for Holo detection and her body area for blue/red robe, CAM is also considering other regions of the image as well. This phenomenon is more clear in next image, fig. 21.2. Being an object detection algorithm, not very resources are present for implementing CAM on objection detection problems. Therefore, CAMs on inferred images were not promising and many times the map was shown all over the image instead of just focusing on specific objects in the image on which the detection was performed.

# Section 5: Conclusion

This thesis presented an experiment to do an iconographic analysis on the images of Holy Mary by using object detection models like YOLOv5 and Faster R-CNNs. These machine learning models were trained on specific region of interests like Holy Mary's clothing and were able to detect these region with good results. Afterwards, Class Activation Mapping (CAM) technique was used on the inferred images to get the discriminative regions of images from the model's predictions. However, the CAM implementation could not be done successfully because of the form of output data of object detection models.

# References


[1] “https://prothesiswriter.com/blog/how-to-write-an-acknowledgement-for-a-thesis].”

[2] “https://examples.yourdictionary.com/examples-of-iconography.html.”

[3] “https://www.christianiconography.info/maryPortraits.html.”

[4] L. D. Couprie, “Iconclass: an iconographic classification system.”

[5] N. Gonthier, Y. Gousseau, S. Ladjal, and O. Bonfait, “Weakly Supervised Object Detection in Artworks,” Oct. 2018, doi: 10.1007/978-3-030-11012-3_53.

[6] F. Milani and P. Fraternali, “A Data Set and a Convolutional Model for Iconography Classification in Paintings,” Oct. 2020, doi: 10.1145/3458885.

[7] G. Hua and H. Jégou, Eds., *Computer Vision – ECCV 2016 Workshops*, vol. 9913. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-46604-0.

[8] M. C. Marinescu, A. Reshetnikov, and J. M. Lopez, “Improving object detection in paintings based on time contexts,” in *IEEE International Conference on Data Mining Workshops, ICDMW*, Nov. 2020, vol. 2020-November, pp. 926–932. doi: 10.1109/ICDMW51313.2020.00133.

[9] G. Hua and H. Jégou, Eds., *Computer Vision – ECCV 2016 Workshops*, vol. 9913. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-46604-0.

[10] E. Cetinic, T. Lipic, and S. Grgic, “Fine-tuning Convolutional Neural Networks for fine art classification,” *Expert Systems with Applications*, vol. 114, pp. 107–118, Dec. 2018, doi: 10.1016/j.eswa.2018.07.026.

[11] N. Gonthier, Y. Gousseau, S. Ladjal, and O. Bonfait, “Weakly Supervised Object Detection in Artworks,” Oct. 2018, doi: 10.1007/978-3-030-11012-3_53.

[12] “https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4.”

[13] P. Madhu, R. Kosti, L. Mührenberg, P. Bell, A. Maier, and V. Christlein, “Recognizing characters in art history using deep learning,” in *SUMAC 2019 - Proceedings of the 1st Workshop on Structuring and Understanding of Multimedia heritAge Contents, co-located with MM 2019*, Oct. 2019, pp. 15–22. doi: 10.1145/3347317.3357242.

[14] P. Madhu, R. Kosti, L. Mührenberg, P. Bell, A. Maier, and V. Christlein, “Recognizing characters in art history using deep learning,” in *SUMAC 2019 - Proceedings of the 1st Workshop on Structuring and Understanding of Multimedia heritAge Contents, co-located with MM 2019*, Oct. 2019, pp. 15–22. doi: 10.1145/3347317.3357242.

[15] C. B. Madsen, “An Explainable Deep Learning Baseline for Iconography Research in Artworks.” [Online]. Available: https://github.com/christophermadsen/iconography_dl_baseline

[16] “http://cnnlocalization.csail.mit.edu/.”

[17] “https://medium.com/analytics-vidhya/introduction-to-object-detection-with-rcnn-family-models-310558ce2033.”

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” Jun. 2015, [Online]. Available: http://arxiv.org/abs/1506.02640

[19] “J. Browniee, ‘A gentle introduction to object recognition with deep learning,’ 22 05 2019. [Online]. Available: https://machinelearningmastery.com/objectrecognition-with-deep-learning.”

[20] P. Felzenszwalb, D. McAllester, and D. Ramanan, “A discriminatively trained, multiscale, deformable part model,” in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2008, pp. 1–8. doi: 10.1109/CVPR.2008.4587597.

[21] “P. Sharma, ‘A step-by-step introduction to the basic object detection algorithms (part 1),’ 11 10 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/10/a-step-by-stepintroduction-to-the-basic-object-detection-algorithms-part-1.”

[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Sep. 2014, pp. 580–587. doi: 10.1109/CVPR.2014.81.

[23] R. Girshick, “Fast R-CNN,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1440–1448. doi: 10.1109/ICCV.2015.169.

[24] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[25] “S. K, ‘Region Proposal Network — A detailed view,’ 21 12 2019. [Online]. Available: https://towardsdatascience.com/region-proposal-network-a-detailed-view-1305c7875853.”

[26] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” Dec. 2016, [Online]. Available: http://arxiv.org/abs/1612.08242

[27] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” Apr. 2018.

[28] “A. Kathuria, ‘ What’s new in yolo v3? [Online],’ 2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.”

[29] “A. Kathuria, ‘ What’s new in yolo v3? [Online],’ 2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.”

[30] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” Apr. 2020, [Online]. Available: http://arxiv.org/abs/2004.10934

[31] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” Dec. 2019, [Online]. Available: http://arxiv.org/abs/1912.01703

[32] “J. Solawetz, ‘Yolov5 new version - improvements and evaluation,’ 2020. [Online]. Available: https://blog.roboflow.com/yolov5-improvements-andevaluation.”

[33] "[13] J. Solawetz, 'Yolov5 new version - improvements and evaluation,' 2020. [Online]. Available: https://blog.roboflow.com/yolov5-improvements-andevaluation."

[34] "roboflow.com."

[35] "https://www.tensorflow.org/tutorials/load_data/tfrecord."

[36] "https://pytorch.org/hub/ultralytics_yolov5/."

[37] "https://blog.roboflow.com/when-to-use-contrast-as-a-preprocessing-step/."

[38] "https://blog.roboflow.com/tag/augmentation/."

[39] "https://blog.roboflow.com/why-to-add-noise-to-images-for-machine-learning/."

[40] S. Dodge and L. Karam, "Understanding How Image Quality Affects Deep Neural Networks," Apr. 2016, [Online]. Available: http://arxiv.org/abs/1604.04004

[41] "https://colab.research.google.com/?utm_source=scs-index."

[42] "https://docs.python.org/3/reference/."

[43] "https://github.com/ultralytics/yolov5."

[44]. "Kathuria, 'How to train yolo v5 on a custom dataset,' 2021. [Online]. Available: https://blog.paperspace.com/train-yolov5-custom-data."

[45] "https://github.com/ultralytics/yolov5."

[46] "https://github.com/ultralytics/yolov5#pretrained-checkpoints."

[47] "https://github.com/ultralytics/yolov5/issues/5863."

[48] "https://github.com/xiaowk5516/yolov5/tree/grad-cam."

[49] "@misc{tensorflowmodelgarden2020, author = {Hongkun Yu, Chen Chen, Xianzhi Du, Yeqing Li, Abdullah Rashwan, Le Hou, Pengchong Jin, Fan Yang, Frederick Liu, Jaeyoun Kim, and Jing Li}, title = {{TensorFlow Model Garden}}, howpublished = {\url{https://github.com/tensorflow/models}}, year = {2020} }."

[50] "https://keras.io/api/."

[51] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017, [Online]. Available: http://arxiv.org/abs/1704.04861

[52] "https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359."

[53] "https://blog.roboflow.com/training-a-tensorflow-faster-r-cnn-object-detection-model-on-your-own-dataset/#:~:text=Faster%20R-CNN%20is%20one%20of%20the%20many%20model,case.%20Take%20advantage%20of%20the%20TensorFlow%20model%20zoo."

[54] "https://towardsdatascience.com/yolov5-compared-to-faster-rcnn-who-wins-a771cd6c9fb4."

[55] “https://medium.com/@linda0511ny/tensorflow-train-dataset-by-epochs-or-steps-3839705f307d.”

[56] “https://dengking.github.io/machine-learning/Theory/Deep-learning/Guide/Batch-epoch-step/Steps-VS-epochs/.”

[57] R. Tahir, “Road Traffic Signs Detection using YOLOv5.”

**Notable and Honorary mentions:**

- Respectable Prof. Yann LeCunn (https://scholar.google.com/citations?user=WLN3QrAAAAAJ)
- An Explainable DL based Iconography [15]
- Road Traffic Sign Detection using YOLOv5 [58]

# Appendix A: Results of comparison between YOLOv5s and YOLOv5x

Initially both YOLOv5x and YOLOv5x were trained for 150 epochs, which resulted that the model started to overfit after 25 epochs. Thus, for accuracy and inference, in sec. 4.1, the model YOLOv5s is trained only for 25 epochs at batch size of 32.

Below Fig. no. 24 shows a brief summary of YOLOv5s and YOLOv5x models that were trained for 150 epochs. Even with a considerable smaller size, YOLOv5s performs almost at par with YOLOv5x for mAP@0.5. However, when it comes to mAP@0.5:0.95, the bigger YOLOv5x shows considerable improvement in results for both classes. Moreover, the computational time used by both models differed considerably too with smaller version taking only 31 minutes to train, while biggest version (YOLOv5x) took almost 6 hours and 54 minutes. Moreover, considering Precision and Recall, these metric were definitely better when it comes to YOLOv5x and especially precision for Holo class which was just 66% with the smaller YOLOv5s but 80% with YOLOv5x

```
YOLOv5s summary: 213 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 4/4 [00:01<00:00,  2.11it/s]
                 all        100        203      0.797      0.854      0.877      0.535
                Holo        100        105      0.666        0.8      0.795       0.49
                Mary        100         98      0.927      0.908      0.959       0.58
```

```
YOLOv5x summary: 444 layers, 86180143 parameters, 0 gradients, 204.0 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 4/4 [00:06<00:00,  1.71s/it]
                 all        100        203      0.899      0.883      0.899      0.666
                Holo        100        105      0.808        0.8      0.824      0.621
                Mary        100         98       0.99      0.967      0.975       0.71
```

**Fig.24** Summary of YOLOv5s (1st image) and YOLOv5x (1ns image) models showing Precision, Recall,

mAP@0.5 and mAP@0.5:0.95

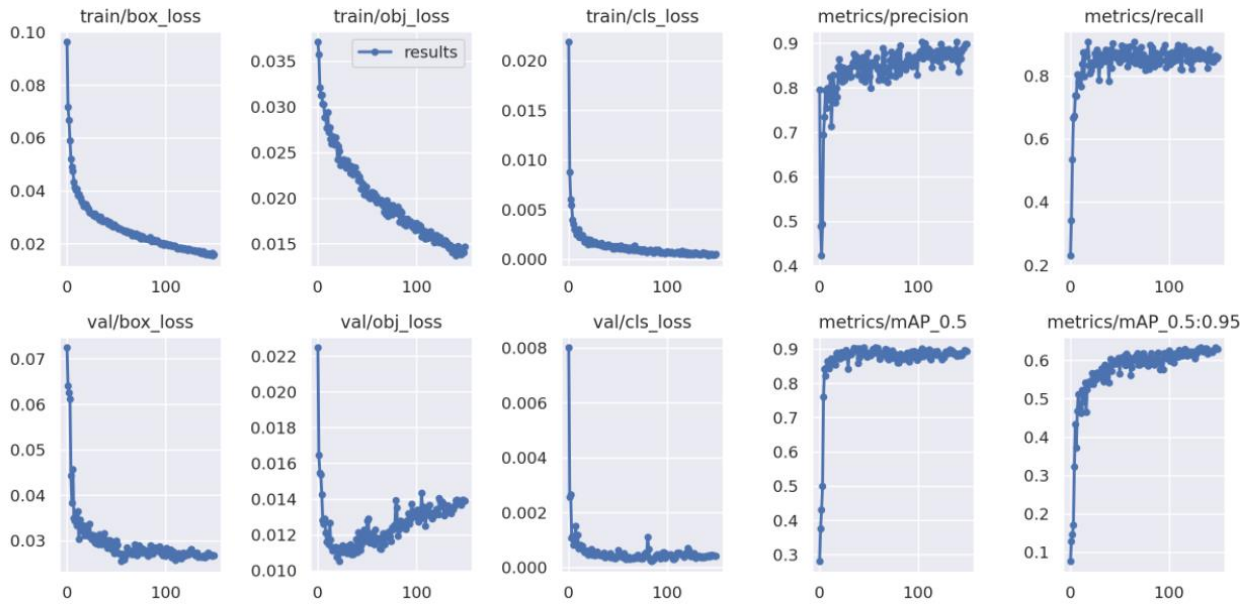



**Fig.25** Plots of the metrics for the trained YOLO v5s

Fig. 24 shows plots for YOLOv5s trained for 150 epochs. Considering second plot from left on both upper and lower row, it can be seen that val/obj_loss, which is the loss of objectness, starts to rise after about 25 epochs. At the same time, train/obj_loss continues to decrease showing that the model has started to overfit the data. Therefore, there was no reason to train the model beyond 25 epoch

Considering the two plots of mAP (lower row, first and second from right), it is also noted that the model had almost converge after about 25 epochs. After that, the mAP almost remained same especially for IoU 0.5

# Appendix B: Statutory Declaration

## Statement of Authorship / Selbstständigkeitserklärung

I hereby declare that I have completed this master's thesis independently and exclusively using the specified literature and resources. The aids taken directly or indirectly from external sources are marked as such. The work has not been submitted to any other examination authority.

Magdeburg, 31.03.2022 Signature