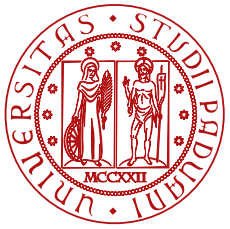

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

An Empirical Study of Object Detection Methods with Deep Ensemble and Stochastic Selection of Activation Functions

Master Candidate

M Aqib Ismail

Student ID 2043892

Supervisor

Prof. Loris Nanni

University of Padova

Academic Year
2023/2024

To my parents
and friends

Abstract

The task of object detection is one of the challenging problems in computer vision. Over the recent years, as deep learning has rapidly evolved, researchers have dedicated considerable efforts to experimenting and contributing to improving object detection performance and its associated tasks, including object classification, localization, and image segmentation. Generally, the performance of any object detector is evaluated through detection accuracy and inference time. The introduction of YOLO (You Only Look Once) and its architectural successors have notably improved detection accuracy. The presented approach suggested changing the backbone of Yolov4 and Yolov3 by replacing them with a custom ResNet50 convolutional neural network. The architecture of the ResNet50 is changed to design a new model by replacing each activation layer of a ResNet50 which is usually a ReLU layer with a different varients of ReLU AF stochastically drawn from a set of activation functions. The goal of this project is to evaluate the performance of modified CNN with Yolo's base CNN network and to evaluate the performance of ensemble methods.

Contents

List of Figures

List of Tables

1 Introduction

Object detection is an important problem involving the identification of object instances within an image and their classification into specific classes, such as humans, animals, or cars. It addresses the question "What objects are present here?" Typically, object detection can be categorized into two groups: general object detection and detection applications. In the former, the objective is to explore methods to identify various object types using a unified framework to emulate human vision and cognition. In the latter, the focus is on recognizing objects of a particular class within specific application scenarios for example pedestrian detection, face detection, or text detection. Object detection models can be classified into two macro-categories: two-stage and one-stage detectors. [56][11].

In the initial phase, Regions of Interest (RoI) are generated by using a Region Proposal Network (RPN). This stage primarily focuses on selecting viable region proposals, employing techniques like negative proposal sampling. However, the second stage predicts the objects and bounding boxes corresponding to the proposed regions. The popular models falling into this category include Region-based Convolutional Neural Networks (RCNN) [28][70], Fast RCNN [27], and Faster RCNN [68]. Single-stage object detectors are explicitly tailored for conducting object detection in a single stage, considering all region proposals. These detectors yield output comprising bounding boxes and class-specific probabilities for the underlying objects, capturing the spatial dimensions of an image in one shot.

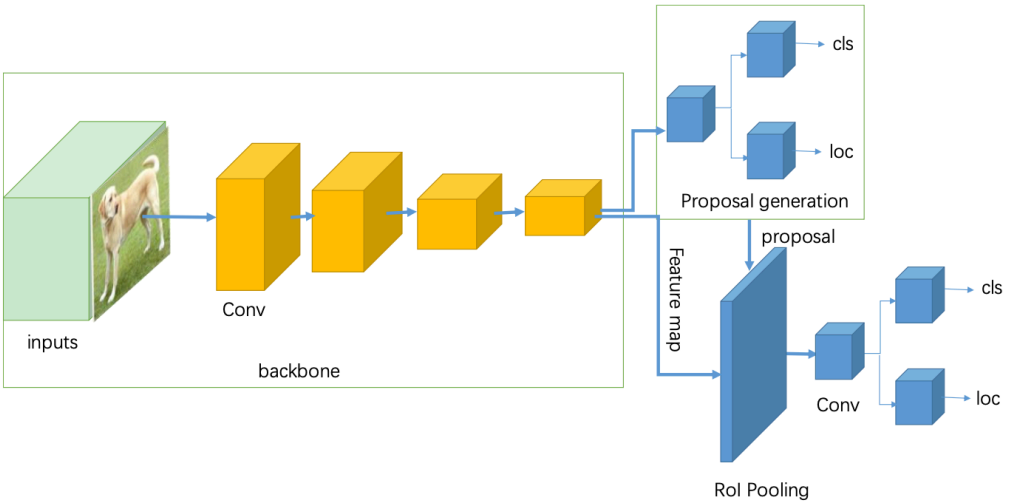

Figure 1.1: Illustrates the two-stage object detectors and an incremental improvement in the architecture.

During the initial phase of the region proposal, several key algorithms, including Deformable Parts Models (DPM) [22], OverFeat [71], and Edge Boxes [95], adopted the sliding window technique. In this approach, a fixed-sized window traverses the entire image, generating region proposals after passing through a classifier. This procedure is iterated with progressively larger window sizes. In contrast, R-CNN and its successors leverage a selective search algorithm to derive region proposals. R-CNN, standing for region-based convolutional neural network, represents an object detection algorithm introduced by Ross Girshick [27].

In the R-CNN, many region proposals for example around 2000 are initially extracted from the input image, and labeled with their corresponding classes and bounding boxes. Subsequently, a Convolutional Neural Network (CNN) is employed to execute forward propagation on each region proposal, extracting its distinctive features. The features extracted from each region proposal are utilized to predict both its class and bounding box. Notably, a significant bottleneck in R-CNN's performance stems from the independent CNN forward propagation for each region proposal, lacking shared computation. This often results in redundant computations due to the overlapping nature of these re-

gions. A key enhancement introduced in Fast R-CNN, compared to R-CNN, is the optimization achieved by conducting CNN forward propagation solely on the entire image. The R-CNN comprises of the following four steps:

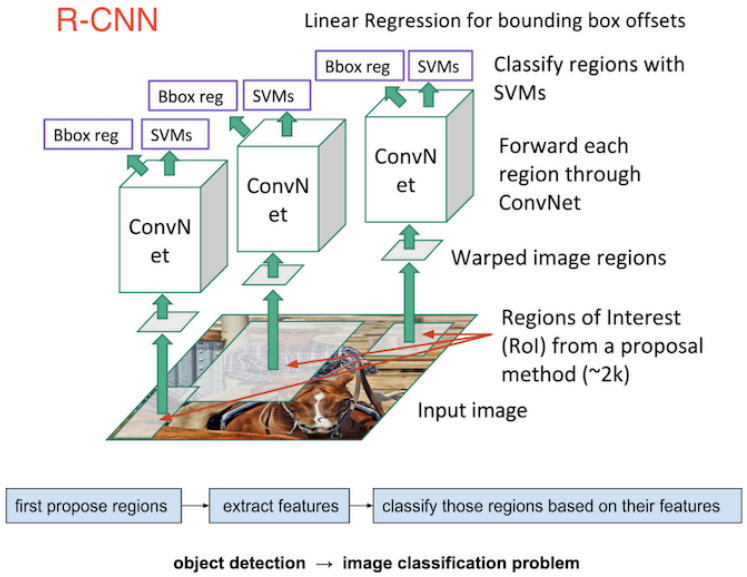

Figure 1.2: R-CNN architecture

1. Perform a selective search [85] to extract multiple high-quality region proposals on the input image. Each region proposal are selected at multiple scales with different shapes, sizes and will be labeled with a class and a ground-truth bounding box.
2. Choose a pre-trained CNN and truncate it before the output layer. Resize each region proposal to the input size required by the network, and output the extracted features for the region proposal through forward propagation.
3. Take the extracted features and labeled class of each region proposal. Train multiple SVMs to classify objects, where each SVM individually determines whether the example contains a specific class.
4. Take the extracted features and label the bounding box of each region proposal. Train a linear regression model to predict the ground-truth bounding box.

Despite the effectiveness of using pre-trained Convolutional Neural Networks (CNNs) to extract image features in the R-CNN model, its speed remains a significant drawback. The necessity to evaluate thousands of region proposals

from a single input image results in a substantial computational load, rendering widespread adoption of R-CNNs impractical in real-world applications.

Fast R-CNN [27] marks a significant advancement in both model training and inference times, accompanied by an enhancement in object detection performance, as measured by metrics like mAP. In single-stage object detection, a multi-task loss function is instrumental, allowing for the update of all network layers during model training without the need for specific disk storage to cache features. Instead of extracting CNN feature vectors independently for each region proposal, this model consolidates them in a single CNN forward pass over the entire image, and the region proposals share this feature matrix. A crucial modification involves substituting the pre-trained CNN's maximum pooling layer with a Region of Interest (RoI) pooling layer. This RoI pooling layer produces fixed-length feature vectors for each region proposal, applied to the output of a selected internal layer of the CNN. Each feature vector is then fed into a fully connected layer equipped with a SoftMax activation function, facilitating the output of class probabilities and bounding box offsets. This innovative approach contributes to the efficiency and effectiveness of Fast R-CNN.

Faster R-CNN [68], introduced in early 2016 as a successor to Fast R-CNN, represents a notable evolution in object detection methodologies. The Fast R-CNN model typically relies on generating numerous region proposals through selective search. To address the challenge of maintaining accuracy while reducing the number of region proposals, Faster R-CNN introduces a pivotal change. Instead of utilizing selective search, It suggests the integration of a Region Proposal Network (RPN) this modification aims to enhance the efficiency of object detection without compromising accuracy. It has two modules;

1. The first is a CNN known as the Region Proposal Network (RPN) Its primary role is to generate region proposals by taking a single image as input and producing bounding boxes along with object confidence scores as outputs.

2. In the training phase, the RPN undergoes training on the ImageNet dataset. The generated RP are then utilized for both detection and separate training processes. Finally, Fast R-CNN is fine-tuned, incorporating unique dense layers to refine its performance.

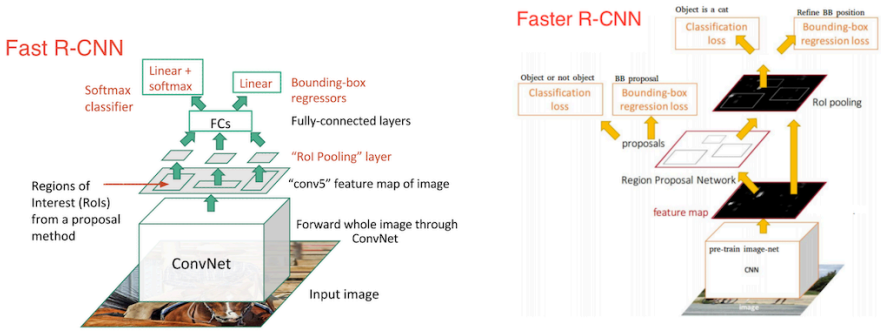

Figure 1.3: Fast-RCNN & Faster-RCNN architecture

The comparison between two-stage and single-stage object detectors reveals that two-stage detectors generally outperform their single-stage counterparts in terms of accuracy. Although they excel in achieving high accuracy by focusing on highly probable regions for object detection, they tend to be slower. In our thesis, we center our study around first-stage detectors, specifically focusing on You Only Look Once (YOLO) [39]. J. Redmon et al. [39] introduce an innovative solution to object detection by consolidating various components into a single network. This approach compels the network to analyze the entire image simultaneously, as opposed to specific regions enables a more comprehensive understanding of the environment, facilitating the localization of different classes of objects.

Additionally, YOLO establishes an implicit connection between closely related classes of objects, a feature that distinguishes it from existing object detection models. Notably, YOLO stands out for its remarkable speed compared to its predecessors. This efficiency is primarily attributed to YOLO's unique approach of not dividing the recognition process into multiple stages. Instead, it predicts bounding boxes, probabilities, and classes of objects in a single phase for the input image. While YOLO may incur more localization errors compared to some other object detection systems, it exhibits a distinct advantage in its reduced likelihood of recognizing false positives in the background of the image and considerably faster.

YOLO is not the first algorithm to employ a Single Shot Detector (SSD) for object detection. Several other algorithms introduced in recent times also adopt this approach, including Single Shot Detector (SSD) [55], Deconvolution Single Shot

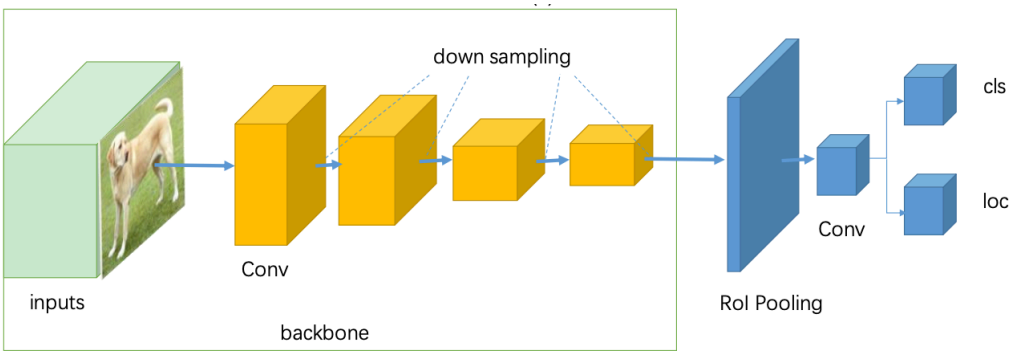

Figure 1.4: The generic schematic architecture of single-stage object detectors.

Detector (DSSD) [25], RetinaNet [51], M2Det [15][93], RefineDet++ [92]. These algorithms are all based on single-stage object detection. In contrast, two-stage detectors are known for their complexity and robustness, often outperforming single-stage detectors. Despite the inherent advantages of two-stage detectors, YOLO stands out by presenting a formidable challenge not only to two-stage detectors but also to previous single-stage detectors in terms of both accuracy and inference time.

Deep neural networks have gained immense popularity owing to their ability to achieve state-of-the-art performance across various critical applications, including image classification, image segmentation, language processing, and computer vision. These networks typically consist of linear components whose parameters are learned to fit the data, alongside nonlinearities specified in the form of activation functions such as sigmoid, tanh, rectified linear units (ReLU) [29], or max-pooling functions. The inclusion of nonlinear activation functions at each neuron is crucial, providing the network with the capability to approximate arbitrarily complex functions [89]. The choice of activation function significantly impacts both the training speed and the overall accuracy of the network. Ongoing research is actively focused on designing new activation functions that can enhance training speed and network accuracy [23][16]. In recent times, the widely used sigmoid and hyperbolic tangent activation functions have been replaced by Rectified Linear Units (ReLU) in training deep networks. This shift reflects the continuous effort in the field to optimize neural network architectures and improve their efficiency in various applications.

ReLU, a piece-wise linear function equivalent to the identity for positive inputs and zero for negative ones, has gained popularity due to its good performance, speed, effectiveness, and simplicity. In this thesis, an extensive study is carried out on several alternatives to the standard ReLu function. One well-known alternative is Leaky ReLU [62], an activation function that mirrors ReLU for positive inputs but introduces a small slope $\alpha > 0$ for negative inputs. Another option is ELU [16], which exponentially decreases to a limit point in the negative space. SELU [44], a scaled version of ELU by a constant λ, is also considered. In addition to these "fixed" activation functions, several "learnable" activation functions are explored. Parametric ReLU (PReLU) [33] is a variation of Leaky ReLU where the amount of is learned during training. Adaptive Piece-wise Linear Unit (APLU) [23] is a piece-wise linear activation with learnable parameters. Swish, a high-performing function, is a combination of a sigmoid function and a trainable parameter. These alternatives represent a diverse set of activation functions that are investigated for their impact on training and performance in this research and are represented in a separate chapter of this thesis.

In this thesis, several pre-trained ResNet50 [61] CNN models were modified by combining different varients of ReLU activation functions at distinct levels of the network graph. To achieve this, a method for stochastic selection of activation functions is implemented to replace each ReLU layer. These ResNet50 were then fine-tuned and trained on object classification tasks in our case it was the classification of shark objects with non-shark objects. After completing the training process, the newly trained models were substituted within the backbone architecture of object detection models, YOLOV4 [3] and YOLOv3 [66]. These object detection models are designed to locate shark subjects in images by using a transfer learning technique, predict bounding boxes around them, and assign object confidence scores to each bounding box. Following the development of the proposed solution, a comprehensive empirical evaluation was conducted, and the approach was subsequently compared to YOLO's base network which is darknet53.

The remainder of this thesis is organized as follows: Section 2, includes an in-depth overview of the prior studies and research carried out in the field of computer vision, with a specific focus on object detection and what will be the impact of changing CNN model architecture. In Section 3, we elaborate on

the techniques used in this research, such as the topologies, activation functions, and data augmentation methods. The results are in Section 4, which that demonstrates our best ensemble approach outperforms other methods. Lastly, in Section 5, we present our conclusions and suggestions for future work.

2

Related Work

2.1 Shark Detector Pipeline

Sharks play a crucial role as indicators of the ocean environment's health. However, they confront persistent challenges arising from escalating fishing pressures and inadequate management and conservation practices. These issues are largely attributed to limited data, insufficient taxonomic knowledge, and underdeveloped monitoring methods [38]. The collection of observation data on sharks through surveys and fisheries monitoring is often expensive or logistically challenging, particularly when dealing with species that have larger home ranges [7]. Sharks continue to be a highly data-deficient group of marine animals, and these information gaps contribute to the absence of abundance and distribution indices, as well as taxonomic precision, necessary for a proper assessment of population statistics [7][38].

Image-based biomonitoring presents a transformative and cost-effective alternative method in ecological surveys conducted in marine and terrestrial environments [74][86][87]. Significant advancements in technologies such as baited underwater remote videos (BRUVs), motion-activated camera traps, and crowd-sourced citizen science media have led to the production of ecological information at an unprecedented rate [30][86][87]. Notably, remote monitoring methods, including these advanced technologies, play a crucial role in generating visual media to address information gaps related to sharks. These methods are non-invasive and prove useful in minimizing sampling effort. However, they

generate large volumes of media that require post-processing for species identification and analyses, including the removal of irrelevant images [81]. Studies such as [82] and [63] emphasize the importance of utilizing deep learning methods to filter out unrelated content and facilitate rapid sampling in the context of image-based biomonitoring.

Deep learning algorithms prove to be highly flexible and well-suited for addressing a variety of tasks in the context of marine and ecological studies [48][63][74]. They have demonstrated effectiveness in tasks such as estimating fish sizes from images [4][26], identifying discarded and processed fish [24], and classifying acoustic and movement data [9][21][40]. However, the application of machine-learned detection and image classification specifically for shark species is infrequently studied, primarily due to a shortage of sufficient training data. Video and photographic documentation of sharks are seldom obtained from commercial fisheries.

In a previous study by Jenrette and his team [36] on shark detection and classification, a hierarchical approach using locators and classifiers achieved approximately 70% species classification accuracy. The methodology involved a shark detection and classification pipeline with multiple steps, consisting of three main components:

- An object-detection model called the Shark Locator (SL), which locates one or several shark subjects in images and draws bounding boxes around them;
- A binary sorting model called Shark Identifier (SI) sorts images of sharks from a pool of heterogeneous images;
- Multi-class models called Shark Classifiers (SCs) which classify shark images to the genus and species levels.

By integrating these three modeling components, they developed a comprehensive shark identification and classification pipeline known as Shark Detector (SD). This pipeline is designed to process various media containing shark subjects, effectively locate and organize these subjects based on relevance, and ultimately classify the sharks down to the species level. For the development of these models, shark training images were primarily sourced from sharkPulse [36]. The detailed description of all these three components are:

The first component is the Shark Locator (SL), focusing on object detection. To construct the SL, they utilized TensorFlow's Model Garden and implemented a Faster Region-based Convolutional Neural Network (Faster-RCNN) algorithm. The model was trained using the Common Objects in Context (COCO) datasets, which included 236 shark images, enabling the detection and draw bounding boxes around shark subjects [52][69]. The SL significantly augmented the classification dataset, increasing it from 24,546 images to 53,345 images.

The second component, the Shark Identifier (SI), is a binary model developed for sorting purposes. The SI is designed to distinguish between shark and non-shark subjects in images, serving as a filter to exclude non-shark images before the remaining images undergo taxonomic classification. A total of 53,345 shark images were sourced from Instagram and sharkPulse, while an additional 50,260 non-shark images were obtained from Instagram.

The SI was constructed to learn key shark features from training images, optimizing the binary cross-entropy loss function [47]. Transfer learning was implemented by incorporating a pre-trained model to reduce the number of training steps. Specifically, the SI was pre-trained with the VGG16 network, which has been trained on 1.28 million images across 1000 categories and achieves a 92.7% test accuracy on the ImageNet dataset [78]. Convolutional-pooling layers were constructed to serve as checkpoints for summarizing features learned by the model [47]. When shark features are learned from trained images, the Convolutional Neural Network (CNN) generates parameters called weights, initially initialized from pre-trained networks on the ImageNet dataset. The bottom pre-trained layers were frozen to prevent weight modifications while training the top layers specifically for shark features. VGG16, containing 16 pre-trained layers, had four additional layers added for training shark features.

The final segment of their pipeline involves the development of a Shark Classifier (SC) designed for genus and species classification (GSC). This component operates as a hierarchical classification framework for taxonomically classifying the identified shark images. The researchers trained one genus-specific model and a set of local species-specific models, each tailored for a particular genus. The SC takes in the filtered shark images and conducts classification at the genus level using the genus-specific classifier (GSC). Subsequently, based on

the identified genus, a species-specific classifier (SSCg) predicts the most likely species.

The SC was trained using the sharkPulse database which incorporates information from 74 genera and 219 species of sharks, with an average of 167 images per species. The Genus-Specific Classifier (GSC) was trained with 36,722 images, while the Species-Specific Classifiers (SSCg) were trained with 19,243 images. To assess the performance of the SC, the researchers evaluated the recall of a genus class concerning its training data quantity. The results indicated an average of 433 ± 47 images were needed to achieve >50% recall. Recall measures the proportion of shark images that were correctly classified. However, for most genera (>64%), the available images did not meet this threshold for adequate training quality. Following the classification of a genus, a similar evaluation was conducted for species classes, revealing an average of 161 ± 41 images were needed to achieve >50% recall. These averages were subsequently employed as training data quantity thresholds for the SC.

2.2 Performance of different activation functions across image classification and image segmentation problem

In the referenced paper [47], the authors conduct a comprehensive empirical comparison of various activation functions across a range of image classification tasks and an image segmentation problem. They initiate this comparison using two of the best-performing models: ResNet50 [34] for the classification task and DeepLab-v3 [12] for the segmentation task. The focus of their investigation includes assessing different approaches for replacing ReLU layers and exploring various methods for constructing ensembles of Convolutional Neural Networks (CNNs) by varying the activation function layers.

The proposed ensemble framework is assessed in two distinct applications: image classification and image segmentation. In the realm of image classification, the framework is applied to various medical problems, encompassing 13 image classification datasets in the benchmark. CNNs have demonstrated high

performance in medical datasets, addressing issues such as keratinocyte carcinomas and malignant melanomas detection [19], thyroid nodules classification from ultrasound images [13], and breast cancer recognition [61]. The testing protocol involves fine-tuning each model on every dataset, followed by a thorough evaluation and comparison. The experiments reveal that the proposed method performs effectively across all tested problems, achieving state-of-the-art classification performance [59].

In the domain of image segmentation, the framework is applied to address the skin segmentation problem precisely, distinguishing between skin and non-skin regions in a digital image. This task holds significance across various applications, including face detection [35], body tracking [5], gesture recognition [32], and objectionable content filtering [49]. Skin detection is also highly relevant in the medical field, where it serves as a component in applications such as face detection and body tracking. In their experiment, the authors conduct a comparison of several approaches by training on a small dataset comprising only 2000 labeled images. Testing is then performed on 11 different datasets including images from diverse applications. The reported results demonstrate that the proposed method achieves state-of-the-art performance [57] across most benchmark datasets, even without ad-hoc tuning.

In the experiments, the authors choose two top-performing models: ResNet50 [34] for image classification and Deeplabv3+ [12] for segmentation. In the image classification experiments, all the models are fine-tuned on the training set of each classification problem. For the task of image segmentation, the authors opt for DeepLabv3+ [12], a recent architecture that employs atrous convolution. Unlike traditional convolutions, atrous convolution skips certain adjacent pixels in a spaced-out lattice fashion. DeepLabv3+ incorporates four parallel atrous convolutions, each with varying atrous rates, followed by a "Pyramid Pooling" method. Given its encoder-decoder structure, DeepLabv3+ can leverage a powerful pre-trained CNN architecture, and in this instance, the authors choose ResNet50. It's noteworthy that internal evaluations indicate that ResNet101 and ResNet34 exhibit similar performance for this task.

3 Methods

3.1 Topologies

3.1.1 ResNet50

In this study, we explored the ResNet50 [76] model, which is a convolutional neural network (CNN) an architecture belonging to the Residual Network family, also known for its skip or shortcut connections. ResNet, introduced by Kaiming He et al. in 2015 [76], introduced a key innovation to address the issue of vanishing gradients in deep neural networks. ResNet-50 is designed to take an input image of size 224 x 224 x 3, where 3 represents the RGB channels. The initial layers of ResNet-50 include standard convolutional layers responsible for feature extraction from the input image. The fundamental building blocks of ResNet are residual blocks [34]. Each residual block comprises two paths: a "shortcut" path and a "main" path. The main path incorporates several convolutional layers, batch normalization [72], and activation functions [73]. Simultaneously, the shortcut path establishes a direct connection from the input to the output, effectively bypassing the main path. This shortcut mechanism proves crucial in mitigating the vanishing gradient problem. The output of a residual block is obtained through the element-wise sum of the shortcut and the output of the main path, facilitating more effective learning in deep neural networks.

ResNet-50 consists of several stacks of residual blocks. Some of these blocks are identity blocks (shortcuts have no convolutions), and others are convolu-

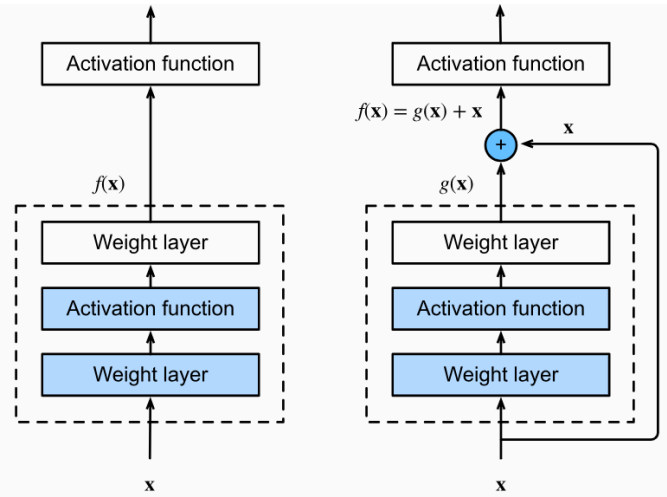

Figure 3.1: The left part of this figure is a classic block of two convolutional layers and an activation layer. On the right, a residual connection: the block input is added to the block output if the block output is zero then we have zero plus x, so f(x) is equal to x.

tional blocks (shortcuts include a convolution to match dimensions). After the stacks of residual blocks, global average pooling is applied. This operation reduces the spatial dimensions to a 1x1 size for each feature map, resulting in a vector that represents the entire image. A fully connected layer is added to the output of the global average pooling to map the features to the desired number of classes. The final layer usually includes a softmax activation function to obtain class probabilities. ResNet-50, in particular, has 50 layers, including weight layers and batch normalization layers. The use of residual connections allows ResNet architectures to be very deep without suffering from degradation issues, making them effective for image classification tasks, among others.

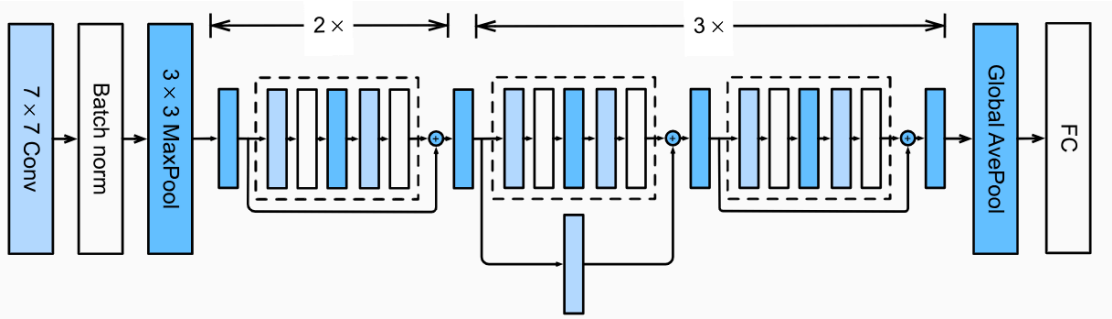

Figure 3.2: The ResNet-18 architecture.

3.1.2 YOLO

YOLO (You Only Look Once) is a family of real-time object detection algorithms popular for their speed and accuracy. The core concept of YOLO involves partitioning the input image into a grid and generating predictions for each grid cell. Unlike conventional object detection approaches that follow region proposal and sequential classification steps, YOLO executes both tasks concurrently in a single pass through the neural network. To grasp the YOLO algorithm, it's crucial to comprehend the nature of the predictions. The ultimate goal is to predict the object's class and the bounding box that precisely outlines the object's location. These bounding boxes are characterized by four descriptors:

- center of a bounding box (b_x, b_y),
- width (b_w),
- height (b_h)

and the value **c** corresponds to a class of an object (e.g., the shark in our case). In addition, we have to predict the p_c value, which is the probability that there is an object in the bounding box y = (p_c,b_x,b_y,b_w,b_h,c).

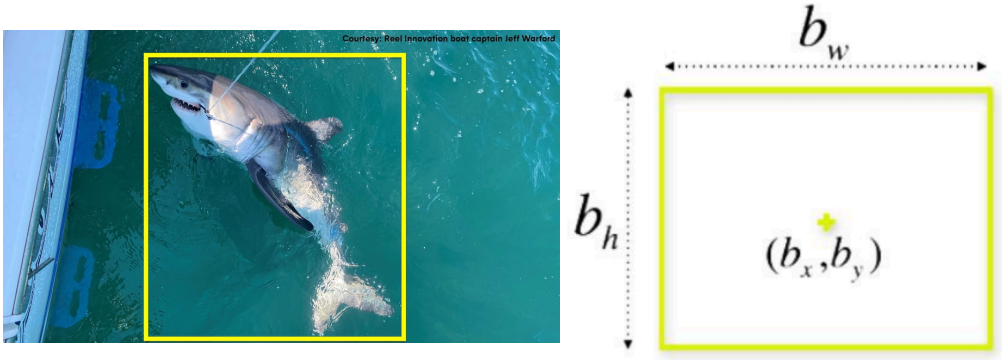

Figure 3.3: Description of bounding box.

As we mentioned earlier, when working with the YOLO algorithm we are not searching for interesting regions in our image that could potentially contain an object. The final level of the network predicts both class probabilities and bounding box coordinates for recognized objects.

The YOLO architect integrates the various elements of object detection into a unified neural network. Utilizing features from the entire image, the network predicts each bounding box, simultaneously forecasting all bounding boxes across all classes for a given image. This global reasoning approach considers the complete image and all its objects collectively. The architecture partitions the input image into an S x S grid, where each grid cell becomes responsible for detecting an object if the object's center falls within that cell. Every grid cell predicts B bounding boxes and assigns a confidence score to each, indicating the likelihood of containing an object (Pr(Object)), along with the accuracy of the bounding box's size and location concerning the object (IOU intersection over union).

For every predicted bounding box, the model provides five values: x, y, w, h, and the confidence score "Pr(Object) $*$ IOU" Here, (x, y) denote the coordinates of the bounding box center relative to the edges of the grid cell, while w and h represent the width and height of the bounding box relative to the entire image. Each grid cell, predicts C classes along with their conditional probabilities $Pr(\text{Class}_i|\text{Object})$. Subsequently, the model calculates, for each bounding box, the product of the conditional probabilities of the classes (of the cell) with the confidence score of the bounding box, resulting in a specific confidence score for each class in each bounding box:

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \quad (3.1)$$

The value obtained contains both the probabilities of an object belonging to a specific class within the bounding box and the precision with which the predicted bounding box defines the spatial boundaries of the object.

Architecture of YOLO

The authors implemented the model as a CNN and evaluated it on the PASCAL VOC [20] detection dataset. In the architecture, the initial convolutional layers are responsible for feature extraction, and the fully connected layers predict the output probabilities and coordinates. The network architecture draws inspiration from the GoogleNet model [36], featuring 24 convolutional layers

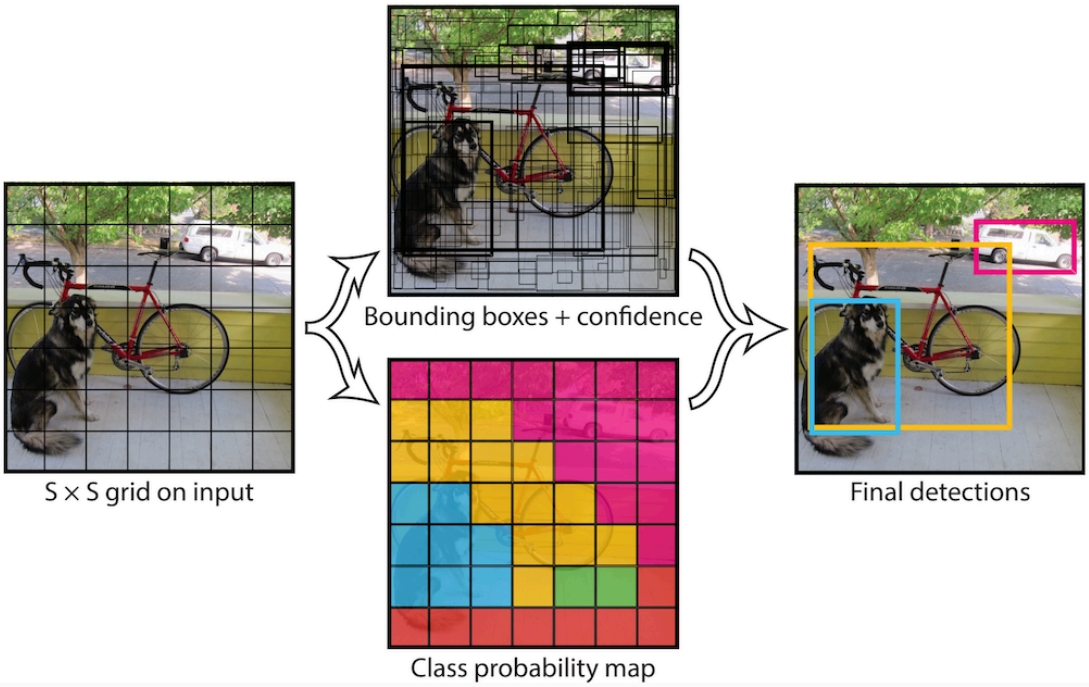

Figure 3.4: The graphical representation of YOLO's workflow.

and 2 fully connected layers. Instead of employing the inception modules of GoogLeNet, the model utilizes 1 x 1 reduction layers followed by 3 x 3 convolutional layers, similar to [62]. The complete network structure is depicted in Figure 3.5.

YOLO employs a linear activation function for the last layer where backpropagation is not feasible due to the constant derivative. For all other layers, the following activation function is utilized:

$$\Phi(x) = \begin{cases} x & \textbf{if } x > 0 \\ 0.1x & \textbf{otherwise} \end{cases} \tag{3.2}$$

The convolutional layers are initially pre-trained on the ImageNet classification task with a resolution of 224 x 224 for the input image. Later, for detection, the resolution is doubled. In YOLO, the output layer has a size of 7 x 7 x 30, where 7 is fixed in the original YOLO. The image is divided into a 7 x 7 grid, and for each grid cell, the model predicts 2 bounding boxes, confidence scores for those boxes, and probabilities for 20 classes (C = 20). This results in predictions

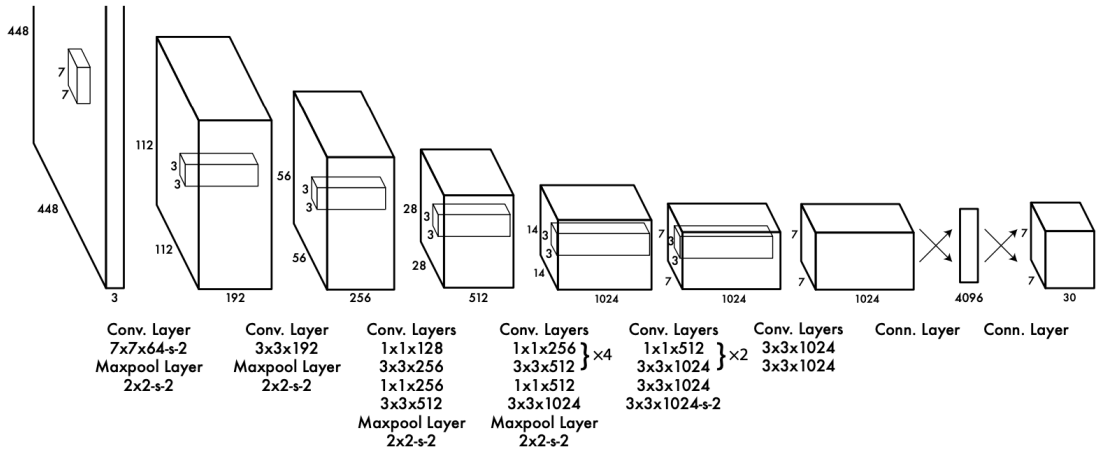

Figure 3.5: The architecture of the detection network has 24 convolutional layers followed by 2 fully connected layers.

encoded as an S x S x (B x 5 + C) tensor, where B is set to 2 for this example. The training dataset used is the PASCAL VOC, containing 20 labeled classes.

Training of YOLO

According to the authors, The YOLO training process consists of two phases: an initial pre-training phase, where only the deepest levels of the network are trained, and a subsequent phase that involves training the entire network.

During the pre-training phase, the authors of YOLO initiate the training by focusing on the first 20 convolutional layers. In addition to these layers, they include an average pooling level and a fully connected layer. The training utilizes the ImageNet dataset, which involves a classification problem with input images sized at 224x224 pixels. During the training phase, they eliminated the last two levels of the network, added specifically for the first part of the training, to added another 4 convolutional levels, finally followed by 2 fully connected levels.

For the second training phase, crucial for enhancing accuracy in object detection, the authors opt for high-resolution images (448x448 pixels). This phase spans 135 epochs and employs both the training and validation sets from the PASCAL VOC 2007 and PASCAL VOC 2012 datasets. To mitigate overfitting, a dropout layer with a 50% reduction rate is strategically placed after the first fully connected layer, preventing co-adaptation among these layers. Further

measures to counter overfitting involve the introduction of data augmentation techniques. Before utilizing the input images, the model processes them by incorporating translations and altering the aspect ratio by up to 20% compared to their original characteristics. Additionally, the images undergo modifications in exposure and saturation, with changes occurring by a factor between 0 and 1.5.

In the final layer of the network, the model is designed to predict both class probabilities and bounding box coordinates for identified objects. Notably, the width and height of the bounding boxes are normalized concerning the size of the input image, ensuring their values fall within the real range between zero and one. Similarly, the center of the bounding boxes undergoes normalization, expressed as a function of the grid cell to which it belongs, thereby restricting its values to the interval between zero and one. The learning rate begins with a low value of 10^{-3}, The rate slowly increases in the first epochs until it reaches the value of 10^{-2}. Remains constant until epoch 75 and for the following 30 epochs a rate of 10^{-3} is applied. For the last 30 epochs a learning rate of 10^{-4} is used.

LOSS FUNCTION OF YOLO

The YOLO loss function consists of:

1. Classification loss.
2. Localization loss (spatial error between the predicted and the real bounding box).
3. Confidence loss (general error of the bounding box, i.e. calculated confidence score error).

The classification loss measures the error in classifying a specific object, determining whether the identified class aligns with the object present in the image. The error in this operation is computed using a quadratic function, evaluating the conditional probabilities associated with the recognized class.

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \tag{3.3}$$

where, $\mathbb{1}_i^{\text{obj}}$ it is 1 if an object appears in the i-th cell, OR 0 otherwise;
$\hat{p}_i(c))^2$ denotes the conditional probability of the c-th class to appear in the i-th cell.

Localization loss quantifies the error in predicting the bounding boxes, including the height, width, and coordinates of the center. YOLO specifically computes this error solely for the bounding boxes responsible for recognizing an object. The localization error is determined through the following calculation:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right) \quad (3.4)$$

where, $\mathbb{1}_{ij}^{\text{obj}}$ it is 1 if the j-th bounding box of the i-th cell is responsible for recognizing an object (i.e. an object is located in that bounding box), 0 otherwise and $\lambda_{\text{coord}} = 0.5$.

It is worth noting that the error concerning the size of the bounding box is computed on the square root of the values. This adjustment is made to avoid assigning the same weight to errors irrespective of the size of the bounding box. The square root operation scales down larger values while having a lesser impact on smaller values of width and height.

The Confidence Loss is employed to measure the certainty of the prediction made for an individual bounding box. When the model identifies an object within a bounding box, the confidence loss is calculated as follows:

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_{ij} - \hat{C}_{ij} \right)^2 \quad (3.5)$$

where, $\mathbb{1}_{ij}^{\text{obj}}$ it is 1 if the j-th bounding box of the i-th cell is responsible for recognizing an object, or 0 otherwise; $\hat{C}_{ij}$ confidence score of the j-th bounding box of the i-th cell, i.e. it is one of the parameters of the bounding box, it is an output of the network.

As many bounding boxes do not encompass any objects, it creates an imbalance in the model, leading it to more frequently train to recognize backgrounds rather than objects. To address this, the confidence loss for bounding boxes that do not contain objects is adjusted by a factor (default value: 0.5). This

adjustment is expressed as:

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_{ij} - \hat{C}_{ij})^2 \tag{3.6}$$

where, $\mathbb{1}_{ij}^{\text{noobj}}$ is the complement of $\mathbb{1}_{ij}^{\text{obj}}$. $\hat{C}_{ij}$ is the confidence score of the j-th bounding box of the i-th cell.
On combining equations 3.5 and 3.6 we get,

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_{ij} - \hat{C}_{ij})^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_{ij} - \hat{C}_{ij})^2 \tag{3.7}$$

Consequently, the loss function of YOLO is expressed as follows:

$$\begin{aligned} \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \big((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2\big) \\ + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_{ij} - \hat{C}_{ij})^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_{ij} - \hat{C}_{ij})^2 \\ + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \tag{3.8}$$

Limitations of YOLO

The YOLO model enforces strong spatial constraints on bounding box predictions as each grid cell predicts only two boxes and is restricted to one class. This spatial constraint restricts the number of nearby objects the model can predict, leading to challenges in dealing with small objects appearing in groups, such as flocks of birds. The model encounters difficulties in generalizing to objects with new or unusual aspect ratios or configurations. The training process employs a loss function that approximates detection performance, yet it treats errors the same way for both small and large bounding boxes. While a small error in a large box might have limited impact, a small error in a small box significantly affects the Intersection over Union (IOU). The primary source of error in the model is attributed to incorrect localizations. YOLO faces challenges in accurately localizing objects, with localization errors constituting a significant

portion of its overall errors.

3.1.3 YOLOv3

YOLOv3 [66], an advancement over YOLOv2, introduces improvements to address limitations and enhance the overall performance of the object detection algorithm. One notable enhancement is the incorporation of Darknet-53, a neural network architecture with 53 convolutional layers. Additionally, YOLOv3 adopts multi-label classification, allowing for output labels that are not mutually exclusive, such as "pedestrian" and "child." These improvements contribute to the algorithm's capabilities in accurately detecting and classifying objects in diverse scenarios.

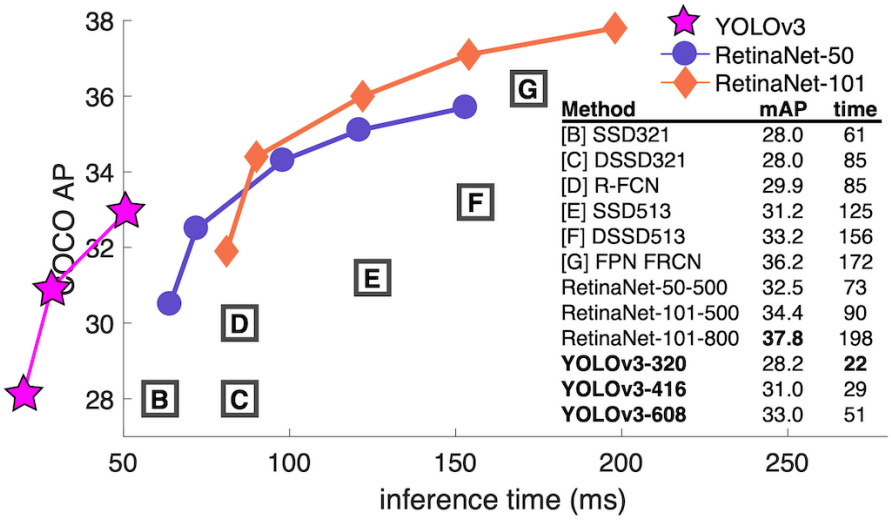

Figure 3.6: YOLOv3 runs significantly faster than other detection methods with comparable performance.

Bounding Box Prediction

In YOLOv3, the network adopts a system similar to YOLO9000 for predicting bounding boxes, utilizing dimension clusters as anchor boxes. The network provides predictions for four coordinates associated with each bounding box: t_x, t_y, t_w, t_h. These predictions are determined based on the offset of the cell from the top left corner of the image (c_x, c_y) and the width and height of the

bounding box prior (p_w, p_h):

$$\begin{aligned} b_x &= (t_x) + c_x \\ b_y &= (t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{3.9}$$

In YOLOv3, an objectness score is predicted for each bounding box through logistic regression. This score is expected to be 1 if the bounding box prior has a higher overlap with a ground truth object than any other bounding box prior. Bounding boxes with class probabilities below a certain threshold are discarded, and the author of YOLOv3 uses a threshold value of 0.5.

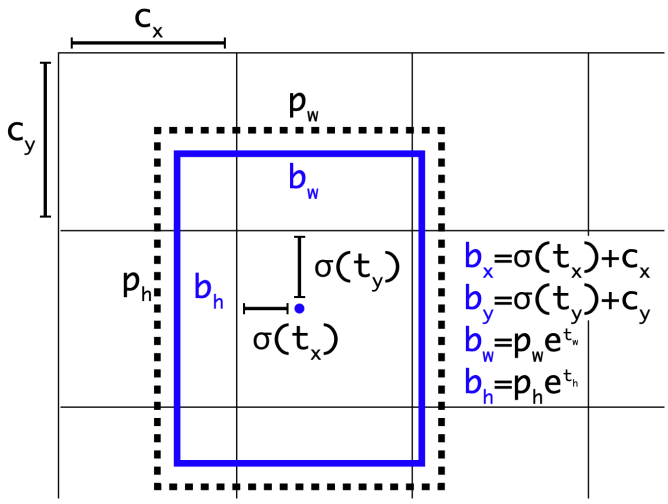

Figure 3.7: Bounding boxes dimensions and location prediction.

Class Prediction

Each bounding box predicts the classes it may contain through multilabel classification. Unlike the standard DarkNet [67], which uses the softmax function, YOLOv3 employs independent logistic classifiers to estimate the likelihood of the input belonging to specific labels. YOLOv3 introduces changes in the calculation of the cost function and predicts boxes at three different scales. This multiscale approach, inspired by feature pyramid networks, allows the system to extract features from different scales, aiding in the detection of objects with varying sizes and handling scale variations.

Feature Extraction

The developers introduced a new network for feature extraction called Darknet-53. This network adopts a hybrid approach, incorporating elements from the YOLOv2 network, Darknet-19, and the concept of residual networks. Darknet-53 utilizes a sequence of 3 x 3 and 1 x 1 convolutional layers, and unlike its predecessor, includes shortcut connections. With a total of 53 convolutional layers, Darknet-53 is a significantly larger network designed to enhance feature extraction capabilities.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.8: Structure of DarkNet-53

Intersection over Union (IoU)

Intersection over Union (IoU) is a critical component in the YOLOv3 and is commonly employed in state-of-the-art object detection models. IoU serves as a metric to assess the extent of overlap between predicted bounding boxes and

ground truth bounding boxes. The concept involves comparing the ratio of the overlapping area to the total combined region of the two boxes, expressed as:

$$IoU = \frac{\text{Area of overlap}}{\text{Area of Union}} \tag{3.10}$$

In the YOLOv3, object detection relies on bounding boxes and the concept of Intersection over Union (IoU). A score of 1 indicates a perfect match between the predicted bounding box and the ground truth box whereas a score of 0 signifies no overlap between the predicted and ground truth boxes.

Differences from YOLOv2

- **Multi-Scale Detection:** YOLOv3 introduces the concept of multi-scale detection, allowing the model to detect objects of different sizes more effectively.
- **Deeper Backbone:** YOLOv3 uses the Darknet-53 backbone, which is a deeper neural network compared to the Darknet-19 used in YOLOv2. This deeper backbone captures more complex features.
- **Feature Pyramid Network:** YOLOv3 incorporates a Feature Pyramid Network to handle scale variations and improve the detection of small and large objects.
- **IoU Loss:** YOLOv3 replaces the traditional mean squared error loss with the IoU loss for bounding box predictions.
- **Class Confidence Threshold:** YOLOv3 introduces class confidence threshold, providing a way to filter out low-confidence predictions.

3.1.4 YOLOv4

YOLOv4, introduced in April 2020 by Alexey Bochkovsky and his team [3], represents the fourth version of the YOLO series. This version achieved state-of-the-art (SOTA) performance on the COCO dataset, which comprises 80 different object classes.

The architecture consists of a set of input training images which will be fed to the network and they are processed in batches in parallel. Next are the Backbone and the Neck which do the feature extraction and aggregation. The Detection Neck and Detection Head together can be called the Object Detector. Finally, the head does the detection/prediction both localization and classification.

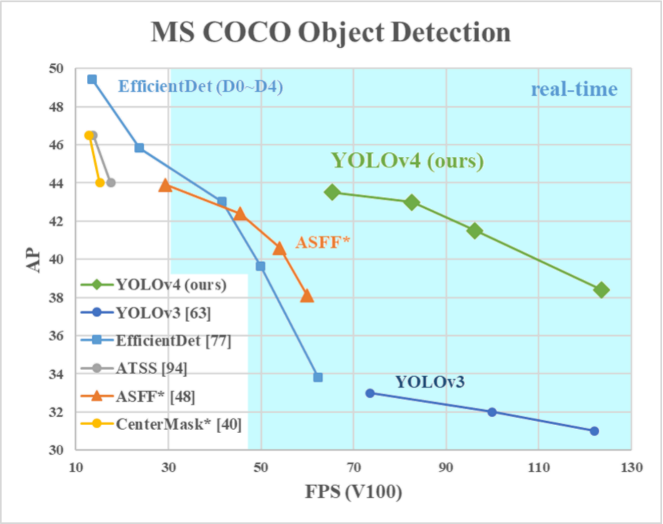

Figure 3.9: Comparison of the YOLOv4 and other state-of-the-art object detectors.

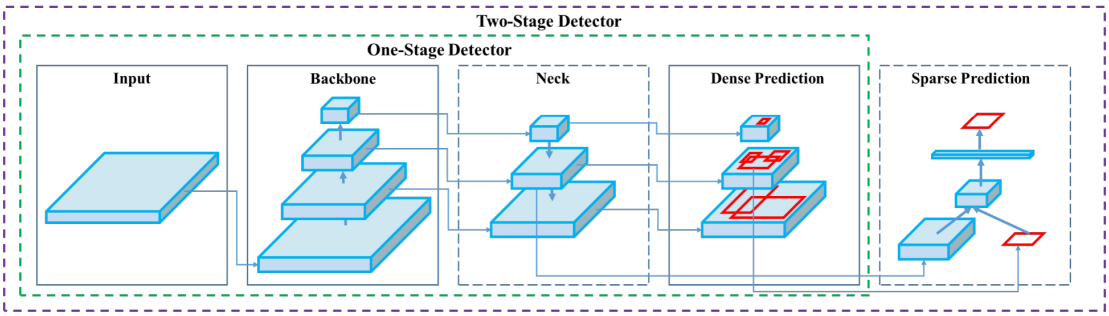

Figure 3.10: Different components of object detector.

An ordinary object detector is composed of several parts:

- **Input**: Image, Patches or Image Pyramid
- **Backbones**: VGG16[77], ResNet50[15], SpineNet[88], EfficientNet B0/B7[83], CSPResNeXt50[14], CSPDarknet53[14]
- **Neck**:
  - Additional blocks: SPP[42], ASPP[50], RFB[54]
  - Path-aggregation blocks: FPN [84], PAN[53], NAS-FPN[31], Fully-connected FPN, BiFPN[64], ASFF[79], SFAM[65]
- **Heads**:
  - Dense Prediction (one-stage):
    RPN[69], SSD[55], YOLO[39], RetinaNet[92] (anchor based)
    CornerNet[46], CenterNet[43], MatrixNet[1], FCOS[94] (anchor free)

- Sparse Prediction (two-stage): Faster R-CNN [69],
  R-FCN [37], Mask RCNN[41] (anchor based)
  RepPoints[91] (anchor free)

Backbone Network

Initially, the authors evaluated CSPResNet50, CSPDarknet53, and EfficientNet-B3 as potential backbone networks but based on extensive testing and experimental results, they opted for CSPDarknet53. It follows the DenseNet design, employing a dense connectivity pattern where the previous inputs are concatenated with the current input before passing through the dense layers. It consists of two blocks:

- Convolutional Base Layer
- Cross Stage Partial (CSP) Block

The Cross Stage Partial (CSP) strategy involves dividing the feature map in the base layer into two parts and merging them through a Cross-stage hierarchy. This approach facilitates increased gradient flow through the layers, addressing the challenge of vanishing gradients. The Convolutional Base Layer consists the entire input feature map. The CSP block, positioned alongside the Convolutional Base layer, partitions the input into two halves, where one half undergoes processing through the dense block, and the other half is directly routed to the subsequent step without any processing. This strategy in CSP helps retain fine-grained features, promotes the network to reuse features, and reduces the overall number of network parameters.

Neck

The neck serves as the component responsible for feature aggregation. It gathers feature maps from various stages of the backbone and employs a combination of bottom-up and top-down paths to mix and integrate these features, preparing them for subsequent processing. Typically, a neck comprises multiple pathways for both bottom-up and top-down operations.

Between the CSPDarkNet53 backbone and the feature aggregator network (PANet) in YOLOv4's architecture, an extra block named SPP (Spatial Pyramid Pooling) is introduced. This addition aims to expand the receptive field, isolating crucial contextual features with minimal impact on network operation speed. SPP is linked to the final layers of the densely connected convolutional layers of CSPDarkNet.

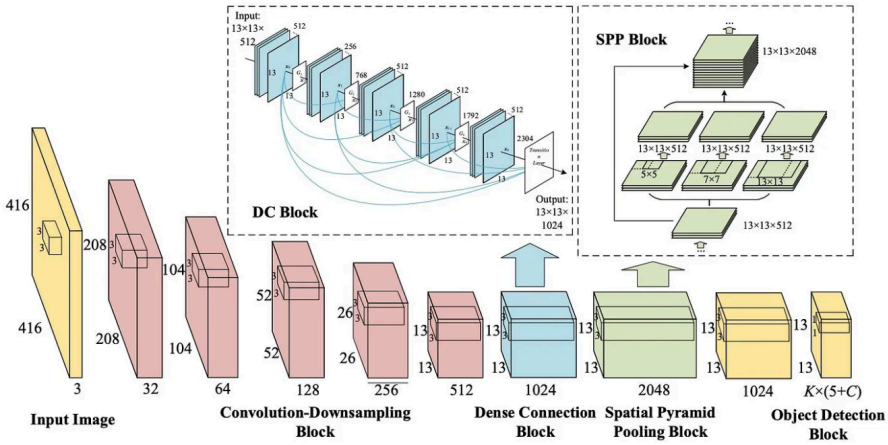

Figure 3.11: The diagram demonstrates how SPP block is integrated into YOLOv4

The Path Aggregation Network (PANet) plays a crucial role in improving instance segmentation by preserving spatial information, leading to more accurate localization of pixels for mask prediction. The main properties of PANet include Bottom-up Path Augmentation, Adaptive Feature Pooling, and Fully-Connected Fusion. A key modification in the YOLOv4 PANet involves concatenating neighboring layers instead of adding them during the use of Adaptive Feature Pooling. This modification contributes to the accuracy of mask prediction.

YOLOv4 Additions

Two new terms were introduced by the authors called Bag of Freebies (BoF) and Bag of Specials (BoS).

The Bag of Freebies (BoF) aimed to improve the network's performance without introducing additional inference time. These enhancements primarily involve data augmentation techniques. Data augmentation is a strategy that involves creating various versions of a single image by applying transformations

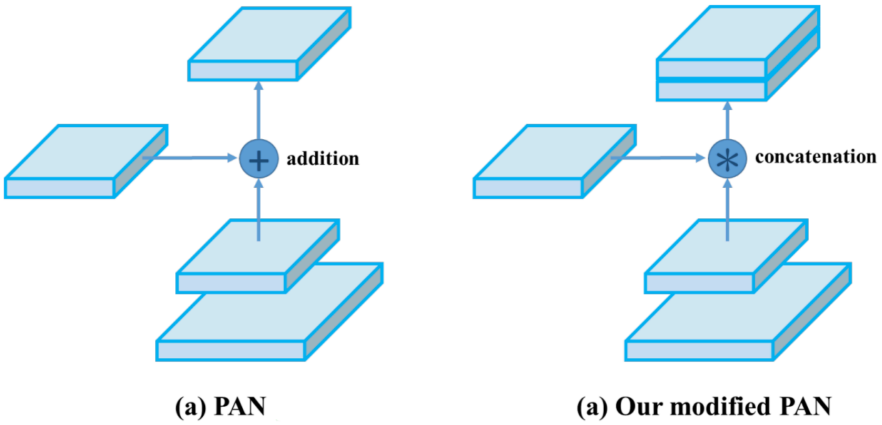

Figure 3.12: Modified PAM

such as rotation, scaling, and flipping. The goal is to augment the training dataset and make the network more robust, improving its ability to make accurate predictions across different variations of input data.

Mosaic Data Augmentation and Self-Adversarial Training (SAT) are the two main techniques introduced with this architecture.

- Bag of Freebies (BoF) for the backbone includes CutMix and Mosaic data augmentation, Drop-Block regularization, and Class label smoothing.
- Bag of Freebies (BoF) for the detector include CIoU-loss, CmBN, Drop-Block regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler, Optimal hyper-parameters, Random training shapes.

Bag of Specials (BoS): These strategies add marginal increases to inference time but significantly increase performance.

- BoS for backbone: Mish activation, Cross-stage partial connections (CSP), and Multi-input weighted residual connections (MiWRC)
- BoS for detector: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, and DIoU-NMS.

Non-maximum suppression

The idea of Non-maximum suppression is to remove redundant and overlapping bounding boxes. It works in one class at a time. For a particular class, it picks the box with the maximum score obtained using SVM. Then it calculates the IoU score with all other bounding boxes belonging to that class. The boxes having IoU score greater than 70% are removed. In other words, the bounding boxes which have very high overlap are removed. Then the next highest score box is chosen and soon all the overlapping bounding boxes are removed for that class. This is done for all classes to obtain the result as shown in the figure below.

Non-maximum suppression algorithm

Input: A list of Proposal boxes B, corresponding confidence scores **S** and overlap threshold **N**.
Output: A list of filtered proposals **D**.
Algorithm:

1. Select the proposal with the highest confidence score, remove it from B, and add it to the final proposal list D. (Initially D is empty).
2. Now compare this proposal with all the proposals calculate the IOU (Intersection over Union) with every other proposal. If the IOU is greater than the threshold N, remove that proposal from B.
3. Again take the proposal with the highest confidence from the remaining proposals in B, remove it from B, and add it to D.
4. Once again calculate the IOU of this proposal with all the proposals in B and eliminate the boxes that have higher IOU than the given threshold.
5. This process is repeated until there are no more proposals left in B.

3.2 Activation functions

The primary objective of a neural network is to convert non-linearly separable input data into a more linearly separable representation through a series of layers. These layers consist of combinations of linear and nonlinear activation functions several popular options include Logistic Sigmoid, Tanh, ReLU, ELU, Swish, and Mish. They play a very pivtol role in introducing nonlinearity and

the goal of any activation function is to avoid problems for instance vanishing gradient, exploding gradient, overfitting, and underfitting. For instance, ReLU is effective in addressing the vanishing gradient problem by permitting only positive values, preventing the gradient from diminishing during backpropagation. In contrast, activation functions like sigmoid and tanh can potentially lead to vanishing or exploding gradients, especially in deep networks. The choice of activation function is a critical consideration in the design of neural networks, impacting their ability to capture complex patterns and hierarchical representations in the data. [58]. This study considers the different activation functions below, namely the widely used ReLU and several variants.

3.2.1 ReLU

ReLU is a widely adopted activation function for hidden layers owing to its simplicity and non-saturating characteristics. On the other hand, Sigmoid and Tanh activations are frequently employed in the output layer for binary and multi-class classification tasks, respectively. This choice is driven by the requirement for interpretable outputs that represent probabilities. In practice, the selection of activation functions remains an active area of research, with various proposed variants and ongoing studies. The following summary includes the functions used, along with their derivatives.
The well-known ReLU activation function is defined as:

$$y_i = f(x_i) = \begin{cases} 0, & x_i < 0 \\ x_i, & x_i \geq 0 \end{cases} \tag{3.11}$$

and its derivative is zero for negative or zero values of net and 1 for positive values and therefore evaluated as:

$$\frac{dy_i}{dx_i} = f^{'}(x_i) = \begin{cases} 0, & x_i < 0 \\ 1, & x_i \geq 0 \end{cases} \tag{3.12}$$

In our study, we also included learnable ReLU and wider learnable ReLU. Unlike traditional ReLU, where the rectification threshold is fixed at zero, in Learnable ReLU, the threshold is a trainable parameter. The Wider Learnable ReLU Layer is an extension of the Learnable ReLU Layer that introduces an

additional scaling parameter to control the width of the rectified output. The wider parameter scales the rectified output by a learned factor.

3.2.2 Leaky ReLU

Leaky ReLU addresses the dying ReLU problem, where the gradient of ReLU becomes zero for inputs of 0 or negative values, hindering back-propagation for that neuron.

$$y_i = f(x_i) = \begin{cases} \alpha x_i, & x_i < 0 \\ x_i, & x_i \geq 0 \end{cases} \tag{3.13}$$

where α is a small real number which is 0.01 in this study. The main advantage of Leaky ReLU is that the gradient is always positive (no point has a zero gradient):

$$\frac{dy_i}{dx_i} = f^{'}(x_i) = \begin{cases} \alpha, & x_i < 0 \\ 1, & x_i \geq 0 \end{cases} \tag{3.14}$$

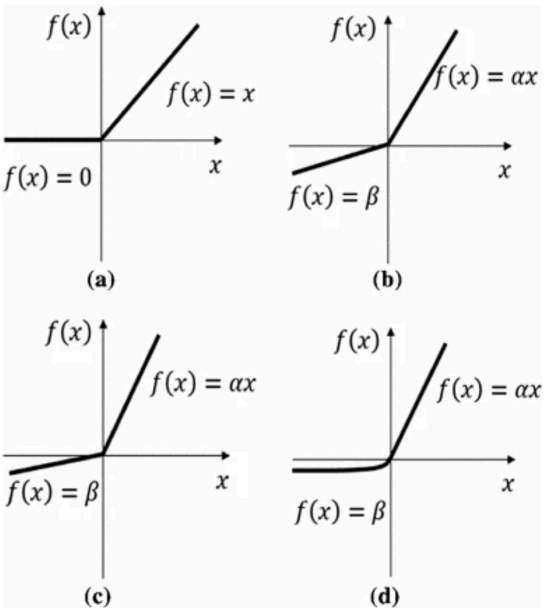

Figure 3.13: Illustrations of different activation functions: ReLU, LReLU, PReLU, and ELU

3.2.3 Scaled Exponential Linear Unit (SELU)

The third variant of ReLU considered is the Scaled Exponential Linear Unit (SELU). SELU is specifically designed to exhibit self-normalizing properties, intending to maintain the mean and variance of activations close to certain values for improved training stability. The SELU function is defined as follows:

$$y_i = f(x_i) = \begin{cases} s\alpha(exp(x_i) - 1), & x_i < 0 \\ sx_i, & x_i \geq 0 \end{cases} \tag{3.15}$$

where α and s are real numbers in our case $alpha = 1.6733$ and $s = 1.0507$. SELU is similar to ELU but includes additional scaling parameters to address gradient issues such as exploding or vanishing. The gradient in this case is given by:

$$\frac{dy_i}{dx_i} = f^{'}(x_i) = \begin{cases} s\alpha exp(x_i), & x_i < 0 \\ s, & x_i \geq 0 \end{cases} \tag{3.16}$$

3.2.4 Parametric ReLU (PReLU)

The Parametric ReLU (PReLU) is the fourth variant that is considered here. PReLU is an activation function that extends the Rectified Linear Unit (ReLU) by introducing a learnable parameter to control the slope of the negative part of the function. While the standard ReLU uses a fixed slope (zero for negative values), PReLU allows this slope to be learned during training. It is defined by:

$$y_i = f(x_i) = \begin{cases} \alpha_c x_i, & x_i < 0 \\ x_i, & x_i \geq 0 \end{cases} \tag{3.17}$$

where α_c is a set of real numbers, one for each input channel. PReLU is similar to Leaky ReLU, with the key distinction that the α_c parameters are learned. The gradient of PReLU is given by:

$$\frac{dy_i}{dx_i} = f^{'}(x_i) = \begin{cases} \alpha_c, & x_i < 0 \\ 1, & x_i \geq 0 \end{cases} \textbf{ and } \frac{dy_i}{d\alpha_c} = f^{'}(x_i) = \begin{cases} x_i, & x_i < 0 \\ 0, & x_i \geq 0 \end{cases} \tag{3.18}$$

PReLU offers a notable advantage over traditional ReLU by allowing the

learning of slopes for negative values. This becomes particularly valuable in addressing the "dying ReLU" problem, where neurons can become inactive (output zero) for all inputs during training, leading to issues such as vanishing gradients.

3.2.5 S-Shaped ReLU (SReLU)

The fifth variant, S-shaped ReLU (SReLU), is defined as a piece-wise linear function with a smooth transition between its linear and nonlinear regions. To achieve this, it introduces two additional parameters, t^l and t^r, which control the points at which the function transitions from linear to nonlinear. The SReLU function is expressed as follows:

$$y_i = f(x_i) = \begin{cases} t^l + \alpha^l(x_i - t^l), & x_i < t^l \\ x_i, & t^l \leq x_i \leq t^r \\ t^r + \alpha^r(x_i - t^r), & x_i > t^r \end{cases} \tag{3.19}$$

In this case, four learnable parameters are used, t^l, t^r, α^l, and α^r expressed as real numbers. Where t^l and t^r are the left and right transition points and α^l and α^r are the slopes of the linear regions to the left and right of the nonlinear part. They are initialized to $\alpha^l = 0$, $t^l = 0$, $t^r = maxInput$, where $maxInput$ is a hyper-parameter. SReLU is highly flexible thanks to the rather large number of tunable parameters. The gradients are given by:

$$\frac{dy_i}{dx_i} = f'(x_i) = \begin{cases} \alpha^l, & x_i < t^l \\ 1, & t^l \leq x_i \leq t^r, \\ \alpha^r, & x_i > t^r \end{cases} \tag{3.20}$$

$$\frac{dy_i}{\alpha^l} = \begin{cases} x_i - t^l, & x_i < t^l \\ 0, & x_i \geq t^l \end{cases} \tag{3.21}$$

$$\frac{dy_i}{t^l} = \begin{cases} -\alpha^l, & x_i < t^l \\ 0, & x_i \geq t^l \end{cases} \tag{3.22}$$

3.2.6 Adaptive Piece-wise Linear Unit (APLU)

The Adaptive Piece-wise Linear Unit APLU is the fifth variant. As the name suggests, it is characterized by a linear piece-wise function with adaptive slopes and biases, allowing it to approximate any continuous function within a compact set:

$$y_i = ReLU(x_i) + \sum_{c=1}^{n} \alpha_c min(0, -x_i + b_c) \tag{3.23}$$

where α_c and b_c are real numbers, one for each input channel. The gradient of ALPU is given by the sum of the gradients of ReLU and of the functions contained in the sum. With respect to the parameters, α_c and b_c, the gradients are:

$$\frac{df(x,a)}{d\alpha_c} = \begin{cases} -x + b_c, & x < b_c \\ 0, & x \geq b_c \end{cases} \textbf{ and } \frac{df(x,a)}{db_c} = \begin{cases} -\alpha_c, & x < 0 \\ 0, & x \geq 0 \end{cases} \tag{3.24}$$

3.2.7 Gaussian ReLU (GALU)

The Gaussian ReLU, also called GaLU, Its definition is based on the Gaussian type functions:

$$\phi_g^{\alpha,\lambda}(x) = max(\lambda - |x - \alpha|, 0) + min(|x - \alpha - 2\lambda| - \lambda, 0), \tag{3.25}$$

where α and λ are real numbers. The GaLU activation function is defined as:

$$y_i = GaLU(x_i) = PReLU^{c_0}(x_i) + \sum_{j=1}^{k-1} c_j \phi_g^{\alpha_j,\lambda_j}(x_i) \tag{3.26}$$

The parameter k represents the number of learnable parameters for each input channel, c_j are the learnable parameters, c_0 is the parameter vector in PReLU, and α_j and λ_j are fixed parameters chosen recursively.

3.2.8 Soft-Root-Sign (SRS)

In contrast to ReLU, SRS has a non-monotonic region when x < 0 which helps capture negative information and provides zero-mean property. Meanwhile, SRS is bounded output when x > 0 which avoids and rectifies the output

distribution to be scattered in the non-negative real number space [90].

$$SRS(x) = \frac{x}{\frac{x}{\alpha} + e^{-\frac{x}{\beta}}} \tag{3.27}$$

where α and β are a pair of trainable non-negative parameters. Its derivative can be defined as:

$$SRS'(x) = \frac{(1 + \frac{x}{\beta})e^{-\frac{x}{\beta}}}{(\frac{x}{\alpha} + e^{-\frac{x}{\beta}})^2} \tag{3.28}$$

3.2.9 SWISH and MISH Activation

According to research carried out in this paper [45] SWISH activation function performs better than ReLU activation function, and also its variants because none of these variants have managed to replace the inconsistent gains (i.e. calculation of derivatives). SWISH can be considered a type of self-gated function, expressed as:

$$SWISH(x) = x * sigmoid(x\beta) \tag{3.29}$$

Where x is the input of the activation function and β is the hyperparameter. Although the introduction of SWISH solved both vanishing gradient and provide consistent gains, the development of the MISH activation function turned out to provide an equivalent, and in many tasks, it had even better performance than the SWISH activation function. Its mathematical form is presented as:

$$MISH(x) = x * tanh(ln(1 + e^x)) \tag{3.30}$$

We also included the learnable swish layer and learnable mish layer. In the Learnable Swish Layer, the parameter β becomes trainable. This allows the network to adaptively adjust the shape of the Swish function based on the input data. Similarly, in the Learnable Mish Layer, the parameter α in $tanh(\alpha ln(1+e^x))$ becomes trainable. This enables the network to learn the optimal shape of the Mish function for the given task.

3.3 Transfer Learning

In this project, we used transfer learning techniques to train pre-trained ResNet50 and Darknet53 for our object detection task. Basically, it consists of

importing previously trained CNN models trained on a large dataset for a specific task, typically on a large-scale dataset like ImageNet for image classification or BERT for natural language processing for a specific problem, and using them after some small adaptations for a problem in hand. Transfer learning was first introduced in 1976 by Stevo Bozinovski and Ante Fulgosi [8] for the field of machine learning and artificial intelligence, its a technique that aims to transfer previous "knowledge" of a task to a new one, and it is divided in two categories:

Fine-tuning: An existing and trained network is taken and the input or output layers of the network are modified if necessary, the resulting model is trained on the new dataset (it does not need to be very large) for a different task to adapt to it. This significantly reduces the training time and computational resources required for training a model from scratch.

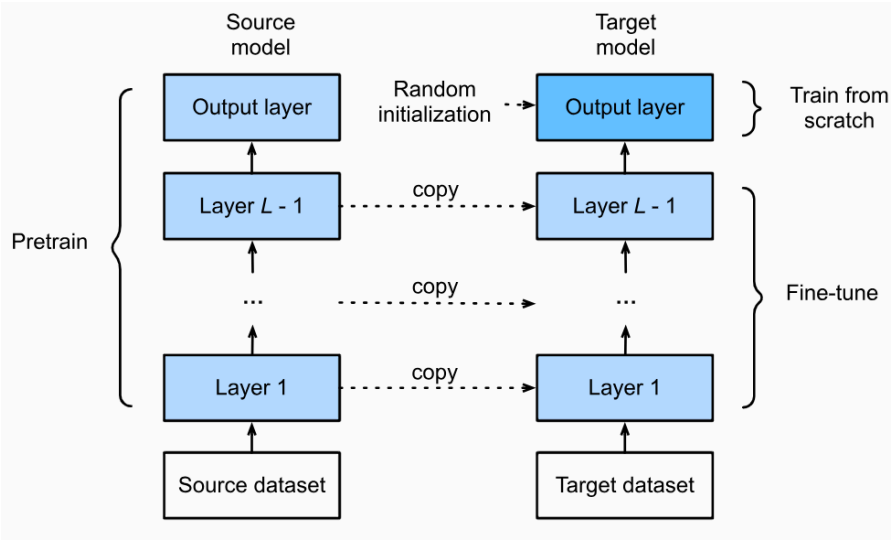

Figure 3.14: Structure of Fine Tuning [6]

Deep Features: An existing and trained network is used without further fine-tuning to generate deep feature vectors, which are then classified by another ML algorithm like a Support Vector Machine. The weights of the existing network are not updated during the training process, just the post-processing scheme that uses its deep features as inputs.

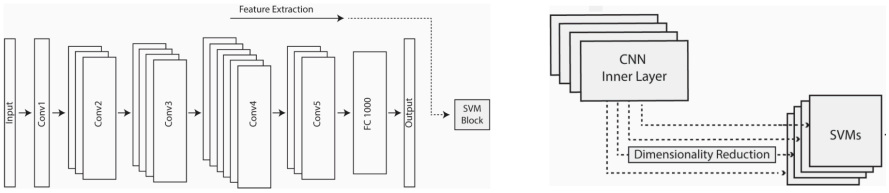

Figure 3.15: Example of deep features transfer learning [60]

In our case, the fine-tuning approach is taken, with the pre-trained architectures taken as a starting point for further training on our dataset. Fine-tuning has been shown very valuable when there isnt enough data available for a newly trained model to converge.

4

Experiments and Results

4.1 Data Augmentation

Data augmentation is a crucial technique extensively utilized in deep learning to tackle the challenge of limited annotated data, especially crucial for computer vision tasks. In the domain of deep learning, models thrive on abundant data, and the effectiveness of a model is directly tied to the quantity and quality of the data it is trained on. The objective of data augmentation methods is to improve performance by expanding the pool of training data without the necessity of collecting new data. This is accomplished by generating synthetic samples that either replicate the original ones with modifications or are automatically created to possess the same statistical characteristics as the real samples, or a combination of both. The approach to generating additional samples varies based on the specific requirements of the classification task. A comprehensive exploration of data augmentation techniques for deep learning is available in [75]. The process can be categorized into two main groups.

The first one is data augmentation through image manipulations and it can be further categorized into various classes. Geometric transforms, including rotation, flipping, warping, cropping, and more, can alter the spatial arrangement of the image. Filters, such as low-pass filters and noise injection, introduce variations to the image. The technique of random erasing, introduced by [2], involves replacing random pixel regions with a constant value or noise. Statistical approaches, like equalization and color casting, can alter the color space of the

image. Furthermore, the generation of new images can be achieved by per-pixel weighted mixes of other images.

The other is deep learning techniques for image data augmentation, which consists of various strategies. Feature space augmentation involves utilizing the lower-dimensional representations of images generated by intermediate layers of convolutional neural networks to create new data. Adversarial training is another approach where an auxiliary network produces synthetic images to mislead the main network. Generative adversarial networks (GANs), are a widely adopted method for generating synthetic images resembling real ones. Neural style transfer is a technique that uses an auxiliary network to transfer the style of one image to another while retaining the original content. These deep-learning approaches serve as potent tools for significantly enhancing the performance of image classification models.

Here are the examples of data augmentation techniques we used in this thesis to enhance our training data set:

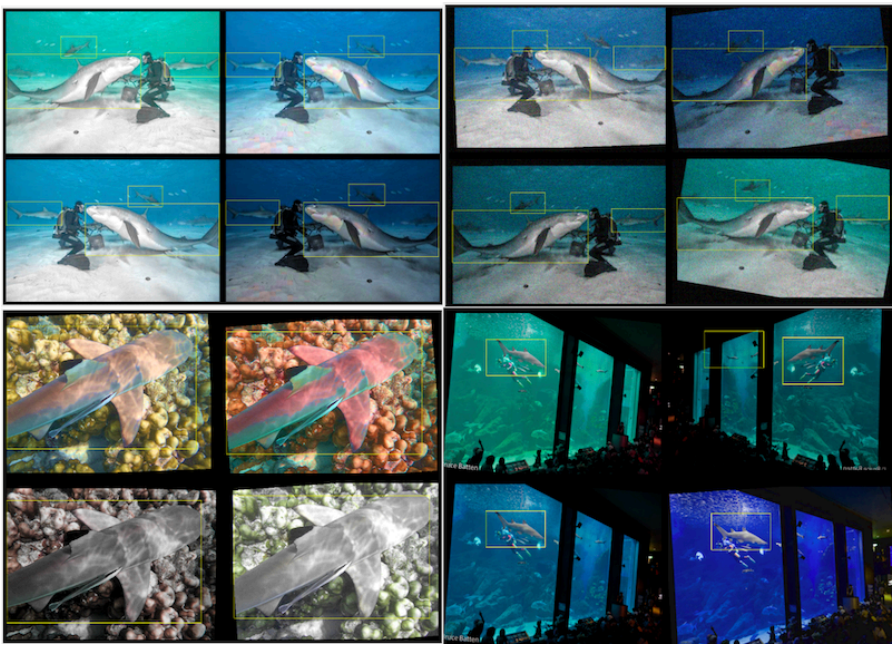

Figure 4.1: Application of random horizontal flipping, random X/Y scaling, random rotation, random translation, motion blur, and Gaussian noise.

4.2 Work Flow of our Proposed Method

The workflow in this project aims to apply the Stochastic Selection of Activation Layer for CNNs - SSAL method to the Shark Detector - SD originally presented in this paper [36]. We test our approach on Shark Identifier which is a classification task and Shark Locator which is an object detection task.

4.2.1 Shark Identifier/classifier

The development of the classifier is similar to any classification task and is mainly composed of these steps. First, we took ResNet50 CNN which is pre-trained on the imageNet dataset [17]. These neural networks have been trained on more than a million images and can classify images into 1000 object categories. We fine-tune ResNet50 according to our classification task two classes shark and non-shark. We created a dataset of approx. 61966 sharks and approx. 50,260 non-shark images. The input layer size of models is 224x224 pixels, with three color channels (Red, Green, and Blue). To align with the model's input layer, all dataset images were resized. The training duration for all models spanned 10 epochs. Stochastic Gradient Descent with momentum (SGDM) [80] optimizer was employed for training, with the initial learning rate set to 3×10^{-4} with a mini-batch size of 50. The final three fully connected layers underwent modification to ensure the correct number of output neurons, and their outputs were normalized using the SoftMax activation function [10]. This normalization allowed the outputs to be interpreted as probabilities. We trained five ResNet50 each having a distinct activation layer stochastically drawn from a set of activation functions as mentioned earlier.

4.2.2 Shark Locator/Detector

Regarding the development of the Shark Locator, we took each trained network from the Shark Identifier task, and again we performed a fine-tuning process and replaced the CNN as a backbone of YOLOv4 and YOLOv3 object detection models and created our custom object detectors. The experiment was conducted on a dataset consisting of 514 shark images. Some images from the training set were reserved for validation. The training of YOLOv4 object detector involved using hyperparameters such as 50 maximum epochs, the Adam

optimizer [18], L2Regularization is set to 0.0005, an initial learning rate set to 0.001, minibatch size is set to 8, and validation dataset is used in the model training process every 5 epochs. The training of YOLOv3 object detector involved using options such as 80 maximum epochs, the SGDM optimizer [18], L2Regularization is set to 0.0005, and an initial learning rate set to 0.001. We subsequently combined the trained networks into an ensemble and assessed and compared the individual locator's performance and ensemble performance based on our modified ResNet50s with YOLO's base CNN network darknet53.

4.3 Ensemble Learning Algorithm

Ensemble learning refers to the idea of combining the predictions done by multiple object detectors into a final output. Instead of relying on a single model, By aggregating the predictions from diverse models, ensemble methods can often achieve higher accuracy and robustness compared to any single model.

The algorithm that we have designed to combine the object detections obtained from several detectors follows: The input of our ensemble algorithm is a list of detectors $L = [D_1, . \ . \ . \ , D_m]$ where each D_i, with $i \ \epsilon \ 1, ..., m$ is a list of detections for a given image I and m is the total number of detectors m is 5 in our case as we trained 5 ResNets and 5 DarkNet53. Each D_i contains the list of detections produced using a particular model M_i for example D_1 is the detections obtained from ResNet(1) or DarkNet53(1) for a given input image I. In a few cases, we have observed that each D_i doesn't contain the detection so we applied the majority/consensus rule meaning that out of 5 detectors if 3 of them have detections then we keep the detection and continue to process them.

Subsequently, the detections in D_i are grouped together based on the overlapping of their predicted bounding boxes. To determine the overlap of bounding boxes, the Intersection over union - IoU metric is employed. Considering two bounding boxes b_1 and b_2, the IoU formula for finding the overlapped region between them is given by

$$IOU(b_1, b_2) = \frac{area(b_1 \cap b_2)}{area(b_1 \cup b_2)} \tag{4.1}$$

This measure is employed to group the elements of L producing as a result a list $K = [D_1^K, ..., D_m^K]$ where each D_i^K is a list of detections such that for all $\bar{d}(= [\bar{b}, \bar{c}, \bar{s}])$, $\hat{d}(= [\hat{b}, \hat{c}, \hat{s}]) \epsilon D_i^K$. Where $IoU(\bar{b}, \hat{b}) > 0.5$ and $\bar{c} = \hat{c}$. At this point, each $D_i^K \epsilon$ K is focused on a particular subject in the input image and we merge all the predicted bounding boxes of K_i by computing the mean of the bounding boxes in K_i. At this point, list K has the most robust bounding boxes for each detector i.e. one bounding box for each subject and the final step is to calculate IOU and merge all the bounding boxes in K for the final detection.

4.4 Metrics

The performance of shark classifier five in our case is evaluated using metrics: precision, recall, accuracy, and F1-Score. These metrics measure the ability of the classifiers to classify between our two classes accurately. Formally, they can be described as:

- Precision = $\frac{TP}{TP+FP}$ The rate of correct positive predictions among all positive predictions.
- Recall = $\frac{TP}{TP+FN}$, The rate of correct positive predictions among all true positives.
- Specificity = $\frac{TN}{TN+FP}$, The rate of correct negative predictions among all true negatives.
- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$, The proportion of correct predictions among all predictions.
- F1 score = $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$, The harmonic mean of precision and recall.

True Positives (TP) are instances correctly predicted as positive by the model. False Positives (FP) are instances incorrectly predicted as positive by the model. In other words, the model predicted a positive outcome, but the true class is negative. False Negatives (FN) are instances incorrectly predicted as negative by the model. In other words, the model predicted a negative outcome, but the true class is positive. True Negatives (TN) are instances correctly predicted as negative by the model. The performance of the individual shark detector and its ensemble was evaluated using Recall, Precision, and Specificity. We formed ensembles by combining our five custom detectors (ResNet50) and five base networks darknet53 and assessed their scores using these metrics.

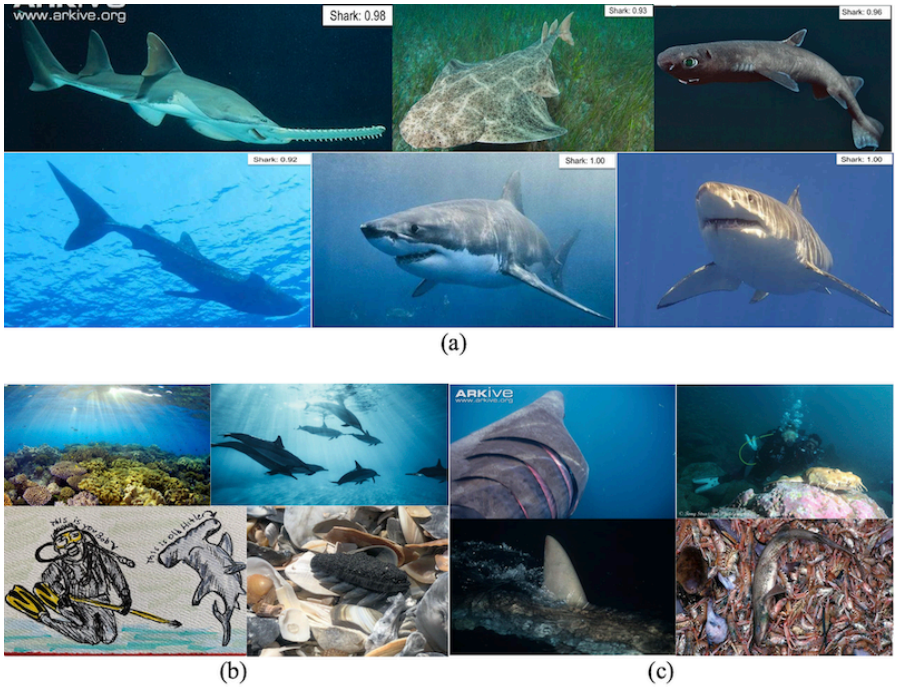

Figure 4.2: (a) True Positive: TP, (b) False Positive (FP). (c): False Negative (FP)

The figure demonstrates the accurate identification of a diverse array of shark images, encompassing underwater photographs, images with foreground and background noise, images featuring challenging-to-discern shark features, and various shark species. Image (b) illustrates instances where the system misclassified common subjects, including cetaceans and other marine and terrestrial animals, empty foregrounds, mysterious objects, and fake sharks. Additionally, the images in the right corner (c) highlight situations where the system failed to detect shark presence due to partially concealed features.

Models	**Accuracy**	**Precision**	**Recall**	**F1-Score**
ResNet50(1)	0.976	0.969	0.978	0.973
ResNet50(2)	0.977	0.978	0.971	0.974
ResNet50(3)	0.978	0.981	0.970	0.975
ResNet50(4)	0.977	0.980	0.969	0.974
ResNet50(5)	0.976	0.970	0.977	0.974

Table 4.1: The results of individual Shark Identifier/classifier

The above table shows the results from the shark classifier made up of different ResNet50 models with modified activation function layers indicating consistently high performance across various metrics. Overall, ResNet50(3) achieved the highest accuracy of 97.8%, closely followed by ResNet50(2) and ResNet50(4) with accuracies of 97.7%. Precision values were consistently high, indicating a low false positive rate, with ResNet50(3) achieving the highest precision of 98.1%. Similarly, recall values, which measure the ability to identify positives correctly, were also commendable, with ResNet50(2) achieving the highest recall of 97.1%. The F1-Score, which balances precision and recall, remained consistently high across all models, with ResNet50(3) achieving the highest F1-Score of 97.5%. These results demonstrate the robustness and effectiveness of the ResNet50 architecture in our classification task.

Architectures	**YOLOv3 (T=0.5)**			**YOLOv3 (T=0.7)**			**YOLOv4 (T=0.5)**			**YOLOv4 (T=0.7)**		
	Recall	Precision	Specificity	Recall	Precision	Specificity	Recall	Precision	Specificity	Recall	Precision	Specificity
Resnet50(1)	0.91	0.82	0.82	0.86	0.86	0.88	0.91	0.82	0.82	0.86	0.86	0.88
Resnet50(2)	0.90	0.83	0.83	0.86	0.84	0.85	0.93	0.85	0.85	0.87	0.86	0.88
Resnet50(3)	0.90	0.83	0.84	0.86	0.86	0.88	0.90	0.86	0.87	0.87	0.87	0.88
Resnet50(4)	0.91	0.82	0.82	0.86	0.84	0.86	0.91	0.86	0.87	0.86	0.86	0.88
Resnet50(5)	0.91	0.83	0.84	0.86	0.86	0.88	0.91	0.85	0.85	0.86	0.86	0.88
Darknet53(1)	0.90	0.82	0.82	0.86	0.85	0.86	0.91	0.84	0.84	0.84	0.84	0.85
Darknet53(2)	0.91	0.84	0.84	0.88	0.86	0.86	0.92	0.82	0.82	0.85	0.85	0.87
Darknet53(3)	0.91	0.85	0.85	0.86	0.86	0.86	0.90	0.88	0.88	0.85	0.85	0.86
Darknet53(4)	0.90	0.85	0.85	0.86	0.85	0.85	0.90	0.88	0.88	0.84	0.84	0.86
Darknet53(5)	0.90	0.85	0.85	0.88	0.86	0.86	0.90	0.88	0.88	0.85	0.85	0.87
EResnet50(5)	0.93	0.86	0.86	0.86	0.86	0.88	0.94	0.86	0.86	0.88	0.86	0.88
EDarknet53(5)	0.92	0.84	0.84	0.86	0.86	0.88	0.93	0.84	0.86	0.86	0.86	0.88
EResnet50(5) + EDarknet53(5)	0.93	0.85	0.85	0.86	0.86	0.88	0.94	0.84	0.84	0.87	0.86	0.88

Table 4.2: The results of individual and ensemble tests for YOLOv3 and YOLOv4

The above table shows the result between YOLO's baseline (DarkNet53) and our modified Resnet50 for YOLOv3 and YOLOv4. According to the obtained results, our modified ResNet50 performs better i.e. ResNet50(2) performs better than all darknet53, and if we can see ensemble i.e. EResNet50(5) and EDarknet(50)is the ensemble of five networks that we trained in classification tasks outperform individual networks and EResNet50(5) performs better in comparison to EDarknet(50) with recall of 0.93 and 0.94 for YOLOv3 and YOLOv4.

5

Conclusion

In this study, a novel approach to modify convolutional neural network (CNN) layer was introduced, involving changing CNN model architecture by stochastic layer replacement. The proposed method entails the random replacement of each activation layer in a CNN with a distinct activation function selected from a predefined set. This results in a model featuring diverse activation function layers, introducing variability and making it well-suited for ensemble creation.

Remarkably, this designed approach demonstrated outstanding performance in ensemble creation. The ResNet50 model, created by stochastically replacing ReLU layers and combined using a majority rule, outperformed the basenet of YOLO and an individual stochastic ResNet50 in our experimental evaluations. Through a comprehensive experimental evaluation focused on shark object detection from images, our approach showcased its potential for building high-performance CNN ensembles.

we plan in future work to evaluate the proposed method on a large class of models including lighter architectures. Exploration of the latest state-of-the-art YOLO models in multi-class object detection and zero-shot object detection methods on large-scale datasets. The difficulty of studies involving the training of CNNs, detectors and ensembles of CNNs lies in the GPU speed and memory resources required to conduct such experiments.

References

[1] Agastya Kalra Abdullah Rashwan and Pascal Poupart. "Matrix Nets: A new deep architecture for object detection." In: 2019. DOI: https://arxiv.org/abs/1908.04646.

[2] Zhun Zhong et al. "Random Erasing Data Augmentation". In: 2020. DOI: 10.1609/aaai.v34i07.7000.

[3] Chien-Yao Wang Alexey Bochkovskiy and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: DOI: https://arxiv.org/pdf/2004.10934v1.pdf.

[4] A. Palmer Alvarez Ellacura and 2019 M. Catalan I.A. Lisani. "Image-based, unsupervised estimation of fish size from commercial landings using deep learning." In: DOI: https://doi.org/10.1093/icesjms/fsz216.

[5] Lourakis M.I.A. Argyros A.A. "Real-time tracking of multiple skin-colored objects with a possibly moving camera." In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*. 2004.

[6] Zachary C. Aston Zhang. "Dive Into Deep Learning". In: DOI: https://d2l.ai/chapter_computer-vision/fine-tuning.html.

[7] J.K. Baum and 2010. Blanchard W. "Inferring shark population trends from generalized linear mixed models of pelagic longline catch and effort data". In: pp. 229–239. DOI: https://doi.org/10.1016/j.fishres.2009.11.006.

[8] Stevo. Bozinovski and Ante Fulgosi. "The influence of pattern similarity and transfer learning upon the training of a base perceptron B2". In: DOI: https://www.informatica.si/index.php/informatica/article/view/2828/1433.

[9] Waldeland A.U. Brautaset O. et al. "Acoustic classification in multifrequency echosounder data using deep convolutional neural networks". In: DOI: https://doi.org/10.1093/icesjms/fsz235.

[10] J.S. Bridle. "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition." In: 2022. DOI: https://doi.org/10.1007/978-3-642-76153-9..

[11] Manuel Carranza-García, Pedro Lara-Benítez Jesús Torres-Mateo, and Jorge García-Gutiérrez. "On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data". In: (). URL: https://www.mdpi.com/2072-4292/13/1/89.

[12] Zhu Y. Chen L.C., Schroff F. Papandreou G., and Adam H. *ncoder-decoder with atrous separable convolution for semantic image segmentation*. 2018.

[13] Walia E. Chi J., Groot G. Babyn P. Wang J., and M. Eramian. "Thyroid nodule classification in ultrasound images by fine-tuning deep convolutional neural network". In: DOI: 10.1007/s10278-017-9997-y.

[14] Hong-Yuan Mark Liao Chien-Yao Wang, Ping-Yang Chen Yueh-Hua Wu, and I-Hau Yeh. Jun-Wei Hsieh. "CSPNet: A new backbone that can enhance learning capability of cnn." In: 2020. DOI: https://arxiv.org/abs/1911.11929.

[15] Yangqing Jia Christian Szegedy Wei Liu et al. "Going Deeper with Convolutions". In: *In Proceedings of the Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2015. DOI: https://arxiv.org/abs/1409.4842.

[16] D.A. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and accurate deep network learning by exponential linear units (ELUs)." In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. 2016. DOI: 10.48550/arXiv.1511.07289.

[17] Dong W. Deng J., Li L. J. Socher R., and Fei-Fei L. Li K. "Imagenet: A large scale hierarchical image database." In: 2022. DOI: https://ieeexplore.ieee.org/abstract/document/5206848.

[18] Jimmy Ba. Diederik P. Kingma. "Adam: A Method for Stochastic Optimization". In: 2022. DOI: https://arxiv.org/abs/1412.6980.

[19] Kuprel B. Esteva A. et al. "Dermatologist-level classification of skin cancer with deep neural networks." In: DOI: https://arxiv.org/pdf/1810.10348.pdf.

[20] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88 (2010), pp. 303–338.

[21] Fielding S. Fallon N. and Fernandes P. 2016. "Classification of Southern Ocean krill and icefish echoes using random forests." In: DOI: https://doi.org/10.1093/icesjms/fsw057.

[22] McAllester D Felzenszwalb P and Ramanan D. "A discriminatively trained, multiscale, deformable part model". In: *IEEE conference on computer vision and pattern recognition* (2008), pp. 1–8.

[23] Matthew Hoffman Forest Agostinelli and Pierre Baldi Peter Sadowski. "Learning activation functions to improve deep neural networks". In: *3rd International Conference on Learning Representations*. 2015. DOI: 10.48550/arXiv.1412.6830.

[24] Mackiewicz M. French G. et al. "Deep neural networks for analysis of fisheries surveillance video and automated monitoring of fish discards." In: DOI: https://doi.org/10.1093/icesjms/fsz149..

[25] Cheng-Yang Fu et al. *DSSD: Deconvolutional Single Shot Detector*. 2017. arXiv: 1701.06659 [cs.CV].

[26] Prados R. Garcia R. et al. "Automatic segmentation of fish using deep learning with application to fish size measurement." In: DOI: https://doi.org/10.1093/icesjms/fsz186..

[27] Ross Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[28] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[29] X. Glorot, A. Bordes, and Y. Bengio. "Deep sparse rectifier neural networks". In: *In Proceedings of the Journal of Machine Learning Research*. 2011. URL: https://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[30] Bond T. Goetze J.S. et al. "A field and video analysis guide for diver operated stereo-video." In: DOI: https://doi.org/10.1111/2041-210X.13189.

[31] Tsung-Yi Lin Golnaz Ghiasi and Quoc V Le. "NAS-FPN: Learning scalable feature pyramid architecture for object detection." In: 2019. DOI: https://arxiv.org/abs/1904.07392.

[32] Award G.M. Han J. and Wu H Sutherland A. "Automatic skin segmentation for gesture recognition combining region and support vector machine active learning." In: 2006.

[33] Zhang X. He K., S. Ren, and J. Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *IEEE International Conference of Computer Vision*. 2015. URL: https://arxiv.org/abs/1502.01852.

[34] Zhang X. He K. and J. Ren S. Sun. *Deep Residual Learning for Image Recognition*. 2016.

[35] Abdel-Mottaleb M. Hsu R.L. and Jain A.K. "Face detection in color images". In: DOI: https://ieeexplore.ieee.org/document/1000242/.

[36] Z. Y.-C. Liu J. Jenrette, T. Hastie P. Chimote, and F. Ferretti E. Fox. "Shark detection and classification with machine learning". In: DOI: https://www.sciencedirect.com/science/article/pii/S1574954122001236.

[37] Kaiming He Jifeng Dai Yi Li and Jian Sun. "R-FCN: Object detection via region-based fully convolutional networks." In: 2016. DOI: https://arxiv.org/abs/1605.06409.

[38] Micheli F. Jorgensen J. et al. "Emergent research and priorities for shark and ray conservation". In: DOI: https://doi.org/10.3354/esr01169.

[39] Santosh Divvala Joseph Redmon and Ali Farhadi Ross Girshick. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: 1506.02640 [cs.CV].

[40] Ladds M.A. Kadar J.P., Lyall B. Day J., and Brown C. 2020. "Assessment of machine learning models to identify port jackson shark behaviours using tri-axial accelerometers." In: DOI: https://doi.org/10.3390/s20247096.

[41] Georgia Gkioxari Kaiming He and Ross Girshick Piotr Dollár. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].

[42] Xiangyu Zhang Kaiming He, Shaoqing Ren, and Jian Sun. "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: 2015. DOI: arXiv:1406.4729.

[43] Song Bai Kaiwen Duan, Honggang Qi Lingxi Xie, and Qi Tian. Qingming Huang. "CenterNet: Keypoint triplets for object detection." In: 2019. DOI: arXiv:1904.08189.

[44] Unterthiner T. Klambauer G. and Hochreiter Mayr A. "Self-Normalizing Neural Networks". In: *In Proceedings of the NIPS*. 2017. URL: https://arxiv.org/abs/1706.02515.

[45] Ravin Kumar. "APTx: better activation function than MISH, SWISH, and ReLUs variants used in deep learning". In: DOI: https://arxiv.org/pdf/2209.06119.pdf.

[46] Hei Law and Jia Deng. "CornerNet: Detecting objects as paired keypoints." In: 2018. DOI: arXiv:1808.01244.

[47] Bengio Y. LeCun Y. "Convolutional Networks for Images, Speech, and Time Series." In: DOI: https://dl.acm.org/doi/10.5555/303568.303704..

[48] Bengio Y. LeCun Y. and 2015. Hinton G. "Deep learning". In: (). DOI: http://www.nature.com/articles/nature14539.

[49] Kuo Y.-M. Lee J.-S. and Chen E.-L. Chung P.-C. "Naked image detection based on adaptive and extensible skin color model." In: 2007.

[50] George Papandreou Liang-Chieh Chen, Kevin Murphy Iasonas Kokkinos, and Alan L Yuille. "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs." In: 2017. DOI: arXiv:1606.00915.

[51] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

[52] Maire M. Lin T.-Y. et al. "Microsoft CoCo: common objects in context. European conference on computer vision." In: DOI: https://doi.org/10.48550/arXiv.1405.0312..

[53] Lu Qi u Liu, Jianping Shi Haifang Qin, and Jiaya Jia. "Path aggregation network for instance segmentation." In: 2018. DOI: arXiv:1803.01534.

[54] Songtao Liu and Di Huang. "Receptive field block net for accurate and fast object detection." In: 2018. DOI: https://arxiv.org/abs/1711.07767.

[55] Anguelov D Liu W et al. *Ssd: single shot multibox detector*. 2016. arXiv: 1512.02325 [cs.CV].

[56] Aditya Lohia, Rahul Raghvendra Joshi Kalyani Dhananjay Kadam, and Dr. Anupkumar M. Bongale. "Bibliometric Analysis of One-stage and Two-stage Object Detection". In: *Library Philosophy and Practice (e-journal). 4910* (). URL: https://digitalcommons.unl.edu/libphilprac/4910.

[57] Alessandra Lumini Loris Nanni. "Fair comparison of skin detection approaches on publicly available datasets." In: DOI: arXiv1802.025312018.

[58] Alessandra Lumini Loris Nanni and Gianluca Maguolo Stefano Ghidoni. "Stochastic Selection of Activation Layers for Convolutional Neural Networks." In: DOI: https://www.preprints.org/manuscript/202002.0231/v1..

[59] Alessandra Lumini Loris Nanni and S. Stefano Ghidoni Brahnam. "Bioimage Classification with Handcrafted and Learned Features". In: DOI: https://arxiv.org/abs/1904.08084.

[60] Stefano Ghidoni Loris Nanni and Sheryl Brahnam. "Deep Features for Training Support Vector Machines". In: DOI: https://www.mdpi.com/2313-433X/7/9/177.

[61] Byra M. "Discriminant analysis of neural style representations for breast lesion classification in ultrasound." In: DOI: https://www.sciencedirect.com/science/article/abs/pii/S020852161730428X.

[62] A.L. Maas, A.Y. Hannun, and A.Y. Ng. "Rectifier nonlinearities improve neural network, acoustic models". In: *In Proceedings of the in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013. URL: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[63] Handegard N.O. Malde K. and Salberg 2019. Eikvil L. "Machine intelligence and the data-driven future of marine science." In: DOI: https://doi.org/10.1093/icesjms/fsz057.

[64] Ruoming Pang Mingxing Tan and Quoc V Le. "EfficientDet: Scalable and efficient object detection." In: 2020. DOI: arXiv:1911.09070.

[65] Tao Sheng Qijie Zhao et al. "M2det: A single-shot object detector based on multi-level feature pyramid network." In: 2019. DOI: arXiv:1811.04533.

[66] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: DOI: https://arxiv.org/pdf/1804.02767v1.pdf.

[67] J. Redmon. "Darknet: Open source neural networks in c." In: DOI: http://pjreddie.com/darknet/.

[68] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.

[69] He K. Ren S. and Sun J. 2016. Girshick R. "Faster R-CNN: towards real-time object detection with region proposal networks. CoRR". In: DOI: https://doi.org/10.48550/arXiv.1506.01497..

[70] Sahil. "Object Detection (R-CNN)". In: (). URL: https://sahiltinky94.medium.com/object-detection-r-cnn-aa2b180bfb49.

[71] Eigen D Sermanet P, Mathieu M Zhang X, and LeCun Y Fergus R. *Overfeat: integrated recognition, localization and detection using convolutional networks*. 2013. arXiv: 1312.6229 [cs.CV].

[72] Dimitris Tsipras Shibani Santurkar and Aleksander Madry Andrew Ilyas. "How Does Batch Normalization Help Optimization?" In: DOI: https://arxiv.org/abs/1805.11604.

[73] Satish Kumar Singh Shiv Ram Dubey and Bidyut Baran Chaudhuri. "Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark". In: DOI: https://arxiv.org/pdf/2109.14545.pdf.

[74] Ahmad Salman Shoaib Ahmed Siddiqui et al. "Automatic fish species classification in underwater videos: exploiting pre-trained deep neural network models to compensate for limited labeled data." In: 2018. DOI: https://doi.org/10.1093/icesjms/fsx109.

[75] Connor Shorten and Taghi M Khoshgoftaar. "A survey on image data augmentation for deep learning". In: 2019. DOI: 10.1186/s40537-019-0197-0.

[76] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: 2019. DOI: https://arxiv.org/abs/1409.1556.

[77] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: 2014. DOI: arXiv:1409.1556.

[78] Zisserman A. 2015 Simonyan K. "Very deep convolutional networks for large-scale image recognition." In: DOI: https://doi.org/10.48550/arXiv.1409.1556..

[79] Di Huang Songtao Liu and Yunhong Wang. "Learning spatial fusion for single-shot object detection." In: 2019. DOI: arXiv:1911.09516.

[80] Martens J. Sutskever I. and G. Dahl G. Hinton. "On the importance of initialization and momentum in Deep Learning." In: 2022. DOI: http://proceedings.mlr.press/v28/sutskever13.html.

[81] Kosmala M. Swanson A., R. Lintott C. Simpson, and 2015. Smith A. Packer C. "Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna." In: DOI: https://doi.org/10.1038/sdata.2015.26..

[82] Norouzzadeh M.S. Tabak M.A. et al. "Machine learning to classify animal species in camera trap images: Applications in ecology." In: DOI: https://doi.org/10.1038/sdata.2015.26..

[83] Mingxing Tan and Quoc V Le. "EfficientNet: Rethinking model scaling for convolutional neural networks." In: 2019. DOI: arXiv:1905.11946.

[84] Piotr Dollar Tsung-Yi Lin, Kaiming He Ross Girshick, and Serge Belongie Bharath Hariharan. "Feature pyramid networks for object detection." In: 2017. DOI: https://arxiv.org/abs/1612.03144.

[85] J.R.R. Uijlings, T. Gevers K.E.A. van de Sande, and A.W.M. Smeulders. "Selective Search for Object Recognition"". In: *A Technical Report 2012, submitted to IJCV* (2013). URL: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf.

[86] 2018 Weinstein B.G. "Automatic fish species classification in underwater videos: exploiting pre-trained deep neural network models to compensate for limited labeled data." In: DOI: https://doi.org/10.1093/icesjms/fsx109..

[87] Swiezewski J. Whytock R.C. et al. "Robust ecological analysis of camera trap data labelled by a machine learning model." In: DOI: https://doi.org/10.1111/2041-210X.13576.

[88] Tsung-Yi Lin Xianzhi Du et al. "SpineNet: Learning scale-permuted backbone for recognition and localization." In: 2019. DOI: arXiv:1912.05027.

[89] Lawrence K Saul Youngmin Cho. "Large-margin classification in infinite neural networks". In: 2010. DOI: 10.1162/NECO_a_00018.

[90] Dandan Li Yuan Zhou, Shuwei Huo, and Sun-Yuan Kung. "Soft-Root-Sign Activation Function". In: DOI: https://arxiv.org/pdf/2003.00547.pdf.

[91] Shaohui Liu Ze Yang, Liwei Wang Han Hu, and Stephen Lin. "RepPoints: Point set representation for object detection." In: 2019. DOI: https://arxiv.org/abs/1904.11490.

[92] Wen L Zhang S and Li SZ Lei Z. "RefineDet++: single-shot refinement neural network for object detection." In: *IEEE Trans Circuits Syst Video Technol 31(2):674687*. 2020. DOI: 10.1109/TCSVT.2020.2986402.

[93] Sheng T Zhao Q et al. "M2det: a single-shot object detector based on multi-level feature pyramid network". In: *Proceed AAAI Conf Artif Intell 33:92599266*. 2017.

[94] Chunhua Shen Zhi Tian and Tong He. Hao Chen. "FCOS: Fully convolutional one-stage object detection." In: 2019. DOI: arXiv:1904.01355.

[95] C. Lawrence Zitnick and Piotr Dollťar. "Edge Boxes: Locating Object Proposals from Edges". In: *Microsoft Research* (). URL: https://pdollar.github.io/files/papers/ZitnickDollarECCV14edgeBoxes.pdf.

Acknowledgments

I want to express my gratitude towards my supervisor, Loris Nanni, for his mentorship and unwavering support throughout this journey. My heartfelt thanks go out to my classmates who stood by me and assisted in various projects. Lastly, thank you to my family, whose faith in me never wavered and who were always there by my side.