UDC 004.932:519.652
DOI: 10.20535/SRIT.2308-8893.2025.4.05

OVERVIEW OF NEURAL NETWORK OBJECT DETECTION METHODS & MODELES ON THE EXAMPLE OF THEIR USE FOR LAB ANIMAL OBSERVATION

M.A. SHVANDT, V.V. MOROZ

Abstract. This article provides a brief overview of a set of the most common basic object detection neural network models. Today, the need for automating surveillance and observation processes remains a growing trend. Moreover, one of the key tasks of such processes is usually the detection of an object of interest for further analysis. Previously, many basic object detection algorithms and approaches have been proposed; however, most of them typically have limitations in terms of their applicability. In most cases, these limitations arise due to the nature of the observed environment or because the detection approaches rely on specific object characteristics, such as color or basic shapes only. To address these problems, a new approach for object detection has been developed using neural networks. This paper presents the basis and central aspects of the most common neural network object detection models. The experiment has demonstrated the features, advantages, and disadvantages of the studied methods in the application case of lab animal detection during their behavioral study. Considering this, conclusions and recommendations for their usage cases were made.

Keywords: object detection, neural network, neural layer, architecture, model, optimization, estimation, prediction, video, image, frame, background, foreground, experiment, comparison.

INTRODUCTION

Since the 1990s the fast advancement of computers alongside with the strong development of computer sciences has led to wide automatization of many everyday processes and procedures of our life. From that time and up until today the visual analysis [1] became one of the most used technologies and it is applied everywhere from pedestrian and traffic control to war operations and factory production. In general object detection and tracking usually play important roles in visual analysis. The tasks usually require to detect some object of interest and to track it consequently from frame to frame on either prerecorded video or from some life streaming directly in order to perform some analysis of that object or its behavior.

Object detection and object tracking generally are two separate tasks that require its own special approaches and methods. While some common basic object detection and tracking methods had already been considered in previous articles [2; 3], this time we will take a look on more complex way of object

detection itself. As already mentioned in many cases it can be necessary to detect a specific type of object that has both specific colors and shape/structure. Searching for it with such approaches as, for example, the template matching [2] is not a good option because such operation does not work well with different object scaling and rotations since it requires multiple comparisons of given template with its multiple scaling and rotations in order to find the best matches with objects on image. Such search is not very efficient in terms of performance per frame and is unlikely to be used especially on live video streams of high resolution. Also if object tends to change its shape from frame to frame even slightly, it will affect the comparison with the template and probably will require more templates to check to match each "new" shape. This approach also works mainly with object shape, thus color checking remains as second problem to solve with this approach. Neural detector can come in handy in such cases, as the model coefficients can be trained to recognize a multiple shapes and colors of some particular object. In general the only possible difficulty here can be providing a good dataset for training as it should contain images where the desired type of object can be clearly seen and not mismatched with the background.

One of many processes that can require surveillance and observation automatization is biological research. It often involves the study of life processes of multiple lab animals, for example mice and fish [2; 3]. Usually animals are put in specific conditions so they are easier to observe and note on their behavior. But doing it manually is a time-consuming process that can be automated to save lab personnel some time. The particular case of animal study is gobies behavior observation (Fig. 1). The gobies are kept in a square aquarium with a camera placed right above it recording all their movements during the day to understand the aspects of their activity. While the development of a complex tracking program for its tracking and behavior study is currently being developed, it requires an object detector based on a neural network in order to enhance fish position localization. At this stage in order to choose the best performing detector from the set of most common open-for-use model an experiment was carried to learn which model suits most as such object finder so it can be later integrated into main detection and tracking algorithm. The experiment showed their algorithmic aspects, advantages and disadvantages in case of their application in such test conditions like ours. This analysis might be useful for anyone who plans to use these models in similar conditions as ours and is presented further in this paper.

a *b*

Fig. 1. Lab fish (gobies) in the study environment (*a*, *b*)

THE PROBLEM OF NEURAL NETWORK OBJECT DETECTION

The selected models were chosen according to two main criteria. The first is the hardware requirements in terms of performance. The usage of many computational

algorithms on practice is often limited by the hardware it is running on. As fish video analysis is intended to be performed on-site the resulting detection and tracking algorithm should be able to deliver fine performance on usual mass-market hardware instead of cloud servers or big mainframes.

The second criteria also decided to be considered is the possibility to train the model on local mass-market hardware as well. During the experiment (is shown later in this article) it was found out that some model versions could not be trained locally with minimum sufficient image batch size. As for the model acceptable performance it was decided that the batch size of 1 or just 2 images could lead to model poor training.

The third criteria is model detection speed vs accuracy ratio. The models' mAP was evaluated previously [4] with COCO evaluator [5; 6]. While the accuracy is an important feature, the detection speed is also very significant. Since each frame will be additionally preprocessed to remove noise and enhance other color characteristics which will take additional time, running detection on a single frame should not exceed some reasonable time limits as overall video processing should not become several times longer as the video itself. Considering it we took the model versions with highest claimed accuracy that did not exceed the detection time threshold of about 100–110 ms.

An additional difficulty of this experiment is that the objects of interest are gobies which being filmed as mentioned above do not visually contain many significant marks or features compared to other object like cars, other bigger animals or buildings. Thus the lack of visually distinguishable features makes both network training and usage more challenging.

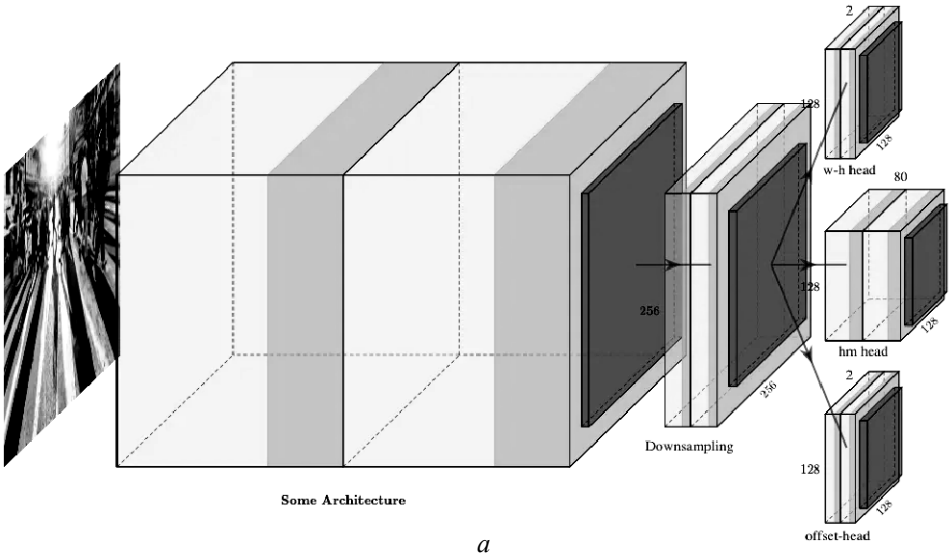
CenterNet architecture. The first considered model architecture is the *CenterNet***.** In order to estimate a bounding box of the searched object and to classify it there are two approaches in the Anchor Free Object Detection: the Keypoint-based approach and the *Center-based approach.* The Keypoint-based approach assumes the network predicts the predefined key points and then they are used for bounding box generation around the object and its classification. Examples of such architectures are CenterNet: Keypoint Triplets [7], CornerNet [8], Grid-RCNN [9]. The Center-based approach [7; 10; 11] uses center-point or any part-point of an object to define positive and negative samples. Then it predicts the distance from these positives to four coordinates for the generation of a bounding box. For example such methods as DenseBox [12], FCOS [13], etc. generate positive samples and use them for estimation of boxes and class probabilities.

As for the CenterNet, the main research [10] treats the center of a box as an object as well as a key point. Then it uses this predicted center to detect the coordinates/offsets of the bounding box. Thus the center prediction task is considered as a standard problem for keypoint estimation. When image gets passed through Fully Convolutional Network, the final feature map provides as an output heatmaps for different key points. The peaks of these output feature maps are considered as predicted centers. The network also makes predictions of the width and height of the box for these centers with each center having its unique box width and height. This binding is intended for removing of the Non-Maximal Suppression step in post-processing. The heatmap peaks are also linked to a particular class to which it belongs to and thus it allows object classification, as using these centers, dimensions, and class probabilities, object detection task is achieved.

In general, the CenterNet architecture works in the following way. The input image I having width and height as W and H respectively, and 3 channels for RGB. R is an output stride that will set the resulting dimensions of the given

heads. All the heads will have the same height H/R and width W/R, but they will have different C values (depth of the keypoint heatmap). Thus the final head dimensions are (W/R, H/R, C=[<*Classes Num*>/<2>/<2>]). Thus if input dimensions are 512×512, then head dimensions are 128×128 considering stride $R = 4$ (Fig. 2, *a*). The three heads as shown in Fig. 2, *a* are *Heatmap Head, Dimension Head, Offset Head*.

Heatmap Head is used for the key points estimation of the given input image. In the case of object detection, keypoints are the box center. One has to predict heatmap $\hat{Y}$ of dimensions (W/R, H/R, C), with R being the output stride, C is the number of classes; $\hat{Y}$ is the functssion of *x, y, c.* A prediction $\hat{Y}(x, y, c) = 1$ corresponds to detected center for that particular class *c*. $\hat{Y}(x, y, c) = 0$ is considered as background. For the loss propagation ground truth heatmaps calculation, these centers are splat using Gaussian Kernels after converting them to low-resolution equivalent (division by stride R, denoted as $\tilde{p}$). For example, in case of three classes $C = 3$ and input image dimensions of 400×400, with a given stride $R = 4$ it is necessary to generate 3 heatmaps (as each heatmap corresponds to a given class) of 100×100 dimension as shown in Fig. 2, *b*. The σ value used in the kernel is the object-size adaptive standard deviation. Also, if two gaussians of the same class are overlapping, they take element-wise maximum to find the target class.



a

b

Fig. 2. a — Three heads are predicted after one forward pass from the network architecture: Offset Head, Heatmap Head. Dimension Head. Here Some Architecture (FCN) referees to any of the feature extractors which we want to use (specified heads are for object detection; *b* — Left: Ground Truths of different classes, shown in different colors; Right: three centers of respective classes splat into heatmaps using Gaussian Kernel) (source: medium.com/visionwizard [11])

$$Y_{xyc} = \exp\left(-\frac{(x-\tilde{p}_y)^2 + (y-\tilde{p}_y)^2}{2\sigma_p^2} \right).$$

Dimension Head is used for the estimation of the dimensions of the boxes width and height. With given box coordinates (x_1, y_1, x_2, y_2) of object *k* and class *c***,** one can regress object sizes $s_k = (x_2 - x_1, y_2 - y_1)$**.** This is achieved by solving a standard L_1 distance norm. Dimensions of this heatmap are (*W*/*R, H*/*R,* 2), with *w* being *h* are predicted width and height of the box. To reduce the amount of computation, single sized heatmaps for all object categories are used. *Offset Head* is used to recover from the discretization error caused due to the downsampling of the input. After the center points prediction, one has to map these coordinates to an input image of higher dimension. Since the original image pixel indices are integer values this will cause a value disturbance because one will be predicting the float values. So to solve this issue they make predictions the local offsets $\hat{O}$, as these local offset alues are shared between objects on an image. Offset Head dimensions are **(***W*/*R*, *H*/*R*, 2**)** (here *x* and *y* are the coordinate offsets). The overall detection flow can be seen on Fig. 3, *a*, *b*.

As a *Feature Extractor* CenterNet can use a variety of backbone/feature extraction approaches [8; 10; 11]. With our research we have considered the following ones: *Stacked Hourglass Network* [14] (Hourglass104 version), *Residual Network* [15] (ResNet101 V1 FPN) and the *MobileNet* [16] (MobileNet V2/V1 FPN). For instance the stacked Hourglass Network downsamples the input by 4×, then followed by two sequential hourglass modules, with each hourglass module being made up of a uniform chain of 5-layer down- and up-convolutional network with skip connections. The original paper [10] also used modified (Fig. 3, *c*) ResNet18**,** ResNet1, Deep Layer Aggregation Networks (DLA) [17] with added Deconvolutional and Deformable Convolutional Layers. Standard ResNet modules were extended with three transposed convolutional networks to incorporate higher resolution outputs. Some modifications were done by reducing the output upsampling layers' filters of to 256, 128, and 64 respectively in order to reduce computation. The authors also added an additional 3×3 deformable convolutional layer between each of upsampling layers led to better results on some standard datasets [10; 11].

The main part of CenterNet algorithm is *Loss Calculation and Propagation*. After heatmaps are generated by the network there is a task of loss propagation for training stabilization. In the original paper [8], authors use several loss functions to get over and balance the bias between the training of different heads**.** There are three Loss functions mentioned: *Heatmap Variant Focal Loss, L1 Norm Offset* Loss and *L1 Norm Dimension Size Loss.* For *Heatmap Variant Focal Loss* the Focal Loss function [18; 19] is divided into two parts of positive and negative samples:

$$L_k = \frac{-1}{N} \sum_{xyc} \begin{cases} (1-\hat{Y}_{xyc})^{\alpha} \log(\hat{Y}_{xyc}) & \text{if} \quad Y_{xyc} = 1, \\ (1-\hat{Y}_{xyc})^{\beta} (\hat{Y}_{xyc})^{\alpha} \log(1-\hat{Y}_{xyc}) & \text{otherwise}. \end{cases}$$

If $Y = 1$ when predicted $\hat{Y}$ is close to 1 (ex. $\hat{Y} = 0.95$), it considers as an easy *example (well-classified example)* and thus by the logic of Focal Loss the weightage of the propagated loss will be decreased. The same logic is for *hard examples (misclassified example)* with a difference that instead of decreasing the weight, it will increase the slope of the value by parameter α (here $\alpha := 2$). If

Y != 1 (Otherwise) with predicted $\hat{Y}$ being very close to 0 (ex. $\hat{Y} = 0.005$), then $\hat{Y}^{\alpha}$ will cause the overall loss to be zero, and less weight will be assigned to the propagated loss as stated in the premise of Focal Loss [18]. The particular case is when $\hat{Y}$ is not very close to 0 and has a value near to 1, but it is in the neighborhood of the ground truth heatmap. As the ground truths are the gaussian kernel outputs there is no sudden drop in the values near Y=1. It considers values, lying inside gaussian outputs, as possible positives. This is an advantage of this loss. For example, let $\hat{Y} = 0.9$ and being near to the center point peak of ground truth. Here a misclassification takes place as the value should be very near to 0 according to simple logistic regression loss logic. But, as predicted $\hat{Y} \approx 1$, the propagated loss will be less weighted even in a condition of misclassification as

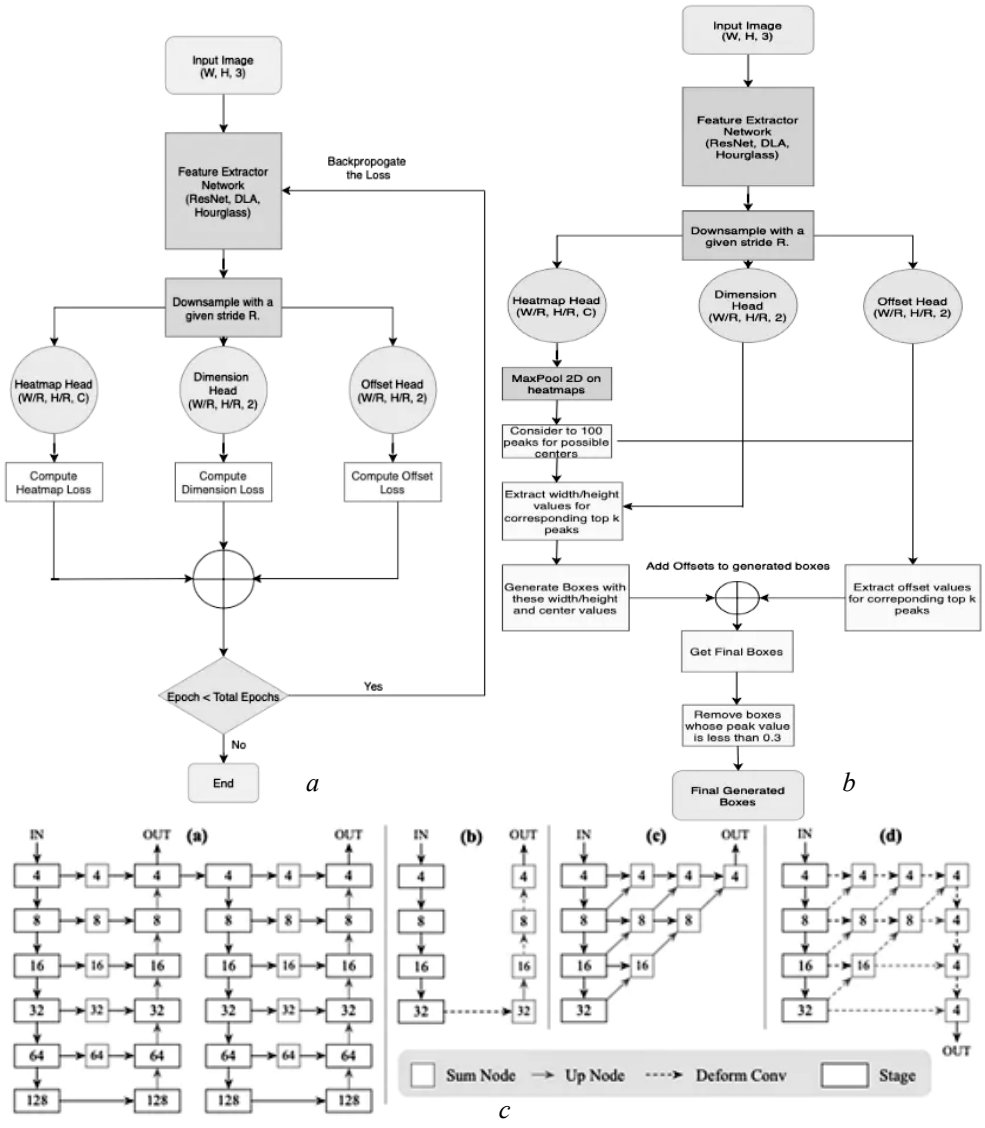


Fig. 3. Training — *a*, *b*; Inference flowchart of the explained object detection algorithm — *c*; (here: (a) Stacked Hourglass Network, (b) ResNets with Transposed and Deformable Convolution layers, (c) Original DLA-34, (d) Modified DLA-34 by adding skip connections and Deformable Convolutional Layers.) (source: medium.com/ visionwizard [11])

the loss will be compensated according to the term $(1-Y)^{\beta}$ (value of Y will be close to 1 in a region near center peak). In case when one has $\hat{Y} = 0.9$ and being far from the center point peak, in this condition of misclassification a large loss will be propagated according to term $(1-Y)^{\beta}$ as it does not placed in that splatted region, and the value of Y will be very close to 0. Here $\beta := 4$. The design of this loss function helps to increase the number of positive examples by considering the heatmap values generated by gaussian kernels which, in its turn, help to decrease the bias between positives and negatives.

The *L1 Norm Offset Loss* is a simple L1 Norm of the predicted offset $\hat{O}$ and the ground truth offset values:

$$L_{off} = \frac{1}{N}\sum_{p}\left|\hat{O}_{\tilde{p}} - \left(\frac{p}{R} - \tilde{p}\right)\right|.$$

The meaning of ground truth offset values can be seen on the example: if there is a center point at (18, 22) in an original high-resolution image, when downsampled, with stride size of 4, the mapped coordinates will be (4, 5) on a low-resolution feature map. Here is an offset error of 0.5 in both cases. In the case of keypoint estimation, it becomes important to handle this problem as keypoints are very position sensitive. In order to solve this task the offset loss function is added in order to obtain more accurate results. This supervision only acts at the position of key points, all other locations are ignored.

The *L1 Norm Dimension Size Loss* of the predicted and ground truth width-height coordinates is used for Regression of the width and height of bounding boxes. Here $\hat{S}$ are the predicted dimensions and s are actual ground truth sizes. Raw pixel values are used to calculate the loss instead of normalizing with the feature map size

$$L_{size} = \frac{1}{N}\sum_{k=1}^{N}\left|\hat{S}_{p_k} - s_k\right|.$$

Total Loss propagated by the network is shown in formula, $\lambda_{size} = \dfrac{0.1}{\lambda_{offset}}$.

The *Total Loss* of CenterNet: $L_{\det} = L_k + \lambda_{size}L_{size} + \lambda_{off}L_{off}$.

The example of object detection process is visualized on Fig. 4: during inference process one calculates the peaks of the heatmaps by finding the maximum value near the 8-pixel neighborhood in a heatmap and keeping the first 100 peaks of all the different classes independently. It is achieved by 3×3 MaxPool opera-

Fig. 4. Left: keypoint heatmap; middle: keypoint offsets; right: dimensions of box (source: original paper [10])

tion on the resulting feature map with the obtained peak coordinates being used to calculate the dimensions and offset predictions.

The Residual Network. As mentioned earlier, CenterNet can use several different backbone architectures for feature extraction. We have considered the models using *Residual Network*, *MobileNet* and *Stacked Hourglass Network*. The first one used in studied models is the *Residual Network* architecture [15; 20; 21]. The Residual Network architecture itself is based on a concept of Residual Blocks. These blocks were designed to address the issue of the *vanishing/exploding gradient*. Inside ResNet a method known as skip connections is applied. This method skips (bypasses) some levels in between link-layer activations to subsequent layers and thus creates a leftover block (Fig. 5, *a*). These leftover blocks are used in stacks to create residual nets (Fig. 6). The main idea of such architecture is to let the network fit the residual mapping instead of having layers learn the underlying mapping and thus, let the network fit instead of using, for example, the initial mapping of $H(x)$:

$$F(x) := H(x) - x \Rightarrow H(x) := F(x) + x\,.$$

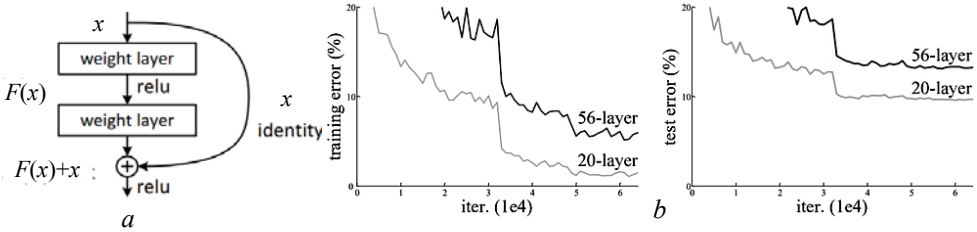


Fig. 5. a — Skip connection or Shortcut; *b* — comparison of 26-layer vs 56-layer architecture (source: medium.com/siddheshb008 [20], original paper [15])

Thanks to this skip link, the regularization will skip any layer that worsens the architecture performance and as the training of a very deep neural network becomes possible without getting issues with vanishing or expanding gradients. The general purpose of ResNet is following: in the Deep Neural Networks extra layers are stacked in order to improve accuracy and performance, often to handle a challenging problem. The main idea of layering is that by adding additional layers they will eventually learn features that are more complicated. Ex an example one can take photographs recognition: when recognizing photographs, the first layer may pick up on edges, the second — textures, the third — objects, and so on. However, the traditional convolutional neural network model was found to have the maximum depth threshold. The graphic (Fig. 5, *b*) shows the percentage of errors for training and test data for a 20-layer network and a 56-layer network, respectively.

In both the training and testing situations, we see the higher error percentage for a 56-layer network in comparison with a 20-layer network. It demonstrates that adding additional layers on top of the network will decline its performance. This might be because of with the initialization of the network, the optimization function, and most significantly — because of the vanishing gradient problem. In this case overfitting is not the issue as the 56-layer network's error percentage is the worst on both training and test data, and it does not happen when the model is overfitting.

The ResNet architecture exists in several configurations. For example it can use the VGG-19-inspired 34-layer plain network architecture that is followed by the addition of the shortcut connection. It is subsequently transformed into the residual network by these short-cut connections, as depicted in Fig. 6. Keras, an open-source, Python-based neural network framework offers ResNet V1 and ResNet V2 with 50, 101, or 152 layers: ResNet50, ResNet101, ResNet152, ResNet50V2, ResNet101V2, ResNet152V2; ResNetV2 and the original ResNet (V1) vary primarily in that V2 applies batch normalization before each weight layer. As a brief conclusion, it can be said that ResNet is a robust backbone model and can be used in various computer vision tasks.

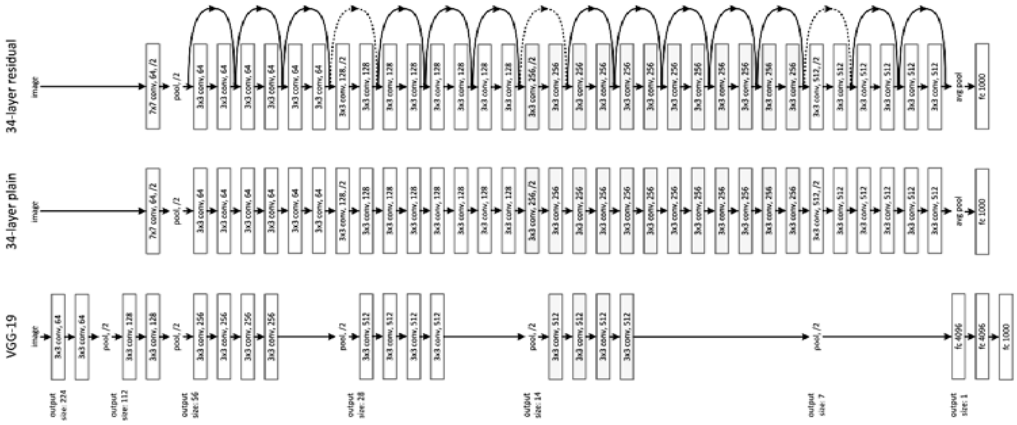
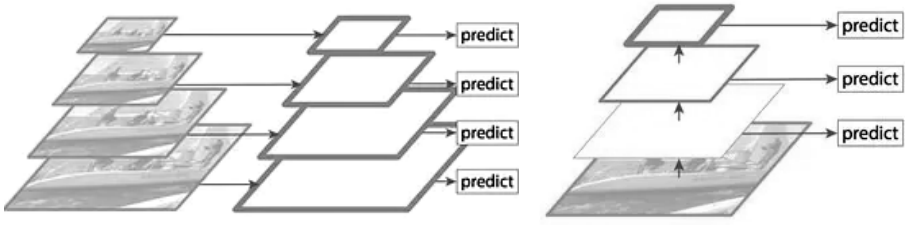


Fig. 6. Resnet34 Architecture (source: medium.com/siddheshb008 [20], original paper [15])

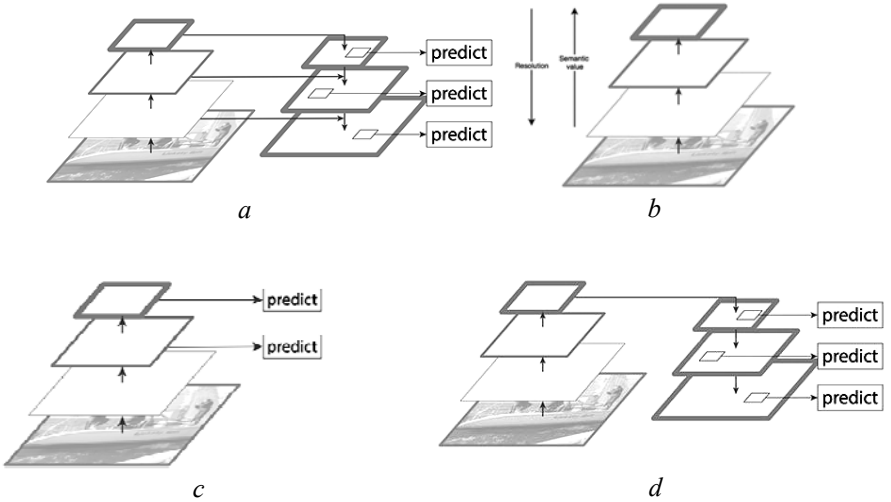
Feature Pyramid Network (FPN). Object detection in different scales is a very difficult task, especially for small objects. The pyramid of the same image at different scale can be used to detect objects (Fig. 7, *a*) but processing multiple scale images too demanding in terms of time and memory to be trained end-to-end simultaneously. That is why it can be used only in inference in order to increase accuracy as high as possible, in particular for cases, when speed is not a concern. As an alternative a feature pyramid can be created and used for object detection (Fig. 7, *b*), but the feature maps, which are closer to the image layer, are composed of low-level structures that are not good for accurate object detection. The Feature Pyramid Network itself is a feature extractor [22–25] created for the described pyramid approach that considers performance speed and accuracy as well. It is used instead of such the feature extractor as Faster R-CNN [26] and generates multi-scale feature maps (multiple feature map layers) and delivers information of better quality than the regular feature pyramid for object detection.



a *b*

Fig. 7. a — pyramid of images; *b* — pyramid of feature maps (source: medium.com/jonathan-hui [23], original paper [22])

The FPN data flow composes of a *bottom-up* and a *top-down* pathway (Fig. 8, *a*). The bottom-up pathway is represented by a usual convolutional network. During it the features are being extracted: as one goes up, the spatial resolution decreases. After detecting more high-level structures, the *semantic value* for each layer increases (Fig. 8, *b*). The Single Shot MultiBox Detector (SSD) [27] calculates detection from multiple feature maps, but it does not select the bottom layers for object detection (Fig. 8, *c*). Despite being in high resolution their semantic value is not high enough to use it as the speed slow-down is significant. Because of that the SSD uses only upper layers for detection and thus its performance is much worse for small objects. The FPN uses a top-down pathway to construct higher resolution layers from a semantic rich layer (Fig. 8, *d*).



a *b* *c* *d*

Fig. 8. a — FPN data flow; *b* — feature extraction in FPN; *c* — SSD object detection with top levels; s*d* — FPN top-down pathway (source: medium.com/jonathan-hui [23]; original paper [22])

The reconstructed layers are semantically strong but the objects are not located precisely after all the downsampling and upsampling operations. In order to enhance object location prediction, the lateral connections between reconstructed layers and the corresponding feature maps were added. It also helps to simplify the training as it also acts as skip connections. Similar approach is used in ResNet [15]. For the bottom-up stage the ResNet is used. The bottom-up pathway consists of many convolution modules $Conv_i$, $i=[1,5]$, with each module composing of multiple convolution layers. Also during this stage, one reduces the spatial dimension by 1/2 (i.e. double the stride) with labeling the output of each convolution module as C_i which are later used during in the top-down pathway (Fig. 9, *a*). In the process of top-down pathway one applies a 1×1 convolution filter in order to reduce C_5 channel depth to 256-d to obtain M_5. Thus, one receives the first feature map layer that will be used for object prediction. With each step down further one upsamples the previous layer by 2 using nearest neighbors upsampling. Again a 1×1 convolution is applied to corresponding feature maps and then they are added element-wise. A 3×3 convolution is applied to all merged layers; this convolution filter is used for reducing the aliasing effect during merging operation with the upsampled layer (Fig. 9, *b*). The same process is repeated for the pyramid feature maps P_3, P_2, but it is stopped at P_2 because the spatial dimension of C_1 is too large. If it is

continued, it will slow down the process too much. As one shares the same classifier and box regressor of every output feature maps, all pyramid feature maps P_5, P_4, P_3, P_2 have 256-d output channels.

As for object detection, FPN is not an object detector by itself. This architecture is a feature extractor that works with object detectors. It is used for feature maps extracting and later feeding them into some detector, for example Region Proposal Network (RPN). RPN then applies a sliding window over those feature maps to predict the objectness (i.e. whether there is an object or not) and object boundary box at each location (Fig. 9, *c*). In the FPN framework, for each scale level, for example P_4 or P_3, one applies 3×3 convolution filter over the feature maps and after that applies separate 1×1 convolution for predictions of objectness and boundary box regression. These 3×3 and 1×1 convolutional layers are called the RPN *head* (Fig. 9, *d*).

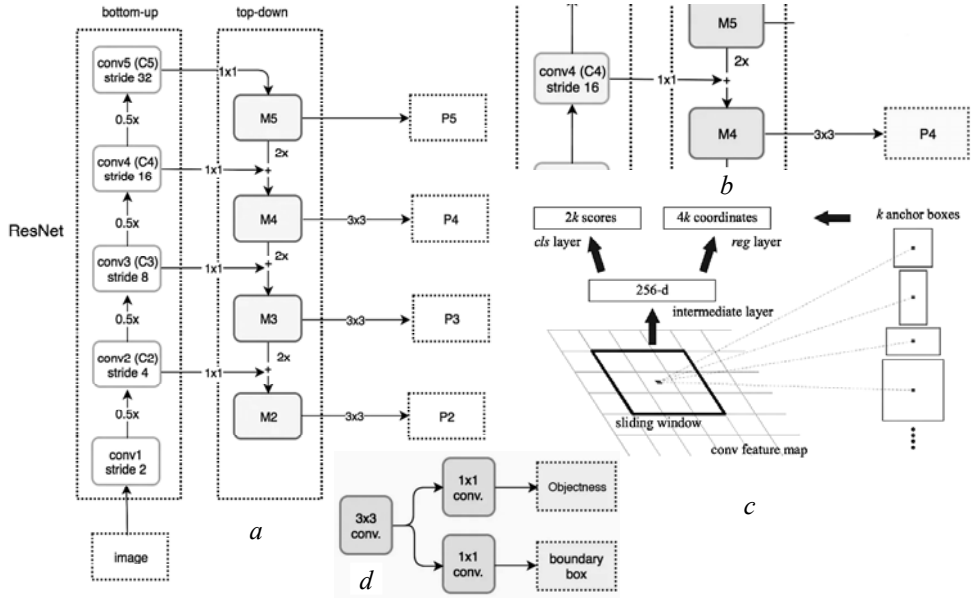


Fig. 9. a — ResNet for FPN bottom-up and top-down pathways; *b* — feature merging operation during top-down pathway; *c* — FPN usage with RPN; *d* — RPN head (source: medium.com/jonathan-hui [23])

MobileNet architecture. The third considered architecture is MobileNet. This architecture was developed by Google in 2017 [16; 28]. It utilizes the approach called *Depthwise Separable Convolution* in order to reduce the model size and complexity. This architecture was primarily created for use in mobile and embedded vision applications (Fig. 10, *a*). It has following benefits: smaller model size (fewer number of parameters) and smaller complexity (fewer multiplications and additions, aka Multi-Adds). To make MobileNet easy to tune, two parameters were introduced: *Width Multiplier* α and *Resolution Multiplier* ρ.

The *Depthwise separable convolution* is a depthwise convolution that is followed by a pointwise convolution (Fig. 10, *b*); the Depthwise convolution is the channel-wise $D_K \times D_K$ spatial convolution. For example, if one has 5 channels, then there are 5 $D_K \times D_K$ spatial convolutions**.** The *Pointwise convolution* actually is the 1×1 convolution intended to change the dimension. Combined with the Depthwise Convolution, the operation cost is:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F ,$$

where the left part of sum is the Depthwise Convolution Cost and the right one is the Pointwise Convolution Cost. Here M is the number of input channels, N is the number of output channels, D_K is kernel size, D_F is the feature map size. For Standard Convolution, its cost is

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F .$$

Thus, the Depthwise Separable Convolution Cost / Standard Convolution Cost is:

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} .$$

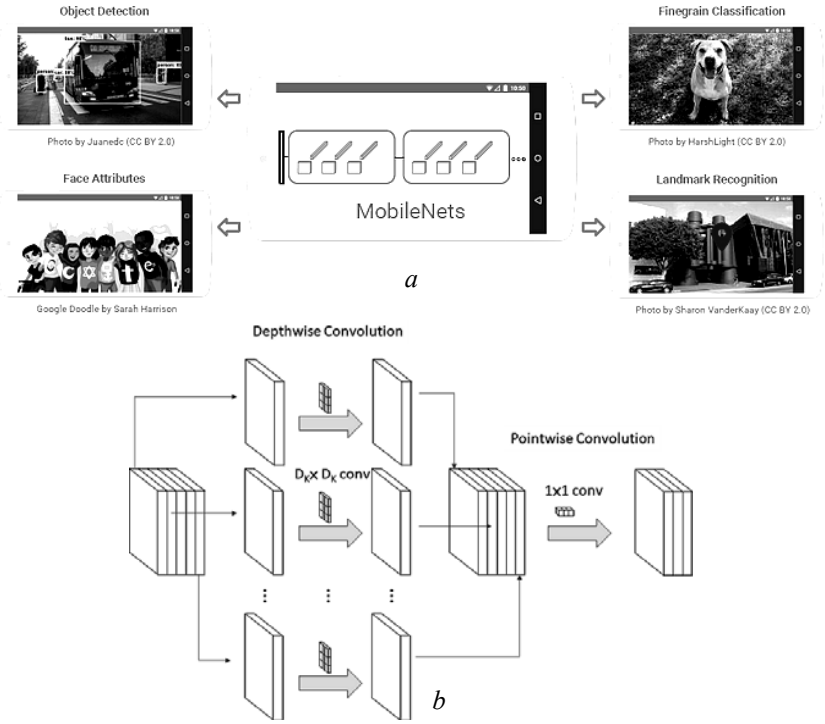


Fig. 10. a — MobileNets usage in practice; *b* — Depthwise separable convolution (source: towardsdatascience.com; original paper [16; 28])

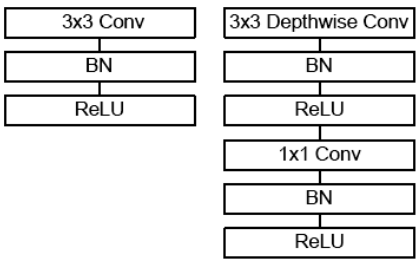


Fig. 10. Left: Standard Convolution, right: Depthwise separable convolution (Right) With BN and ReLU (source: original paper [16])

When $D_K \times D_K$ is 3×3, the amount of computation can be reduced from 8 to 9 times, but with only small reduction in accuracy. The Table 1 shows the architecture of MobileNet; the Batch Normalization (BN) and ReLU are applied after each convolution (Fig. 11), with Width Multiplier α being introduced for controlling of the number of channels or channel depth**,** which makes M become αM. Thus, the Depthwise Separable Convolution cost (with Width Multiplier α) is:

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F ,$$

where $\alpha = [0,1]$, with typical settings of 1, 0.75, 0.5 and 0.25. With $\alpha = 1$, it is the basic MobileNet, and the computational cost and the number of parameters can be reduced quadratically by $\approx \alpha^2$. The Resolution Multiplier ρ is introduced to control the input image resolution of the network and thus the Depthwise Separable Convolution Cost with Both Width Multiplier and Resolution Multiplier is:

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F ,$$

with $\rho = [0,1]$ and the input resolution of 224, 192, 160, and 128. With $\rho = 1$, it is the basic MobileNe.

T a b l e 1. MobileNet Body Architecture (source: original paper [16])

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1 Conv / s1	3 × 3 × 512 dw 1 × 1 × 512 × 512	14 × 14 × 512 14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

Later the modified MobileNet version was introduced, called V2 [29; 30]. It utilizes the inverted residual structure. In this modification the non-linearities in narrow layers are removed. The difference between V1 and V2 can be briefly described in the following way. The MobileNet V1 has 2 layers, with the first layer, called depthwise convolution, performing lightweight filtering by applying a single convolutional filter per input channel, and the second layer, called pointwise convolution, being a 1×1 convolution and used for building new features through calculating linear combinations of the input channels.

The MobileNet V2 has two types of blocks. One is residual block with stride of 1 and another one is block with stride of 2 used for downsizing; the model has 3 layers for both types of blocks. In this version the first layer is 1×1 convolution with ReLU6, the second layer is the depthwise convolution and the third layer is another 1x1 convolution but without any non-linearity (Table 2). It is also claimed that with using ReLU again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.

Table 2. MobileNet V2 layers (source: original paper [29])

Input	Operator	Output
$h \times w \times k$	1×1 conv2d, ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3×3 dwise s=s, ReLU6	$h/s \times w/s \times (tk)$
$h/s \times w/s \times tk$	linear 1×1 conv2d	$h/s \times w/s \times k'$

There is also an expansion factor t. The authors took $t = 6$ for all main experiments [29; 30]. If the input got 64 channels, the internal output would have $64 \times t = 64 \times 6 = 384$ channels. Table 3 demonstrates the MobileNetV2 Overall Architecture, with t being the mentioned expansion factor, c — number of output channels, n — repeatingnumber, s — stride size; for the spatial convolution 3×3 kernels are used. The authors also note that with the removal of ReLU6 at the output of each bottleneck module, accuracy is improved (Fig. 12, *a*), and with shortcut between bottlenecks, it outperforms shortcut between expansions and the one without any residual connections (Fig. 12, *b*) [29; 30].

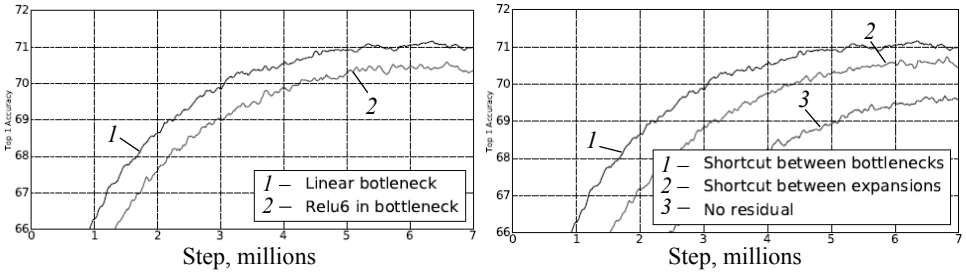


Fig. 12. a — Impact of Linear Bottleneck; *b* — impact of Shortcut (source: original paper [29])

Table 3. MobileNetV2 Overall Architecture (source: original paper [29])

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1×1	-	-	1280	1
$7^2 \times 1280$	avgpool 7×7	-	-	-	1
$1 \times 1 \times 1280$	conv2d 1×1	-	k	-	-

The Hourglass architecture. The so-called *Hourglass Network* is an architecture that combines a contracting path to extract information and an expanding one to map features into locations [31; 32]. The idea behind its name is that the union of the two paths is usually seen as an hourglass, where information gets narrower before getting expanded again. This architecture takes its beginning from *Fully Convolutional Networks* (FCNs) [34]. Presented in 2015, it is aimed at

modifying the typical structure of Convolutional Neural Networks (CNNs) to obtain segmentation maps as output. As one knows, CNN is a deep network that uses 2 operations: convolution and pooling. A convolution is a mathematical function that, through the use of filters, can extract the presence of different (learned) features in an input. The pooling operation is used to reduce the size of the convoluted matrix, condensing the extracted information (Fig. 13).

The basic structure of a CNN can be seen on Fig. 14. Here convolutions and pooling operations are applied in sequence, resulting in fully-connected layers for the classification of the received input [35]. The FCNs are intended to replace the final fully-connected layers with upsampled versions of the pooling layers output, and thus to retain spatial information and map them into the original input [34]. The FCN also employs the upsampled version of the last pooling layer and combines different upsampling of various pooling layers in the network. Thanks to this the outputs of deeper layers, which contain more information about features, can be combined with the outputs of early layers, which still have information about location [34]. Thus the output of the network's final layer is a segmentation map which is built on information from several previous layers, instead of just the final one. However, this approach is only the basis for hourglass networks. More deeply, the idea of the hourglass architecture can be seen on the following models.

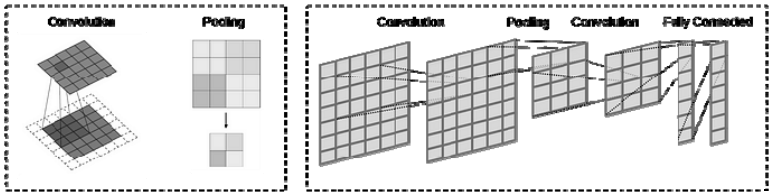


Fig. 13. Convolution and Poling operations (source: medium.com/@calleris.enrico [32])

One of further developments of FCNs is an *U-Net* architecture [36]. It is usually considered the earliest example of an hourglass network and was Initially developed for the biomedical images segmentation. U-Net's approach uses the FCN concept to achieve impressive performance, with the main difference between U-Net and FCNs being that the structure of the upsampling operations, which is not a single one anymore but it matches the length of the downsampling path. Now these two paths are now symmetrical and actually lead to model being called U-Net because of the network shape (Fig. 15, *a*). In this architecture after a downward path with the classic CNN structure of sequential convolution-pooling operations, the upward path upsamples the received input and concatenates it with the corresponding layer of the downward path (similar as FCN does). In each upward layer, the previous output gets upsampled and de-convoluted. Then it is combined with the output of the corresponding layer in the downward path (as

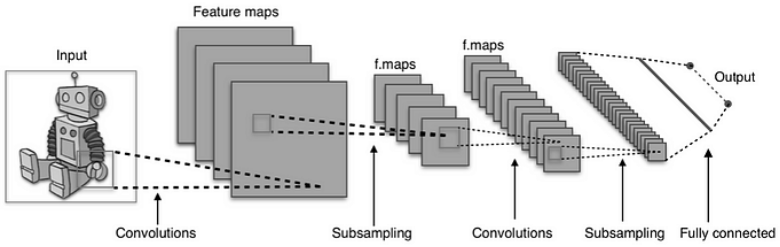
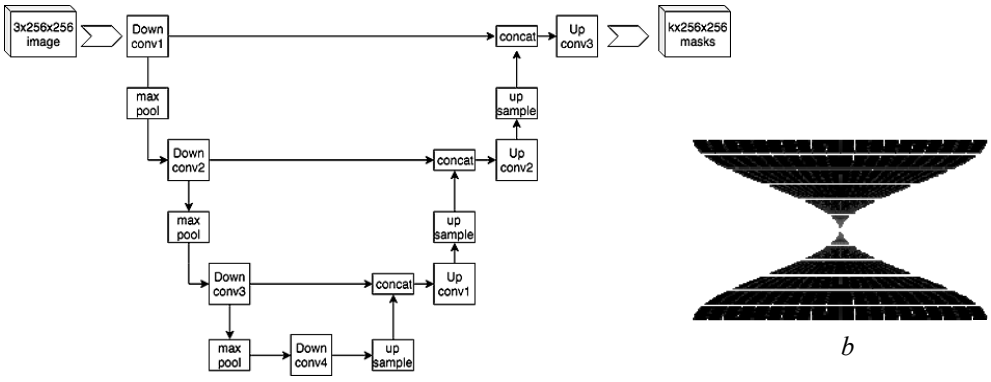


Fig. 14. Basic structure of a CNN: Convolutional Operations with Pooling Operations and Fully-Connected Layers. (source: medium.com/@calleris.enrico [32])

shown by the horizontal arrows in Fig. 15, *a*). Thanks to this operation the feature information extracted by the lower layers can be combined with layers with the more spatially-resoluted outputs of the early convolutional layers. Thus, a complete localized feature map is obtained which gives as final output an accurate segmentation map.

Another variant of an hourglass network is the *V-Net*. This model was introduced in 2016 and was intended for segmentation of 3D medical images [38]. This architecture has quite similar approach as U-Net as it is also based on two symmetric contracting and expanding paths. The main difference is that unlike U-Net it uses a fully-convolutional structure where convolution operations are present exclusively and pooling layers are absent. This approach is two-folded because the pooling operation can easily be replaced by a convolution with a larger stride, and thus the network can be trained faster [39]. Also it would be easier to apply the corresponding upsampling operation in the expanding path as de-convolutions are preferred to un-pooling in order to simplify the understanding and analysis [32; 38]. It is also worth mentioning the difference in the training procedure between U-Net and V-Net: the U-Net uses the classic stochastic gradient descent, while V-Net utilizes residual connections [15] to make convergence faster and improve the segmentation results.

But despite all highlighted differences, U-Net and V-Net still share similar approach, based on two different interconnected paths: the downward (contracting) path for progressive feature extraction from the scene and the upward (expanding) path for mapping of the extracted features to specific locations in the original image [37]. The overall concept shows why it was called a hourglass as it mimicks the two paths that contract and expand, meeting midway to form the hourglass shape (Fig. 15, *b*). The output of this type of architecture is as an accurate segmentation map, and it still is one of the most used approaches in computer vision [40].



a *b*

Fig. 15. a — U-Net architecture with downward path extracting features (left) and an upward path mapping them into locations (right); *b* — conceptualization of the Hourglass Network, with the contracting path (up) and the expanding path (down) (source: medium.com/@calleris.enrico [32])

EfficientDet architecture. EfficientDet is a family of object detection models. As model efficiency is very important in computer vision, a lot of research has been made in recent years towards more accurate object detection [41]. But one knows that the more accurate object detection network is more expensive in terms of number of parameters (FLOPS) it gets. The most simple and straightfor-

ward way to increase the accuracy of object detection network is either to make the network deeper by increasing the number of layers, or increase the number of channels, or increase the model input image resolution. However, the random increase of any one among the dimensions mentioned above will diminish the accuracy gain. So in EfficientDet paper [42] authors introduced a systematic way of model scaling and they show that carefully balancing network depth, width and resolution can lead to better performance.

The initial concept of model scaling, i.e. increase of the network depth, width and resolution to enchance its performance, was presented in the EfficientNet paper [43] for image classification, but in the result of EfficientNet testing the authors implement this technique for object detection and called it as EfficientDet [41; 42]. This architecture is based on the paradigm of one-stage detectors: these detectors use ImageNet-pretrained EfficientNets as the backbone network. Thus, the Bidirectional Feature Pyramid Network (BiFPN) was introduced and it serves as the feature network by taking level 3–7 features (P3, P4, P5, P6, P7) from the backbone network and repeatedly applying top-down and bottom-up bidirectional feature fusion. These fused features are then fed to a class and box network predictor, in order to generate object class and bounding box predictions respectively (Fig. 16).

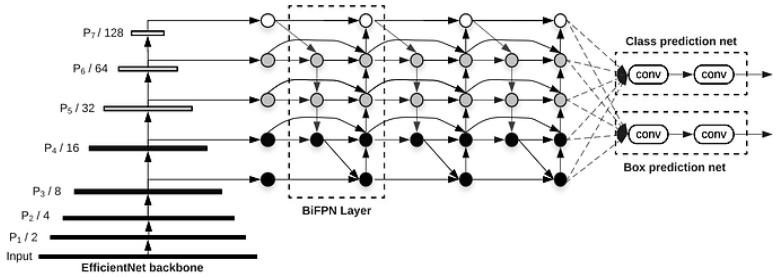
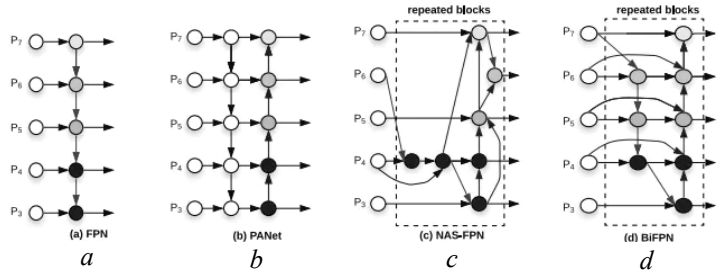


Fig. 16. EfficientDet architecture: it uses EfficientNet as the backbone network, BiFPN as the feature network and shared class/box prediction network. (source: original paper [42])

The development of BiFPN can be seen on Figure 17 [42; 45]. Here, FPN [22] is a baseline way to fuse features with a top down flow; Path Aggregation Network (PA Net) [45] allows the feature fusion to go backwards and forwards from smaller to larger resolution; NAS-FPN [46] is also another feature fusion technique created earlier. The EfficientDet architecture uses the edited structure of NAS-FPN to create on the BiFPN blocks and stacks them on top of each other with number of blocks varying in the model scaling procedure. Also, considering that certain features and feature channels might vary in the amount that they contribute to the end prediction, a set of weights was added at the beginning of the channel that are learnable. Before EfficientDet, model scaling for image detection generally scaled portions of the network independently, as for example, the ResNet scales only the size of the backbone network. This idea is similar to the joint scaling approach used to create EfficientNet. For EfficientDet the scaling task was set to vary the size of the backbone network, the BiFPN network, the class/box network, and the input resolution. The backbone network scales up directly with the pretrained checkpoints of EfficientNet-B0 through EfficientNet-B6 and its width and depth are varied along with the number of BiFPN stacks [44].



a *b* *c* *d*

Fig. 17. Feature network design: *a* — FPN [23] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P3 - P7); *b* — PANet [26] adds an additional bottom-up pathway on top of FPN; *c* — NAS-FPN [10] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; *d* — is our BiFPN with better accuracy and efficiency trade-offs. (source: original paper [42])

The specifications of EfficientDet are the following: the *Backbone Network* uses the same width/depth scaling coefficients as EfficientNet-B0 to B6 so that ImageNet-pretrained checkpoints can be used. The *BiFPN network* width (number of channels) grows exponentially as done in EfficientNets, but the depth (number of layers) is increased linearly since it needs to be rounded to small integers. After a grid search, 1.35 is detected as best scale factor for width:

$$W_{bifpn} = 64 \cdot (1.35^{\varphi}), \quad D_{bifpn} = 3 + \varphi .$$

Box/class prediction network has the same width as the BiFPN but the depth is linearly increased:

$$D_{box} = D_{class} = 3 + \lfloor \phi / 3 \rfloor .$$

As for input image resolution: since feature levels 3–7 are used in BiFPN, the input resolution must be dividable by $2^7 = 128$, so resolutions are increased linearly.

$$R_{input} = 512 + \phi \cdot 128 .$$

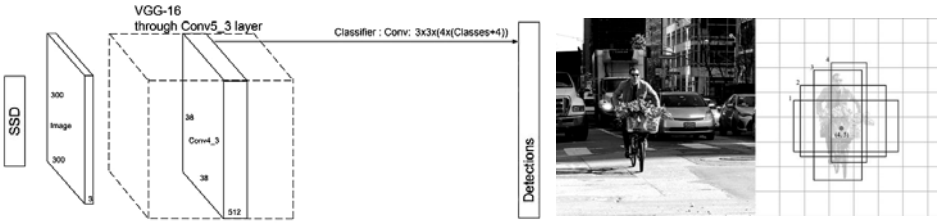
Table 4. EfficientDet scaling (source: original paper [42])

Depth	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7*x*	1536	B7	384	8	5

Single Shot MultiBox Detector (SSD). The SSD [27; 47] is a detector designed for real-time object detection. While Faster R-CNN [26] utilizes an RPN for boundary box creation and uses those boxes for object classification, despite its accuracy it does not perform very fast (up to 7 fps) and thus is not suitable for

real-time object detection. The SSD performs faster due to elimination of need for RPN and maintains fine accuracy by using multi-scale features and default boxes as improvements. This allows SSD to increase its speed using images with lower resolution and keep its performance at the same level of accuracy as Faster R-CNN. This fact was confirmed by our experiment, as shown in the model performance comparison at the end of this paper.

Single Shot MultiBox Detector composes of 2 parts: feature maps extraction and application of convolution filters for object detection. For feature maps extraction the *VGG16* architecture [48; 49] is used and it detects objects with help of *Conv4_3* layer (Fig. 18, *a*). As for example (Fig. 18, *b*), a 8×8 Conv4_3 is drawn spatially (should be 38×38) and for each cell (aka location) it produces 4 object predictions. Each prediction is represented by a boundary box and 21 scores for each class (plus 1 extra class for no object) and one picks the highest score as the bounded object's class. The Conv4_3 in total produces 38×38×4 predictions, four predictions per cell regardless of the depth of the feature maps. Since many of these predictions contain no object, SSD reserves a class "0" to mark that the box has no objects (Fig. 19, *a*). The process of making multiple predictions containing boundary boxes and confidence scores is called multibox [47].



a *b*

Fig. 18. a — General SSD architecture idea; *b* — detector work: left: the original image, right: 4 predictions at each cell (source: article [47])

As for predictors for object detection, SSD does not use a special RPN; instead, it calculates both the location and class scores with help of *small convolution filters*. After feature maps extracting, the detector applies 3×3 convolution filters for each cell to make predictions with these filters calculating predictions in the same way as regular CNN filters. The output of each filter consists of 25 channels: 21 scores for each class + one boundary box (ex. applying four 3×3 filters in Conv4_3 for 512 input channels mapping gives 25 output channels) (Fig. 19, *b*).

$$(38 \times 38 \times 512) \xrightarrow{(4 \times 3 \times 3 \times 512 \times (21+4))} (38 \times 38 \times 4 \times (21+4)).$$

For detection, SSD uses multiple layers (*multi-scale feature maps*) to detect objects independently. The CNN gradually reduces the spatial dimension and thus the resolution of the feature maps also decreases. SSD uses lower resolution layers to detect larger scale objects: for example, the 4×4 feature maps are used for larger scale objects (Fig. 20, *a*). Also, SSD adds 6 more auxiliary convolution layers after the VGG16, with 5 of them added for object detection. In three of those layers, one makes 6 predictions instead of 4 and in total, SSD makes 8732 predictions using 6 layers (Fig. 20, *b*). Thanks to usage of Multi-scale feature maps the accuracy is significantly improved.

Fig. 19. Multibox predictor — *a*; convolutional predictors for object detection (source: article — *b* [47])

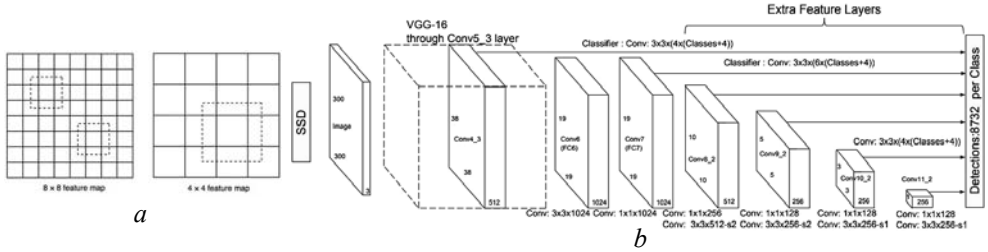


Fig. 20. Lower resolution feature maps (right) detects larger-scale objects — *a* [47]; SSD full architecture — *b* [27; 47]

SSD also uses the idea of *default boundary boxes* that are equivalent to *anchors* in Faster R-CNN [26]. For prediction of boundary boxes one can start with random predictions and use gradient descent for model optimization. During the initial training, however, there can be a problem of determining what shapes (cars or pedestrians) may be optimized for which predictions and research showed that early training can be very unstable. The boundary box predictions on Fig. 21, *a* work well for one category but not for others and it is necessary for initial predictions to be diverse and not looking similar. So for detection of number of objects the predictions have to cover more shapes. Such approach makes training easier and more stable (Fig. 21, *b*).

In real-life, boundary boxes do not have arbitrary sizes and shapes, so the ground truth boundary boxes can be partitioned into clusters with each cluster being represented by a default boundary box, i.e., by the centroid of the cluster. In this way instead of making random predictions one can start the guesses based on those default boxes. The SSD detector also keeps the default boxes to a minimum (4 or 6) with one prediction per default box. As for boundary box localization, its predictions do not use global coordinates; instead, they are relative to the default boundary boxes at each cell (Δcx, Δcy, Δw, Δh), i.e. the offsets (difference) to the default box at each cell for its center (cx, cy), width and height. Each feature map layer shares the same set of default boxes centered at the corresponding cell, but different layers use different sets of default boxes to adjust object detections at different resolutions (Fig. 21, *c*).

SSD's default boundary boxes are chosen manually and network defines a scale value for each feature map layer. As it was shown on Fig. 20, *b*, starting from the left, Conv4_3 detects objects at the smallest scale of 0.2 or sometimes

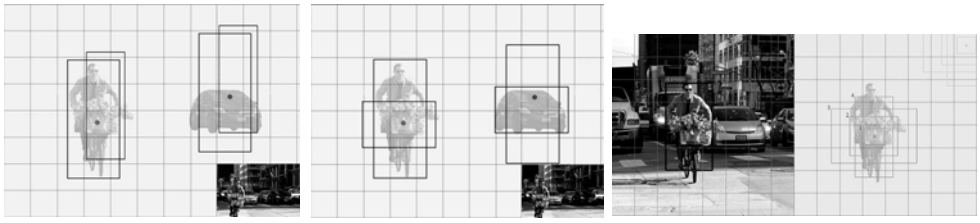
even 0.1. Then, it linearly increases to the rightmost layer at a scale of 0.9. After that one calculates width and height of the default boxes by combining the scale value with the target aspect ratios. As layers make 6 predictions, SSD starts with 5 target aspect ratios of 1, 2, 3, 1/2, and 1/3. Then the width and the height of the default boxes are calculated with $\text{aspect ratio} = 1$. YOLO [52] uses *k*-means clustering on the training dataset to determine those default boundary boxes.

$$w = scale \cdot \sqrt{\text{aspect ratio}},$$

$$h = \frac{scale}{\sqrt{\text{aspect ratio}}};$$

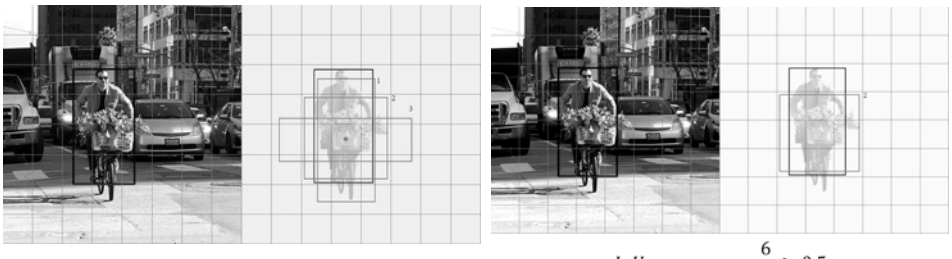
SSD adds an additional default box with scale:

$$scale = \sqrt{\text{scale} \cdot \text{scale at next level}}.$$



a *b* *c*

Fig. 21. With predictions not being diverse, the model will not perform — *a*; diverse predictions cover more object types — *b*; 4 default boundary boxes — c [47]



a *b*

Fig. 22. The ground truth object (blue) and 3 default boundary boxes (green) — *a*; default box 2 has IOU > 0.5 with the ground truth — *b* [47]

With SSD's *matching strategy*, predictions are classified as **positive** matches or negative matches. SSD uses only positive matches for localization cost calculation (the mismatch of the boundary box) and if the corresponding *default boundary box* (and not the predicted boundary box) has an $IOU > 0.5$ with the ground truth, then the match is considered positive; otherwise, it is negative. *IOU (Intersection over Union)* is a metric used to measure the degree of overlap between two bounding boxes and it calculates the ratio of the area of overlap between the two boxes to the area of their union. Mathematically, it is represented as $IOU = S_{\text{intersection}} / S_{\text{union}}$. For example, if there are 3 default boxes and only default box 1 and 2 (not 3) have an $IOU > 0.5$ with the ground truth box above (blue box, Fig. 22, *a*). Then only box 1 and 2 are positive matches and once the positive matches are identified, one calculates the cost using the corresponding

predicted boundary boxes (Fig. 22, *b*). Such matching strategy encourages each prediction to predict shapes closer to the corresponding default box, and in this way the predictions are more diverse and stable in the training [47]. Fig. 23 shows how SSD uses the combination of multi-scale feature maps and default boundary boxes to detect objects at different scales and aspect ratios. The dog matches one default box (in red) in the 4×4 feature map layer, but no other default boxes in the higher resolution 8×8 feature map, while the cat, because of being smaller, is detected only by the 8×8 feature map layer in 2 default boxes (in blue). Higher-resolution feature maps are responsible for detecting small objects and the first layer for object detection, Conv4_3, has a spatial dimension of 38×38 which is large reduction from the input image. That is way SSD usually performs badly for small objects comparing with other detection methods but this problem can be reduced by using images with higher resolution.

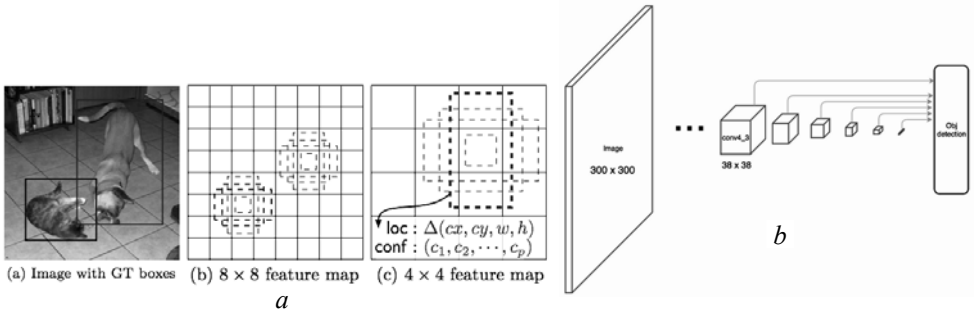


Fig. 23. SSD detector work example — *a* [27]; size reduction — *b* [47]

SSD's *localization loss* is the mismatch between the ground truth box and the predicted boundary box and SSD only penalizes predictions from positive matches. It is necessary for the predictions from the positive matches to get closer to the ground truth; negative matches can be ignored. This loss is defined as the smooth *L1* loss with *cx, cy* as the offset to the default bounding box *d* with width *w* and height *h* [27]:

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{L1}(l_i^m - \hat{g}_j^m);$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w, \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h;$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right), \quad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right);$$

$$x_{ij}^p = \begin{cases} 1, & \text{if IOU} > 0.5 \text{ between default box } i \text{ and} \\ & \text{ground true box } j \text{ on class } p, \\ 0, & \text{otherwise.} \end{cases}.$$

SSD also uses *confidence loss as* the loss of making a class prediction. It penalizes the loss according to the confidence score of the corresponding class for every positive match prediction. For negative match predictions, it penalizes the loss according to the confidence score of the class "0": class "0" means no object

is detected. The loss is the softmax loss over multiple classes confidences *c* (class score), and *N* is the number of matched default boxes:

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^{p} \log(\hat{c}_i^p) - \sum_{i\in Neg} \log(\hat{c}_i^0), \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}.$$

The final loss function is calculated as:

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)),$$

where *N* is the number of positive matches and α is the weight for the localization loss.

For the removal of duplicate predictions pointing to the same object SSD uses non-maximum suppression. It sorts the predictions by the confidence scores and starting from the top confidence prediction, SSD evaluates whether any previously predicted boundary boxes have (Intersection Over Union) $IOU > 0.45$ with the current prediction for the same class and if found, current prediction will be ignored. Despite that, there are still much more predictions made than the number of objects present, sothere are many more negative matches than positive matches. This way the class imbalance problem appears and it worsens the training as the model then is training to learn background space rather than detecting objects. But SSD still requires negative sampling so it can recognize what represents a bad prediction and thus it sorts those negatives by their calculated confidence loss instead of using all of them. It takes the negatives with the top loss and makes sure the ratio between the picked negatives and positives is at most 3:1, which makes the training process faster and more stable [47].

Faster R-CNN architecture. Faster R-CNN is an object detection architecture presented [26; 50; 51] in 2015. It is one of the famous object detection architectures that use convolution neural networks like SSD (Single Shot Detector) or YOLO (You Look Only Once) [52]. The idea behind the development of Faster R-CNN network was to create a unified architecture that not only detects objects within an image but also locates the objects precisely in the image. Faster R-CNN architecture uses the benefits of deep learning, CNNs and RPNs resulting in a combined network that significantly improves the speed and accuracy of the model [50]. It consists of two key components: Region Proposal Network (RPN) and Fast R-CNN detector and as a backbone it utilizes Shared Convolutional Layers, common CNN layers used for both RPN and Fast R-CNN detector (Fig. 24, *a*)

Faster R-CNNs backbone, the CNN is used for extraction of relevant features from the input image and consists of multiple convolutions layers that apply different convolutions kernel to extract those features. The convolutions kernels are designed to capture the hierarchical representations of the input image. This means that starting from the initial layers, CNN captures the low-level features (basic textures or edges) and then with much deeper layers it captures the high level semantic features like objects parts and shapes. Both RPN and Fast R-CNN detector uses the same extracted hierarchical features. This approach helps to significantly reduce the computing time and memory use as the computations performed by these layers are employed for both tasks.

Previously R-CNN and Fast R-CNN architectures use *Selective Search algorithm* [53] region proposals generation. This process is executed on CPU and thus takes more time in computations. With the introduction of Faster R-CNN

[26] this problem was fixed by using a convolutional-based network i.e. RPN. This step reduced proposal time for each image from 2 seconds to 10 ms and improved feature representation by sharing layers with detection stages. Region Proposal Network is a key component of Faster R-CNN architecture, as it is responsible for generating possible ROIs (regions of interest or region proposals) in images that may contain objects. It is based on the concept of attention mechanism in neural networks that tells the subsequent Fast R-CNN detector where in the image the objects should be searched for. The main components of the Region Proposal Network are as follows [50]:

1. *Anchors boxes*: Anchors are used for region proposals generation in the Faster R-CNN model. It uses a set of predefined anchor boxes with different scales and aspect ratios and these anchor boxes are placed at different positions on the feature maps. An anchor box's two key parameters are scale and aspect ratio

2. *Sliding Window approach***:** The RPN runs as a sliding window over the feature map received from the CNN backbone. It uses a small convolutional network, usually a 3×3 convolutional layer, to process the features within the sensitive field of the sliding window and thus this convolutional operation produces scores that represent the object presence probability and regression values for adjusting the anchor boxes.

3. *Objectness Score***:** This value represents the probability that a given anchor box contains an object of interest rather than being just background. RPN predicts this score for each anchor and this objectness score reflects the confidence that the anchor corresponds to a meaningful object region. This score is also used for anchor classification as either positive (object) or negative (background) during training.

4. *IOU (Intersection over Union)* metric.

5. *Non-Maximum Suppression (NMS)***:** This operation is used to remove redundant and select the most accurate proposals, based on the mentioned objectness scores of overlapping proposals and it keeps only proposals with the highest score while eliminating the others.

The RPN uses feature maps the were produced by CNN backbone and applies on them a sliding window approach (as it was shown in Fig. 9, *c*) with anchor boxes of varying scales and shapes to marks potential object positions. This way these anchor boxes are enhanced in the process of training in order to match better actual object positions and sizes. For each anchor, the RPN predicts two parameters: *objectness score*, i.e. probability that anchor contains object, and adjustments for the anchor coordinates to match the actual object's shape. As a large number of region proposals are generated and many of them overlap and correspond to the same object, Non-Maximum Suppression is used to rank the anchor boxes according to their objectness probabilities and take only the top-*N* anchor boxes with the highest scores. Thus it can be guaranteed that the final selected proposals are both accurate and non-overlapping and algorithm considers these selected anchor boxes as as possible region proposals.

The next key component of Faster R-CNN network is the Fast R-CNN detector itself as it is responsible for object detection within the region proposals suggested by the RPN. It operates in several stages [50]:

1. *Region of Interest (RoI) Pooling:* at this first step ROI pooling is applied to the region proposals suggested by the RPN. This operation transforms RPN's variable-sized region proposals into fixed-size feature maps that will be handed

into the network's subsequent layers. RoI pooling divides each region proposal into a grid of cells of equal size and then applies max pooling within each cell, thus, generating a fixed-size feature map for each region proposal (Fig. 24, *b*).

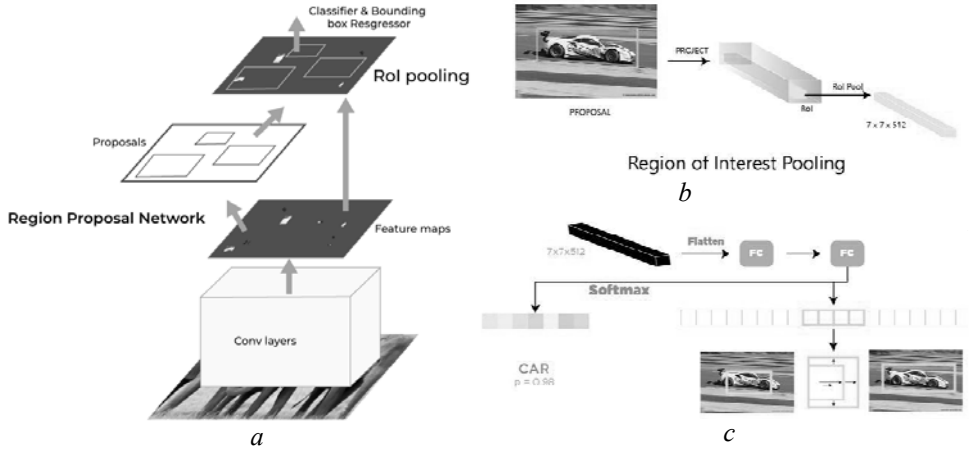


Fig. 24. Faster R-CNN architecture idea — *a*; ROI Pooling —*b*; bounding box regression — c [26; 50]

2. *Feature Extraction:* in this stage feature maps, obtained after ROI Pooling, are fed into the CNN backbone (the same one used in the RPN for feature extraction) in order to extract meaningful features that capture object-specific information and thus one draws hierarchical features from region proposals. Hierarchical features keep the spatial information while separating low-level details and thus allow the network to understand the content of proposed regions.

3. Fully Connected Layers: the algorithm passes the RoI-pooled and feature-extracted regions through a series of fully connected layers as they are used for object classification and bounding box regression.

a. As for *Object Classification*, network predicts class probabilities for each region proposal and points out the possibility that the proposal contains an object of a specific class, and then, the classification is performed by combining the features pulled out from the region proposal with the shared weights of the CNN backbone.

b. Bounding Box Regression: Together with class probabilities, the network predicts bounding box adjustments for each region proposal, which refine the position and size of the region proposal's bounding box and align it more accurately with the actual object boundaries. The first layer is a softmax layer of *N+1* output parameters that predicts the objects in the region proposal, with *N* being the number of class labels and background. The second layer is a bounding box regression layer with $4 \cdot N$ output parameters and is used for bounding box regression of the object in the image.

Fast R-CNN detector uses the *Multi-task Loss Function* as the loss function [26]. It combines classification and regression losses, with classification loss calculating the difference between predicted and true class probabilities and the regression loss calculating the difference between predicted and actual bounding box adjustments:

$$L(p_i,t_i,v_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i,p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i,v_i)$$

where, N_{cls} is the number of ROIs used for classification, N_{reg} is the number of ROIs used for bounding box regression; p_i is the predicted probability of classifying the *i-th* ROI; p_i^* is binary (1 or 0) ground-truth indicator for the *i-th* ROI being a foreground or background object; t_i represents the ground-truth bounding box parameters for the *i-th* ROI; v_i represents the predicted bounding box adjustments for the *i-th* ROI; L_{cls} is the classification loss function, usually computed using cross-entropy loss; L_{reg} is the regression loss function, usually computed using smooth *L1* loss and λ is a balancing parameter for controling the trade-off between the two components of the loss.

After predicting class probabilities and bounding box changes, the final detection results are refined using a *post-processing procedure*. In this step, non-maximum suppression (NMS) is used to reduce redundant detections while keeping the most confident and non-overlapping ones.

Several additional notes on Faster R-CNN Process can be made [50]. For Fast R-CNN object detection network two fully convolution models are commonly used: Zeiler and Fergus model (ZF) [54] with 5 shareable convolutional layers or Simonyan and Zisserman model (VGG-16) [48] with 13 shareable convolutional layers. With the Sliding Window Approach, RPN operates on an $n \times n$ spatial window of the input convolutional feature map with each sliding window being mapped to a lower-dimensional feature. Its dimensions are 256-d for Zeiler and Fergus model (ZF) and 512-d for Simonyan and Zisserman model (VGG-16). Further it is followed by a Rectified Linear Unit (ReLU) activation. The sliding window architecture is effectively realized using an $n \times n$ convolutional layer, followed by two 1×1 convolutional layers for box regression and box classification; if for example, $n = 3$ for the sliding window, it leads to a large effective receptive field on the input image: 171 pixels for ZF and 228 pixels for VGG). For each window position, K region proposals are generated with proposal being defined by an *anchor box*, which is set by scale and aspect ratio. Multiple *anchor boxes* are created by varying these parameters and it results in different scales and aspect ratios. Thus a set of anchor boxes is created, usually $K = 9$, allowing the model to consider various object sizes and shapes. These anchor variations allow the model to handle scale invariance and share features between the RPN and Fast R-CNN.

For each generated region proposal, a feature vector is extracted with a length of 256 (for ZF net) or 512 (for VGG-16 net) and is then processed by two sibling fully-connected (FC) layers: the lower-dimensional feature extracted from the sliding window is fed into two sibling fully-connected layers. The *Box-Classification FC Layer (cls)* predicts an objectness score for the proposed region, it is a binary classifier that assigns an objectness score to each region proposal and determines whether the proposal contains an object or is part of the background. This layer also produces two outputs: one for classifying the region as background and another for classifying the region as an object. The objectness score assigned to each anchor helps to generate the classification label. The *Box-Regression FC Layer (reg)* predicts adjustments for the bounding box of the proposed region and returns a 4-D vector that defines the bounding box of the region proposal.

The Region Proposal Network (RPN) is trained end-to-end using backpropagation and stochastic gradient descent (SGD), i.e. entire network, including the newly added RPN layers and the shared convolutional layers, is optimized together to minimize the loss function. The training strategy is an "image-centric" sampling [50], in which each mini-batch is derived from a single image. This image contains both positive (with object) and negative (background) example anchors and instead of optimizing the loss function for all anchors, the network randomly picks 256 anchors from the image to calculate the loss for the mini-batch. The sampled positive and negative anchors are balanced at a ratio of up to 1:1. Also in order to overcome the potential bias towards negative samples, the training makes sure that each mini-batch contains a *balanced mix* of positive and negative examples. Thus if an image has less than 128 positive samples, additional negative samples are added to the mini-batch to keep the correct ratio.

Regarding the layer initialization, new layers added to the architecture are initialized by getting weights from a Gaussian distribution with a mean of zero and a standard deviation of 0.01. This random initialization is applied to the layers specific to the RPN. The existing shared convolutional layers are initialized using weights pretrained on the ImageNet classification task according to standard practice.

YOLO detector. You Only Look Once (YOLO) [52] is an architecture, that uses end-to-end neural network and allows to makes predictions of bounding boxes and class probabilities all at once [55]. The main difference from other object detection algorithms is that they repurposed classifiers to make detections. YOLO model is based on fundamentally different object detection approach and thus it performs extremely fast, significantly outperforming other real-time object detection algorithms. This detector does all of its predictions with the help of a single fully connected layer in contrast to other networks, like Faster RCNN, which detect possible regions of interest with help of RPN and then do recognition on those regions separately. In other words, for the same image YOLO does a single iteration when RPN-baset networks perform multiple iterations.

The YOLO architecture [52; 55] is depicted on Fig. 25: the algorithm receives an image as input, then detects objects on this image with help of a simple deep convolutional neural network as its backbone. In this model the first 20 convolution layers are pre-trained using ImageNet by plugging in a temporary average pooling and fully connected layer and then, this pre-trained model is converted to perform detection, because thanks to earlier studies it was carried out that adding convolution and connected layers to a pre-trained network helps to improve performance. The model's final fully connected layer makes predictions on both class probabilities and bounding box coordinates.

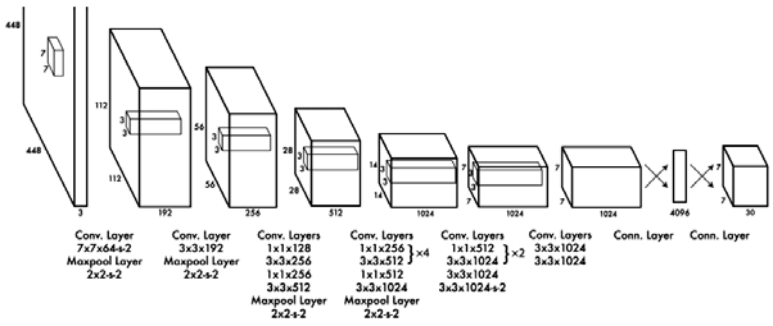


Fig. 25. YOLO original architecture (source: original paper [52; 55])

During processing, YOLO's algorithm divides an input image into an $S \times S$ grid. Within this grid it checks, if the center of an object gets placed within some grid cell, then that grid cell is responsible for detecting that object. Thus, each grid cell predicts B bounding boxes and confidence scores for those boxes and these confidence scores indicate how confident the model is that some box contains an object and how accurate the model thinks that predicted box is. The algorithm predicts multiple bounding boxes per grid cell. As at training stage it is necessary that for each object only one bounding box predictor should be responsible for this object, YOLO assigns one predictor to be "responsible" making predictions of an object based on which prediction has the highest current IOU with the ground truth. This task leads to bounding box predictors being specialized for different tasks: each predictor gets better at forecasting certain sizes, aspect ratios, or classes of objects, improving the overall recall score. YOLO models also use one key approach called *Non-Maximum Suppression (NMS)*, a post-processing step used for improvement of object detection accuracy and efficiency. In the process of object detection it is a common situation when for a single object in an image multiple bounding boxes are generated. As such bounding boxes can be located at different positions or may overlap while still representing the same object, NMS is used to identify and remove redundant or incorrect bounding boxes and to output a single correct bounding box for each object in the image [55].

Since the initial release of YOLO in 2015, it had received several modifications and improvements and thus new versions have appeared. For our research the YOLO v5 version was chosen as it is still one of the most recent modifications and is easy to install for instant usage. The v5 version [56] was introduced in 2020 by developers of original YOLO. Unlike the original model, v5 version uses EfficientDet-based architecture as a backbone. It allowed the new model to increase accuracy and generalization to a wider range of object categories. The v5 version also uses a new method for generating the anchor boxes, called "dynamic anchor boxes", which involves using a clustering algorithm to group the ground truth bounding boxes into clusters and then it uses the centroids of the clusters as the anchor boxes. Thanks to this the anchor boxes can be more closely aligned with the size and shape of detected objects. One more new idea that was introduced in YOLO v5 is the concept of the *Spatial Pyramid Pooling (SPP)*. It is a type of pooling layer used to reduce the spatial resolution of the feature maps. It allows the model to see the objects at multiple scales and thus improves the detection performance on small objects. In addition, v5 model has introduced a new variant of the IOU loss function called "CIOU Loss" and designed to improve the model's performance on imbalanced datasets [55].

Training and experiment. The whole experiment was performed locally on a laptop with an Intel i9-13980HX CPU, NVidia 4090 16GB laptop GPU and 64GB RAM; CUDA 11.2. Due to hardware limitations the training was decided to be limited by 5000 train steps for general comparison. The test runs for each model were performed on the same machine. The dataset used for training consisted of 562 images (selected video frames). The test set consisted of 63 images. The part of test is not only to evaluate the trained model accuracy on train dataset, but to see how fast the models train considering their different internal architecture. As mentioned previously, we did not measure FPS because each video frame

is additionally preprocessed to improve its color and remove noise, and thus this processing takes some additional time.

Also because of hardware limitations (GPU memory size) each model was trained with maximum possible image batch size. For example, we did not consider the EfficientDet D3 or higher versions since on current hardware it was possible only to train with a batch of 4 which was considered not enough for good model training. Also as the CenterNet MobileNetV2 FPN 512×512 training results were very poor despite 5.5h of training it was decided to exclude it from the comparison.

Note on detection speed benchmark: for the first image, the detection time was always much longer comparing to other images in test set due to the model being loaded by program for the first time. Thus we excluded the detection time for the first image from average detection speed calculation as this delay occurs for the very first image only and is insignificant for the further detection period. Fig. 26 shows some examples of work of the neural network object detector.

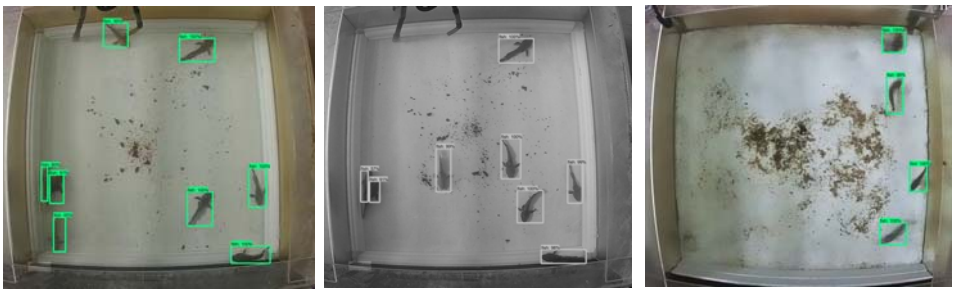


Fig. 26. NN object detection results

CONCLUSIONS

The experiment results are shown on Table 5. According to them, it was carried out, that such models as the EfficientDet, SSD net with ResNet50 V1, ResNet101 V1 backbone and the Faster R-CNNs are not very suitable for training on usual hardware as they operate with large amounts of data during training and thus require a lot of memory when trying larger batch sizes which can become problematic. Using smaller batch size can cause model not to train good enough even despite that its training time was sometimes less compared to other models. Extending the training time also cannot be good option as seen EfficientDet 10K step training. As for other mentioned models (SSDs and Faster R-CNNs), with the same training amount of 5K steps these models also showed smaller accuracy.

The detection speed benchmark results also show that for example, only CenterNet Resnet101 V1 FPN 512x512, SSD MobileNet V1 FPN and YOLO v5 manage to fit within the given detection time threshold of about 110 ms. Other SSD nets managed to show longer detection time with a bit smaller accuracy then the SSD MobileNet V1 FPN.

Thus, for our further research it is decided to consider the SSD MobileNet V1 FPN and YOLO as the most suitable for detection of relatively simple objects with minimum visual details. This result might be useful for anyone trying to use any of studied models for similar tasks. From all tested model only these ones can be trained with minimum significant amount of data and perform accurate and fast enough even at less capable hardware then used for our experiment. Thus we

will consider them as primary neural object detectors for fish detection and tracking. But we still might consider CenterNet with HourGlass104 and Resnet101 V1 FPN for some other tasks as they also maintain relatively good ratio of detection speed and especially training possibilities and resulting accuracy.

Table 5. Model training and evaluation results

Architecture	Backbone	Network input size	Training batch size	Training time (approx.)	AP (%)	AP50 IOU (%)	AP75 IOU (%)	AP(M) (%)	AP(L) (%)	Avg detection speed (ms)
CenterNet	Hour-Glass104	512x512	10	5.5 h	79.16	99.23	93.64	62.02	80.9	183
CenterNet	Resnet101 V1 FPN	512x512	20	5 h	74.59	98.19	91.05	58.01	76.31	83
Efficient-Det D2	Efficient-Net	896x896	6	3 h	55.83	92.31	62.23	32.65	58.32	233
Efficient-Det D2	Efficient-Net	896x896	6 (10K steps)	3 h	65.47	93.9	82.41	45.31	67.44	263
SSD	MobileNet V1 FPN	640x640	32	10 h	87.42	99.03	98.96	75.58	88.8	112
SSD	MobileNet V2 FPNLite	640x640	28	4.5 h	73.43	97.5	89.95	55.31	75.22	116
SSD	ResNet50 V1 FPN	1024x1024	8	7.5 h	73.53	96.6	88.62	47.13	76.17	141
SSD	ResNet101 V1 FPN	1024x1024	5	3.5 h	23.29	44.47	24.45	3.04	25.46	175
Faster R-CNN	ResNet50 V1	800x1333	6	2 h	71.41	98.5	88.89	48.0	73.67	143
Faster R-CNN	ResNet101 V1	1024x1024	5	3 h	70.85	97.65	88.72	51.48	72.85	241
					AP50-95			Precision	Recall	
Yolo V5	CSPDark-net53	640 x 640	80	1.5 h	98.5	98.9	-	98.1	97.7	8

Acknowledgements. Special thanks for the research assistance and provided test videos and images of lab animals to Faculty of Biology of Odesa I.I. Mechnikov National University.

REFERENCES

1. A.R. Smith, “Color Gamut Transform Pairs,” in *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pp. 12–19,1978. doi: 10.1145/800248.807361
2. M.A. Shvandt, V.V. Moroz, “Overview Of The Detection And Tracking Methods Of The Lab Animals”, *System Research & Information Technologies*, no. 1, 2022, pp. 124–148. doi: 10.20535/SRIT.2308-8893.2022.1.10
3. V.V. Moroz, M.A. Shvandt, “Study of movement and behavior of laboratory animals by methods of object detection and tracking”, *Herald of the National Technical University ‘KhPI’, Series of ‘Informatics and Modeling’, Kharkiv: NTU ‘KhPI’, Kharkiv*, vol. 13, no. 1338, pp. 93–103, 2019. doi: 10.20998/2411-0558.2019.13.09
4. TensorFlow 2 Detection Model Zoo. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md
5. T.-Y. Lin et al., Microsoft COCO: *Common Objects in Context*. 2014, 15 p. doi: 10.48550/ARXIV.1405.0312. Available: https://arxiv.org/abs/1405.0312
6. L. Wood, F. Chollet, *Efficient Graph-Friendly COCO Metric Computation for Train-Time Model Evaluation*. 2022, 7 p. doi: 10.48550/ARXIV.2207.12120. Available: https://arxiv.org/abs/2207.12120

7. K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, Q. Tian, *CenterNet: Keypoint Triplets for Object Detection*. 2019, 10 p. doi: 10.48550/ARXIV.1904.08189. Available: https://arxiv.org/abs/1904.08189
8. H. Law, J. Deng, *CornerNet: Detecting Objects as Paired Keypoints*. 2018, 14 p. doi: 10.48550/ARXIV.1808.01244. Available: https://arxiv.org/abs/1808.01244
9. X. Lu, B. Li, Y. Yue, Q. Li, J. Yan, *Grid R-CNN. 2018*, 9 p. doi: 10.48550/ARXIV.1811.12030. Available: https://arxiv.org/abs/1811.12030
10. X. Zhou, D. Wang, P. Krähenbühl, *Objects as Points*. 2019, 12. doi: 10.48550/ARXIV.1904.07850. Available: https://arxiv.org/abs/1904.07850
11. S. Trivedi, *CenterNet: Objects as Points - A Comprehensive Guide*. 2020. Available: https://medium.com/visionwizard/centernet-objects-as-points-a-comprehensive-guide-2ed9993c48bc
12. L. Huang, Y. Yang, Y. Deng, Y. Yu, *DenseBox: Unifying Landmark Localization with End to End Object Detection*. 2015, 13 p. doi: 10.48550/ARXIV.1509.04874. Available: https://arxiv.org/abs/1509.04874
13. Z. Tian, C. Shen, H. Chen, T. He, *FCOS: Fully Convolutional One-Stage Object Detection*. 2019, 13 p. doi: 10.48550/ARXIV.1904.01355. Available: https://arxiv.org/abs/1904.01355
14. A. Newell, K. Yang, J. Deng, *Stacked Hourglass Networks for Human Pose Estimation*. 2016, 17 p. doi: 10.48550/ARXIV.1603.06937. Available: https://arxiv.org/abs/1603.06937
15. K. He, X. Zhang, S. Ren, J. *Sun, Deep Residual Learning for Image Recognition*. 2015, 12 p. doi: 10.48550/ARXIV.1512.03385. Available: https://arxiv.org/abs/1512.03385
16. A.G. Howard et al., *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017, 9 p. doi: 10.48550/ARXIV.1704.04861. Available: https://arxiv.org/abs/1704.04861
17. D. Wang, E. Shelhamer, T. Darrell, *Deep Layer Aggregation*. 2017, 10 p. doi: 10.48550/ARXIV.1707.06484. Available: https://arxiv.org/abs/1707.06484
18. T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. *Dollár, Focal Loss for Dense Object Detection*. 2017, 10 p. doi: 10.48550/ARXIV.1708.02002. Available: https:// arxiv.org/abs/1708.02002
19. S. Trivedi. *Understanding Focal Loss-A Quick Read*. 2020. Available: https:// medium.com/visionwizard/understanding-focal-loss-a-quick-read-b914422913e7
20. S. Bangar, Resnet Architecture Explained. 2022. Available: https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d
21. P. Ruiz, *Understanding and visualizing ResNets*. 2018. Available: https://towards-datascience.com/understanding-and-visualizing-resnets-442284831be8
22. T.-Yi Lin et al., *Feature Pyramid Networks for Object Detection*. 2016, 10 p. doi: 10.48550/ARXIV.1612.03144. Available: https://arxiv.org/abs/1612.03144
23. J. Hui, *Understanding Feature Pyramid Networks for object detection (FPN)*. 2018. Available: https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c
24. S.-H. Tsang, *Review: FPN - Feature Pyramid Network (Object Detection)*. 2019. Available: https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610
25. S. Tanwar, FPN (*feature pyramid networks*). 2020. Available: https://medium.com/analytics-vidhya/fpn-feature-pyramid-networks-77d8be41817c
26. S. Ren, K. He, R. Girshick, J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015, 14 p. doi: 10.48550/ARXIV.1506.01497. Available: https://arxiv.org/abs/1506.01497
27. W. Liu et al., *SSD: Single Shot MultiBox Detector*. 2015, 17 p. doi: 10.48550/ARXIV.1512.02325. Available: https://arxiv.org/abs/1512.02325
28. S.-H. Tsang, Review: MobileNetV1 - Depthwise Separable Convolution (Light Weight Model). 2018. Available: https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69
29. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2018, 14 p. doi: 10.48550/ARXIV.1801.04381. Available: https://arxiv.org/abs/1801.04381v4

30. S.-H. Tsang, Review: *MobileNetV2 - Light Weight Model* (*Image Classification*). 2019. Available: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c
31. A. Newell, K. Yang, J. Deng, *Stacked Hourglass Networks for Human Pose Estimation*. 2016, 17 p. doi: 10.48550/ARXIV.1603.06937. Available: https://arxiv.org/abs/1603.06937
32. E. Calleris, *The Hourglass Network*. 2022. Available: https://medium.com/@calleris.enrico/hourglass-network-6e74cdb9ce2f
33. S. Li, *Simple Introduction about Hourglass-like Model*. 2017. Available: https://medium.com/@sunnerli/simple-introduction-about-hourglass-like-model-11ee7c30138
34. J. Long, E. Shelhamer and T. Darrell, *Fully Convolutional Networks for Semantic Segmentation*. 2014, 10 p. doi: 10.48550/ARXIV.1411.4038. Available: https://arxiv.org/abs/1411.4038
35. A. Krizhevsky, I. Sutskever, G.E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - NIPS'12, Curran Associates Inc., Red Hook, NY, USA, 2012*, vol 1., pp. 1097–1105.
36. O. Ronneberger, P. Fischer, T. Brox, U-Net: *Convolutional Networks for Biomedical Image Segmentation*. 2015, 8 p. doi: 10.48550/ARXIV.1505.04597. Available: https://arxiv.org/abs/1505.04597
37. S. Minaee et al., *Image Segmentation Using Deep Learning: A Survey*. 2020, 22 p. doi: 10.48550/ARXIV.2001.05566. Available: https://arxiv.org/abs/2001.05566
38. F. Milletari, N. Navab, S.-A. Ahmadi, *V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation*. 2016, 11 p. doi: 10.48550/ARXIV.1606.04797. Available: https://arxiv.org/abs/1606.04797
39. J.T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, *Striving for Simplicity: The All Convolutional Net*. 2015, 14 p. doi: 10.48550/ARXIV.1412.6806. Available: https://arxiv.org/abs/1412.6806
40. D. Oñoro-Rubio, M. Niepert, *Contextual Hourglass Networks for Segmentation and Density Estimation*. 2018, 3 p. doi: 10.48550/ARXIV.1806.04009. Available: https://arxiv.org/abs/1806.04009
41. R. Sharma, *EfficientDet: Scalable and Efficient Object Detection*. 2021. Available: https://medium.com/analytics-vidhya/efficientdet-scalable-and-efficient-object-detection-384a5df9011a
42. M. Tan, R. Pang, Q.V. Le, *EfficientDet: Scalable and Efficient Object Detection*. 2019, 10 p. doi: 10.48550/ARXIV.1911.09070. Available: https://arxiv.org/abs/1911.09070
43. M. Tan, Q.V. Le, *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020, 11 p. doi: 10.48550/ARXIV.1905.11946. Available: https://arxiv.org/abs/1905.11946
44. J. Solawetz, *A Thorough Breakdown of EfficientDet for Object Detection*. 2020. Available: https://towardsdatascience.com/a-thorough-breakdown-of-efficientdet-for-object-detection-dc6a15788b73
45. S. Liu, L. Qi, H. Qin, J. Shi, J. Jia, *Path Aggregation Network for Instance Segmentation*. 2018, 11 p. doi: 10.48550/ARXIV.1803.01534. Available: https://arxiv.org/abs/1803.01534
46. C. Peng et al., *MegDet: A Large Mini-Batch Object Detector*. 2017, 9 p. doi: 10.48550/ARXIV.1711.07240. Available: https://arxiv.org/abs/1711.07240
47. J. Hui, *SSD object detection: Single Shot MultiBox Detector for real-time processing*. 2018. Available: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06
48. K. Simonyan, A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014, 14 p. doi: 10.48550/ARXIV.1409.1556. Available: https://arxiv.org/abs/1409.1556
49. J. Boschman, *VGG16* (*2014*) *– one minute summary*. 2021. Available: https://medium.com/one-minute-machine-learning/very-deep-convolutional-networks-for-large-scale-image-recognition-2014-one-minute-summary-44a8f04586ab
50. Faster *R-CNN – ML*. Available: https://www.geeksforgeeks.org/faster-r-cnn-ml/

51. A. Khazri, *Faster RCNN Object detection*. 2019. Available: https://towardsdatascience.com/faster-rcnn-object-detection-f865e5ed7fc4
52. J. Redmon, S. Divvala, R. Girshick, A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*. 2016, 10 p. doi: 10.48550/ARXIV.1506.02640. Available: https://arxiv.org/abs/1506.02640
53. J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, A.W.M. Smeulders, "*Selective Search for Object Recognition", in International Journal of Computer Vision*, 2013, 14 p. doi: 10.1007/s11263-013-0620-5
54. M. D. Zeiler and R. Fergus, *Visualizing and Understanding Convolutional Networks*. 2013, 11 p. doi: 10.48550/ARXIV.1311.2901. Available: https://arxiv.org/abs/1311.2901
55. R. Kundu, YOLO: *Algorithm for Object Detection Explained [+Examples]*. 2023. Available: https://www.v7labs.com/blog/yolo-object-detection
56. S.-H. Tsang, Brief Review: *YOLOv5 for Object Detection*. 2023. Available: https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a

Received 25.12.2023

INFORMATION ON THE ARTICLE

Maksym A. Shvandt, ORCID: 0000-0002-4580-3961, Odesa I.I. Mechnikov National University, Ukraine, e mail: maxim.shvandt@gmail.com

Volodymyr V. Moroz, ORCID: 0000-0002-3240-4590, Odesa I.I. Mechnikov National University, Ukraine, e mail: v.moroz@onu.edu.ua

ОГЛЯД МЕТОДІВ І МОДЕЛЕЙ ДЕТЕКТУВАННЯ ОБ'ЄКТІВ НА БАЗІ НЕЙРОННИХ МЕРЕЖ НА ПРИКЛАДІ ЇХ ЗАСТОСУВАННЯ ДЛЯ СПОСТЕРЕЖЕННЯ ЗА ЛАБОРАТОРНИМИ ТВАРИНАМИ / М.А.Швандт, В.В. Мороз

Анотація. Наведено стислий огляд найпоширеніших базових моделей нейронних мереж для виявлення об'єктів. Потреба в автоматизації процесів спостереження та нагляду постійно зростає. Одним із ключових завдань таких процесів є виявлення об'єкта, що цікавить, для подальшого його аналізу. Було запропоновано багато основних алгоритмів і підходів виявлення об'єктів, але більшість із них, зазвичай, мають деякі обмеження щодо області застосування. Здебільшого ці обмеження зумовлені характером спостережуваного середовища або через те, що підходи до виявлення залежать від окремих характеристик об'єкта, як-от лише колір або деякі основні форми. Для вирішення цих проблем був розроблений загалом новий підхід до виявлення об'єктів із використанням нейронних мереж. Подано основи та основні аспекти найбільш поширених моделей нейронних мереж для виявлення об'єктів. Експеримент продемонстрував особливості, переваги та недоліки досліджуваних методів при застосуванні для виявлення лабораторних тварин під час вивчення їх поведінки. З огляду на це зроблено висновки та надано рекомендації щодо їх використання.

Ключові слова: детектування об'єктів, нейронна мережа, нейронний шар, архітектура, модель, оптимізація, оцінка, прогноз, відео, зображення, кадр, задній фон, передній фон, експеримент, порівняння.