# Introduction to Convolutional Neural Networks.

**Image classification** is a common topic when learning about deep learning and computer vision. It consists of using one image as input to a neural network and receiving a prediction of its class. Is the image a dog or a cat? A type of neural network called **Convolutional Neural Network** (**CNN**), is usually used to solve those kinds of problems [Figure 1].

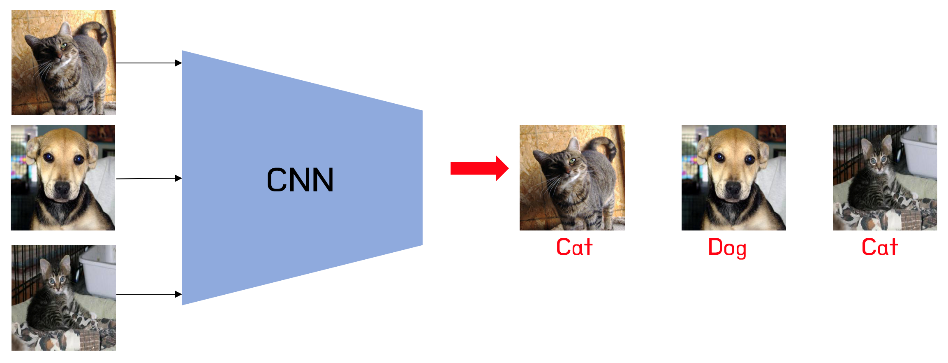


Figure 1: CNNs can be trained to predict classes from given images

A **CNN** is a neural network that includes **convolutional layers**, which consist of a list of a set number of kernels with a given shape (n x n) that slide through the image with a step of a given number of pixels called “**stride**”, computing for each step the dot product of the n x n values of the kernel with the corresponding pixel values of the image in that step. Each kernel is passed through all the images of the input, which can be three for the first layer if the input is an RGB image, having three color channels, or could be the number of kernels in the previous convolutional layer.

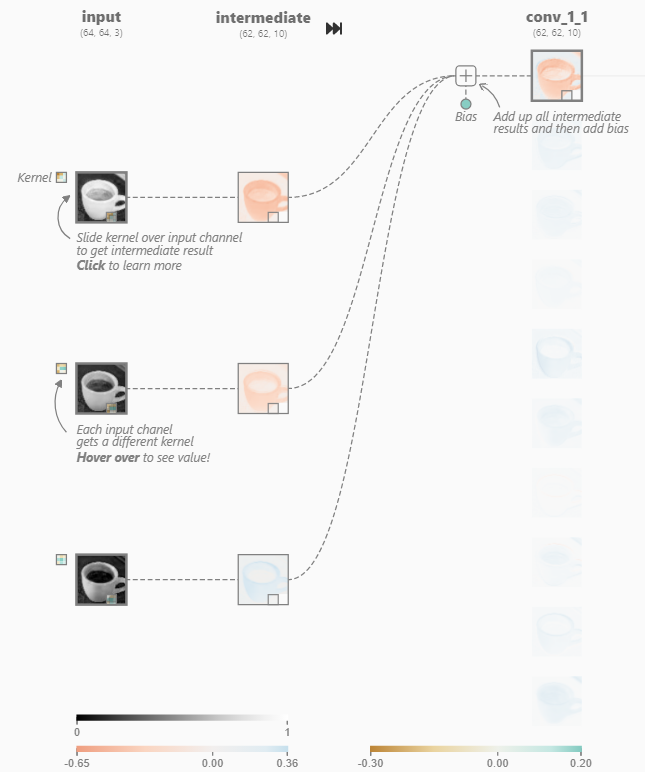
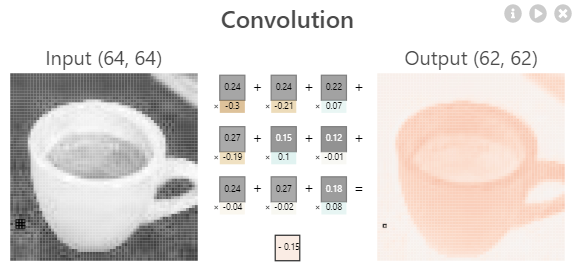


Figure 2: The input layer and first convolutional layer from a CNN (left) and a close view of one of its kernels (right). Source: [CNN explainer website](https://poloclub.github.io/cnn-explainer/)



The values of each of the n x n values in each kernel and the biases are the **trainable weights** of the neural network. Weights, which are initially initialized randomly, are “trained” by optimizing (or minimizing) the final **loss function** of the network, which computes the “distance” between the predicted output and the truth (ground truth). There are different loss functions aimed at different kind of problems. If the network must predict a class, a **Categorical Crossentropy** loss is commonly used.

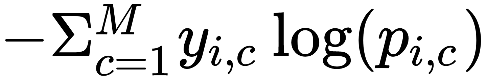


Figure 3: Categorical Cross entropy function. y stands for the true label of the class, being 1 if the example i is that class and 0 if not. pi stands for the output probability of the example i being that class according to the network. This formula increases if the true class has a low prediction probability, as it results in the negated log of that probability, which increases as the probability is closer to 0.

There are different **optimization algorithms**, such as **Stochastic Gradient Descent (SGD)** or **Adam**, and all of them use partial derivatives of the loss function in respect to each weight by applying the chain rule from the loss function back to that weight, by multiplying the partial derivatives of all the functions in the way. That is why the process of optimizing the network is called “**backpropagation”.** After computing the partial derivatives, each weight is updated by multiplying a set parameter known as **Learning Rate,** usually lower than 1, to the result of its partial derivatives (depends on the optimization algorithm) and subtracting that number from the original weight.

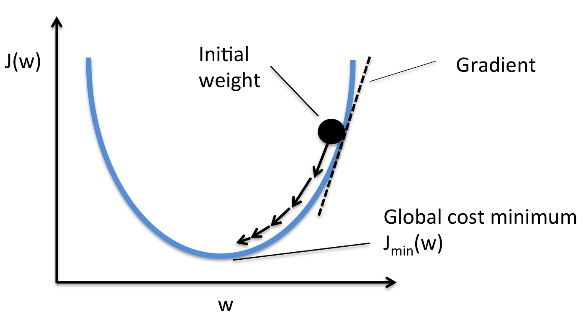


Figure 4: Example of a simple case of gradient descent with one weight. The partial derivative (Gradient) of the final function J(w) with respect to w is positive if J(w) increases as w does, or negative if it decreases when w increases. To minimize J(w), the derivative multiplied by a learning rate is subtracted from w, ensuring that the next value of J(w) will be closer to the minimum, as a positive derivative indicates that w should decrease and a negative one indicates it should increase.

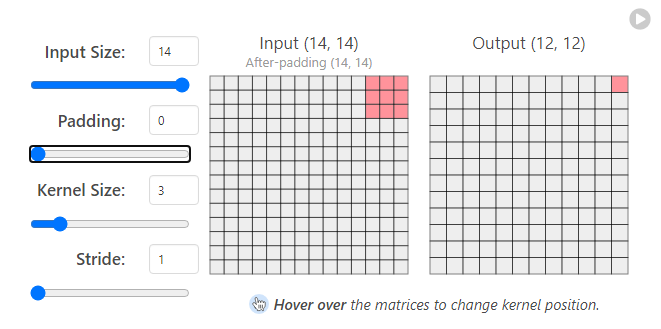
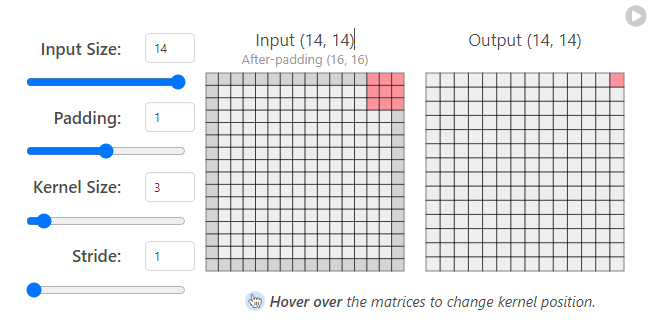
The final output of each kernel is the sum of the results for all the inputs plus a bias, which then is passed through a non-linear activation function, such as **ReLU** = max (input, 0). The output shape can be different from the input depending on the kernel size and stride. For example, a 3x3 kernel going through a 14x14 image outputs a 12x12 image, as the right side of the kernel reaches the end of the image in 12 steps, and the bottom of the kernel reaches the bottom of the image in 12 steps as well. Reshaping can be avoided with the use of **padding**, like adding 0 to each border of the image, thus becoming a 16x16 image. Output sizes can also be higher than the original depending on the stride and padding.

Figure 5: 3x3 kernels can generate an output of the same size as the original image if a padding of 1 pixel is used.

CNNs include other layers as well, typically the **Pooling layers**, whose main function is to reduce the output size of the previous layer. For example, **MaxPooling** layers use a n x n kernel that slides through the input image, like a convolutional kernel, and the output is simply the maximum value of the pixel in that region for each step.

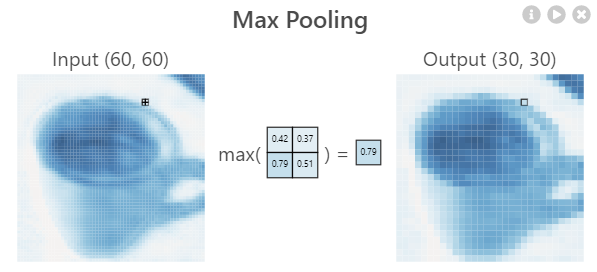


Figure 6

After all the convolutional and pooling layers, the output is flattened to have a 1D array including all the pixel values of the previous output, for example, if the last layer is a Convolutional layer of ten kernels, each generating a 5x5 output, the flattened output would be an array of 10x5x5 = 250 values. CNNs can also have fully connected layers at the top (end) after the flatten layer. The last layer is usually a fully connected layer with the number of neurons equal to the number of classes to predict, giving the final prediction as output, which can be used as input to a **Softmax** activation function that returns the probabilities of the image pertaining to each one of the classes.

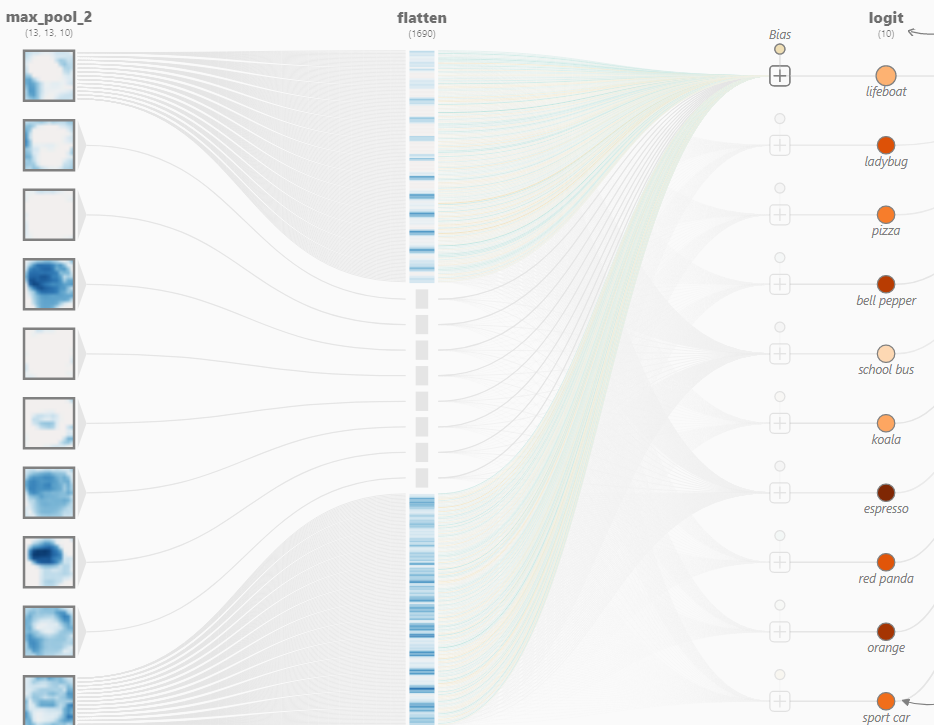


Figure 7

# **Object Detection models**

CNNs solve the problem of image classification, but what happens if there are different objects in an image that we want to identify? If both a cat and a dog appear in an image, the CNN will only output one predicted class. This problem can be solved with **Object Detection**, which consists of finding the area of the image where the object is, and its class. The output of object detections models is a list of each class detected with its bounding box, which can be in the form of the coordinate x and y of the center of the box and its width and height.

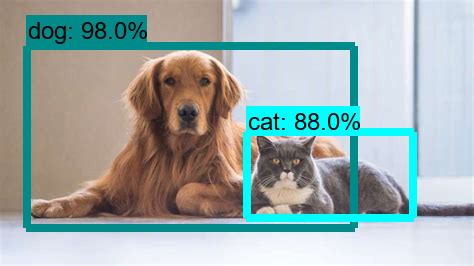


Figure 8: An example of object detection.

## Two-Stage Object Detection

### ****Region-Based CNN (R-CNN)****

Early models of object detection used fixed size **sliding windows** that moved through all the image, sending each cropped image to a CNN at each iteration to classify that part of the image. If a class was found, the model considered there was an object in that place. This method has the problem of being too slow, as lots of images were passed to the CNN.

**Region-Based CNN (R-CNN)** solved this problem by reducing the number of cropped images to feed from the initial image to the CNN by using **Selective Search** before the CNN. Selective search is an algorithm that tries to segment all the objects of an image at different scales and containing components of different color and texture.



Figure 9: With simple image segmentation, the image is segmented by color and texture, so the car could have its wheels in a different group than its chassis, or it would be difficult to detect the whole span of the table containing dishes.

After an initial segmentation and saving the bounding boxes spanning each group, selective search uses a greedy algorithm to combine each group to its closest neighbor, computing that “distance” in terms of color, texture, size, and shape. The bounding boxes of the new combined groups are saved as well. Selective Search keeps doing iterations of combining groups and saving their bounding boxes until only one group remains.

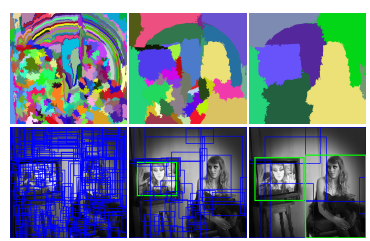


Figure 10: The left most image shows the original segmentation, with its corresponding bounding boxes. Each iteration gets fewer groups and bounding boxes as they combine with each other, being able to have the whole woman from the right in one group and bounding box.

R-CNN takes the bounding boxes from the selective search as input, which would be around 2000, using each image inside them on a CNN, which is called “**backbone**”, to classify them and predict which object contains, thus reducing the number of images to feed to the CNN compared to the sliding window method. The backbone tends to be a popular pretrained image classification model, such as **ResNet** or **EfficientNet**, which have good accuracy scores on big image datasets like **ImageNet**. After the classification of each region is done, the four values describing the bounding box position are used as input for a regressor that adjusts their values using the ground truth to improve the precision of the detection.

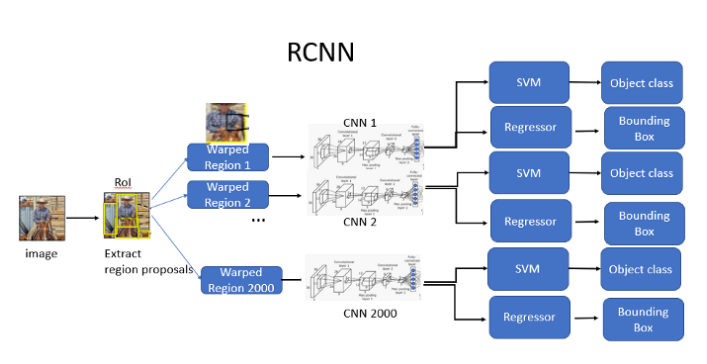


Figure 11: Architecture of a R-CNN

### Intersection Over Union (IoU)

**The Intersection over Union (IoU)** evaluates the similarity between two boxes and is computed by **dividing the area of their intersection by the area of their union**. During training, the IoU between predicted boxes and ground truth boxes is computed. If the prediction and the ground truth are the same, the IoU will be equal to 1. Predicted bounding boxes with IoU > 0.5 are considered true for their detected class and the rest are considered negatives, by assigning them the class “background”. Bounding boxes that have an IoU higher than 0.3 with respect to another bounding box with higher IoU with the same ground truth box are considered overlaps and classified as negative.

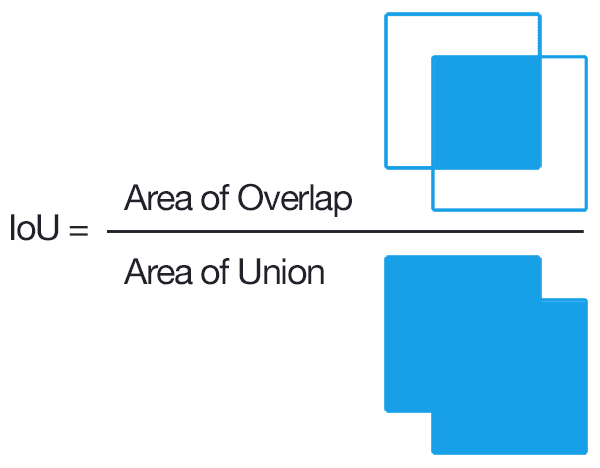


Figure 12

### Non-Max Suppression (NMS)

During inference, a similar method is commonly used in Object Detection models in general to only output the most relevant bounding boxes, called **Non-Max Suppression (NMS)**. This algorithm starts removing bounding boxes with a lower **classification score** than a specified threshold, then, for each class, it sorts the remaining bounding boxes using their probabilities and pick the highest scoring one, which is removed from the list and marked as the “current element”.

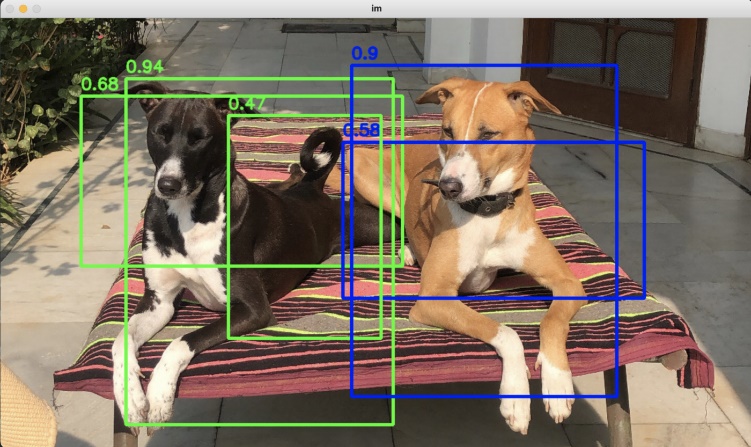
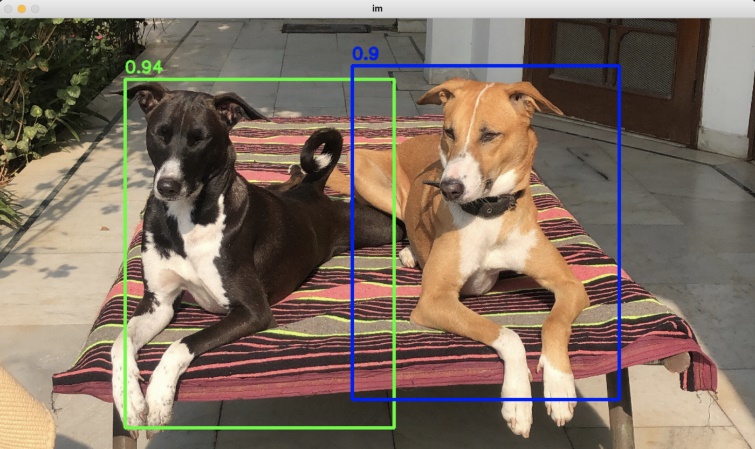
The **IoU** of that element with each of the remaining boxes from the list is computed and if it is above a specified threshold, the box from the list is deleted. Finally, the “current element” is added to the “final list”. The process of picking the highest scoring bounding box and deleting the rest depending on their IoU is repeated until there are no boxes left on the initial list, and the “final list” is returned as output.

Figure 13: Result of using Non-Max Suppression

### Fast R-CNN

R-CNN has the drawback of still being very slow, as feeding the CNN with 2000 regions per image is still very computationally expensive and depends on the output of the selective search algorithm, which is not trainable and can generate bad region proposals from the start.

**Fast R-CNN** tries to solve some of those problems by using the whole image as input for a CNN backbone that **produces a convolutional feature map** which, like the input, is an “image” encoded as a matrix with three dimensions: height, with and color channels. Then, for each region from the selective search, instead of cropping the original image, the region of interest (RoI) is cropped from the convolutional feature map and goes through a **RoI Pooling Layer,** which **turns the RoI into a fixed-length (h x w) feature vector** by dividing the RoI into (h x w) subsections and picking the maximum value out of each. Each feature vector is fed to a set of fully connected layers that branch into two output layers: one that classifies the object and the other that adjusts the 4 values that encode the bounding box.

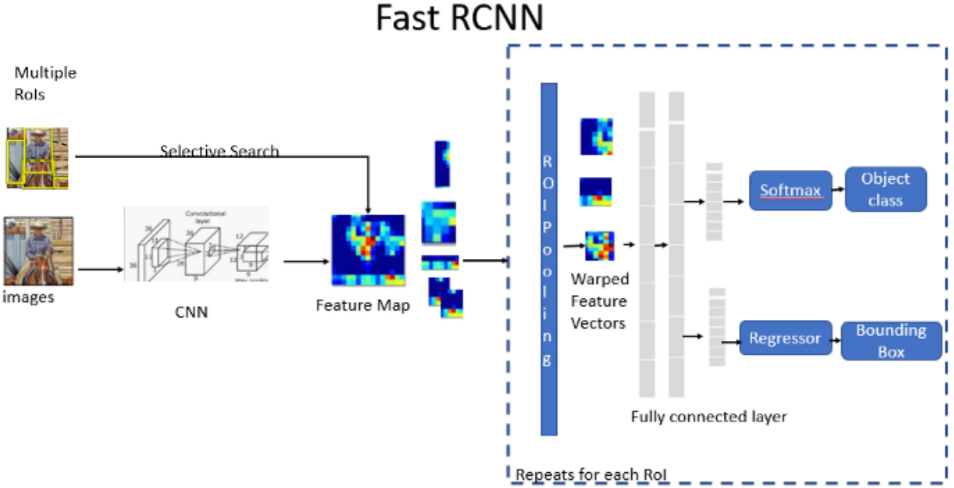


Figure 14

Fast R-CNN does not feed 2000 images to the CNN, as the convolution operation is only done once per image, making it much faster than the original R-CNN at test time. However, it still depends on region proposals from the selective search algorithm, which are computationally expensive and affects its performance.

### Faster R-CNN

**Faster R-CNN** avoids using region proposal methods such as Selective Search by, after generating an initial convolutional feature map like Fast-RCNN, using **Region Proposal Networks (RPN)**, a convolutional network that takes the convolutional feature map (of any size) as input and outputs a set of region proposals. Each region proposal has an **objectness score** measuring how likely that region pertains to a class and not to the background, which is computed by checking the IoU of the region with the ground truth boxes. Those region proposals are used as the input of a **Fast-RCNN** network that outputs the final detections.

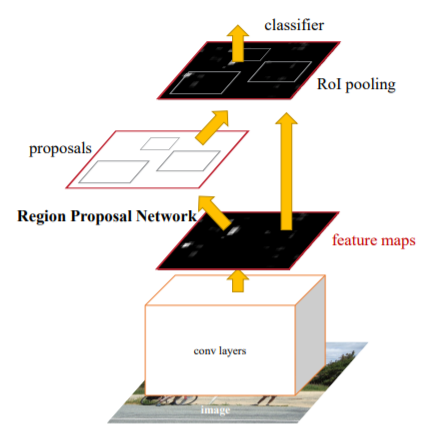


Figure 15: Representation of a Faster R-CNN. Feature maps are used both in the RPN and the Fast R-CNN

The **Region Proposal Network (RPN)** uses the output of the backbone convolutional neural network, which takes the input image after resizing it such that its shortest side is 600 pixels and the longer side not exceeding 1000 pixels. The backbone total stride is 16, meaning that for each consecutive pixel in the backbone output, the corresponding pixels in the original image are 16 pixels apart.

RPN uses by default 3 fixed scales and 3 aspect ratios to create 9 **“anchors”** centered in a 3x3 sliding window that slides through the feature map. These anchors indicate possible objects of various sizes and aspect ratios in the corresponding location at the original image in respect to that point in the feature map. The network then must check if each anchor corresponds to a real object and refine its coordinates to create region proposals that will be fed to the **Fast R-CNN**.

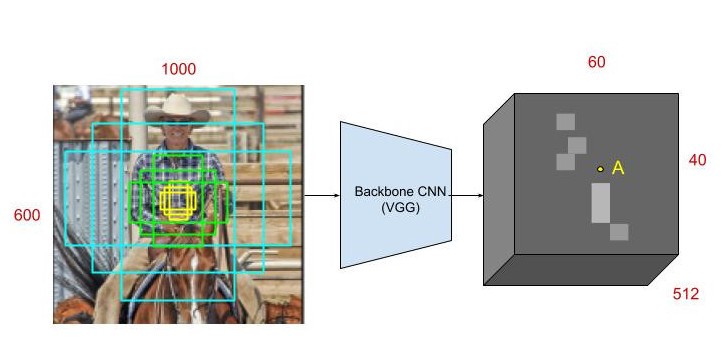


Figure 16: At point A in the feature map, 9 anchors using 3 scales and aspect ratios are generated centered at the corresponding point in the input image.

The 3x3 sliding window has the form of a 3x3 convolution with 512 units, which is applied to the HxWx512 feature map, meaning each of the 512 units uses 512 kernels with size 3x3. The output is fed to two sibling 1x1 convolutional layers that carry out the “objectness” classification and the bounding box regression, respectively.

The classification layer has 18 units as it outputs two values for each of the 9 anchors: The probability of that anchor containing an object and the probability of containing the background, obtained by using the **softmax** activation. Thus, the final output is a HxWx18 feature map, that indicates the probability of there being an object inside of each of the 9 anchors in each point of the convolutional feature map.

The regression layer has 36 units, encoding the coordinates of the 9 anchors of each point of the feature map with 4 values. The regression is used to improve the coordinates of the initialized anchors that have fixed scales and aspect ratios and is only activated if the ground truth of the box containing an object is true.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Figure 17: Architecture of a Region Proposal Network (RPN).

The total output consists of about 40x60 locations with 9 anchors in each, meaning around **20.000 anchors per image**. **During training**, however, anchors that cross the image edges are not used, leaving around 6000 anchors per image. From those anchors, RPN uses only the ones with the highest IoU with a ground truth box, or the ones with more than 0.7 IoU with any ground truth box. During test time, Non-Max Suppression is applied to the 20.000 bounding boxes, removing boxes with more than 0.7 IoU with a higher scoring one, giving around 2000 region proposals by the end.

Finally, those region proposals are fed to a Fast-RCNN, extracting them from the initial feature map, applying RoI pooling, being fed to a set of fully connected layers and then to the final bounding box regressor and classifier. Unlike the RPN regressor that has 4 different regressions (one for each coordinate) for each scale and aspect ratio combination, the bounding box regressor has 4 different regressions for each of the final possible classes.

**RPN only takes around 10 milliseconds to compute region proposals**, a considerable speed up compared to Selective Search, which takes more than one second. Faster R-CNN test time for a single image is below half a second, being able to run detections on real time videos at around **17 frames per second** using.

### Mask R-CNN

**Mask R-CNN** builds up on Faster R-CNN, to solve a different kind of problem, called **Instance Segmentation,** while still detecting objects using bounding boxes**.**

Segmenting an image, as seen in the Selective Search explanation, means dividing the image in groups, usually depending on color, shape, and texture. **Semantic Segmentation** tries to group each object of the same class in the same group, painting the objects pixel by pixel, and coloring the objects of the same class with the same color. Those pixel-by-pixel groups are called segmentation masks. Instance Segmentation decouples those groups of the same class to have a separate segmentation mask for each singular object, having a different color each.

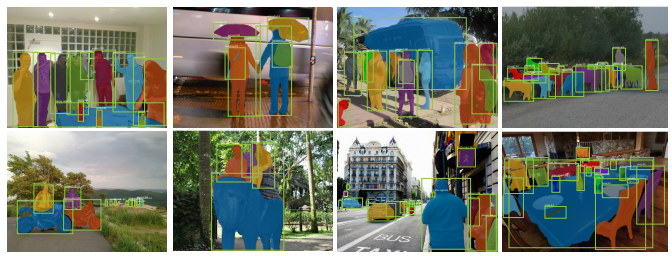


Figure 18

To predict segmentation masks, Mask R-CNN adds a parallel **fully convolutional network** to **Faster R-CNN,** using the same extracted RoIs that are used as input for the image classification and bounding box regression. In Faster R-CNN, those extracted RoIs go through a RoI Pooling layer, that converts each extracted RoI into a feature vector of the same size, as described in the Fast R-CNN section.

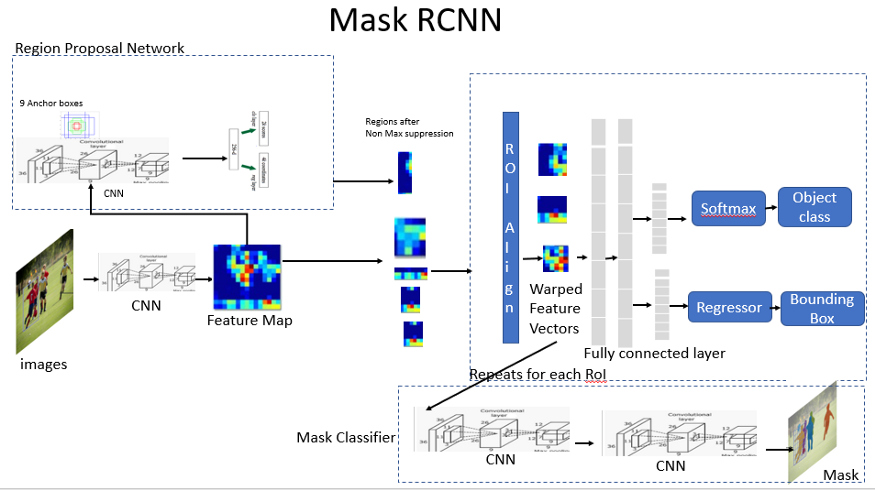


Figure 19

However, **RoI pooling causes misalignments** from the feature map as it uses rounding, which is not important for classification and bounding box detection but has a negative impact when trying to predict pixel-by-pixel segmentation masks.

For example, if the RoI upper left corner is at pixel (26, 37), and the CNN that outputs the feature map has a total stride of 16, the corresponding pixels in the feature map would be at (24/16, 37/16) = (1.625, 2.3125), which, when using RoIPooling is rounded to (2, 2) and the value from that pixel from the feature map is used.

**Mask R-CNN replaces RoI Pooling with RoIAlign**, which avoids rounding the coordinates by using bi-linear interpolation, so that the output is aligned correctly with the input (feature map). In the previous example, the value for the upper-left corner of the roi would be the **bi-linear interpolation** of the four nearest pixels, in that case the pixels at (1, 2) , (1, 3), (2, 2) and (2, 3) in the feature map, which is the **weighted average of the values of those four pixels**, the weights are determined by the distance between the target point (1.625, 2.3125) and each of the 4 nearby pixels.

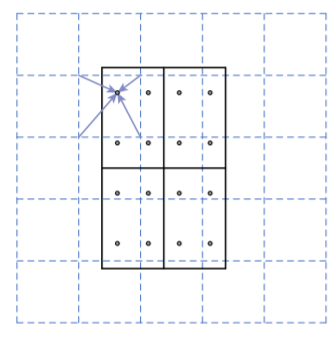


Figure 20: If the final RoI size is 2x2 (solid line), the original RoI is divided in 4 sections, and 4 regularly sampled points are used in each section (dots). The value of each sampling point is obtained using the bi-linear interpolation with its four nearest points in the feature map (dashed grid). Finally, MaxPooling or AveragePooling is used on those four sampling points to get a final RoI size of 2x2.

From the RoI align output, a CNN branches out to predict the segmentation masks **in parallel** to the object classification and bounding box regression. The architecture of this CNN is straight-forward and depends on the backbone used for the feature map generation, which can be ResNet50, which is the one used originally by Faster R-CNN, or **FPN** (**Feature Pyramid Network**) which uses convolutions of different scales and connect them horizontally to a neural network. FPN has residual layers like ResNet so the backbone is called **ResNet-FPN** and is the main backbone for Mask R-CNN.

Diagrama

Descripción generada automáticamente

Figure 21

For each RoI the Mask network outputs **k segmentation masks**, one for each possible class, and the result of the Faster R-CNN object classifier decides which mask will be the final output. During inference, only masks from the 100 highest scoring predicted bounding boxes are generated, speeding up the test time. Overall, Mask R-CNN adds an overhead of 20% to the Faster R-CNN computation time, and generally runs at **5 fps** in real time detection.

### Feature Pyramid Networks (FPN)

Historically, image pyramids were used for testing in object detection tasks, which consisted in using copies of the same image at different scales and detect objects in figure

each of them separately to check if the model can detect the object at different scales. The feature maps generated at each scale would be different, with bigger scales resulting in more **low-level features** (meaning, more specific and less generalizable), than smaller scales.

To train more precise object detectors for multiple scales, Feature Pyramid Networks emulates and improves image pyramids by generating feature maps of the same image at multiple scales, created by a bottom-up CNN with a scaling step of 2 in a process called **bottom-up pathway**. Although many layers produce feature maps of the same size, those layers are considered the same network stage. The output of the final layer of each stage is considered one of the feature maps to be used. Those feature maps extract low-level features at the bottom scales, and **higher-level features** as they go up.

Diagrama

Descripción generada automáticamente

Figure 22: Differences between using convolutions on different scales of the same images (a), a normal multi-layer convolutional feature map extraction (b), using the pyramidal feature hierarchy of a convnet for predictions to emulate image pyramids (c) and Feature Pyramid Networks (c)

FPN use residual blocks like **Resnets,** meaning that the outputs of a layer are added to the inputs of other layers. The activations of the final feature maps of each stage are used as residual inputs and are called (C2, C3, C4, C5) for the convolutional layers conv2, conv3, conv4 and conv5, which have strides of 4, 8, 16 and 32 pixels with respect to the original image. The output of conv1 is not used as it would need more use of memory. FPNs often use a **ResNet-50** as a bottom-up pathway.

The network follows the first stage with a **top-down pathway with lateral connections,** which take the last output of the bottom-up pathway, which is the smaller in scale, passes it through a 1x1 convolutional layer to reduce its channel dimensions and upsamples its spatial resolution by a factor of 2, generating the next upscaled feature map. Each upscaled feature map has a lateral connection with its sibling feature map from the bottom-up path, which has the same resolution. The sibling bottom-up feature map goes through a 1x1 convolution layer to reduce the cannel dimensions and is added to the upsampling result from the previous top-down feature. Finally, the merged maps go through a 3x3 convolution resulting in the final next top-down feature map, which is upsampled again and the process is repeated until the largest map is generated. The final set of feature maps is called (P2, P3, P4, P5) corresponding to (C2, C3, C4, C5).

Diagrama

Descripción generada automáticamente

Figure 23: Lateral connections add the bottom-up feature maps to the top-down upsampled features.

Feature Pyramid Networks are used as a new backbone feature map for **Region Proposal Networks (RPN)** for Faster R-CNN and Mask R-CNN models. The RPN layers that predict the bounding box and objectness score are attached to each of the feature maps from the FPN, sharing the same trained parameters. Instead of using anchors of multiple scales at each step of the RPN sliding window, it only uses one scale for each of the feature pyramid levels as they already capture features at different scales. When used for RPN, FPN generates one extra feature map called P6, which is simply a stride two upsampling from P5, with no lateral connections. Fixed scales with areas of (322, 642, 1282, 2562, 5122) are used for (P2, P3, P4, P5, P6) respectively, with 3 aspect ratios of (1:2, 1:1, 2:1), resulting in a total of 15 total anchors over the pyramid.

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente

Figure 24

FPN is also used as the backbone feature map for Fast R-CNN, which is used as the final object detection stage of Faster R-CNN and Mask R-CNN models. Instead of extracting ROIs from one single feature map, they are extracted from one of the FPN levels depending on the height and width of the ROI. Larger ROIs are extracted from lower, bigger levels, and smaller from higher ones. Those ROIs go through ROI pooling to become a 7x7 feature and are fed to two fully connected layers to predict bounding box coordinates and class labels as usual.

## One-Stage Object Detection

Although Faster R-CNN gets closer to real-time object detection, it does not get near to the standard 24 fps in movies or 60 fps or more that modern videos can achieve. Even after optimizing the test time, Faster R-CNN and similar models still rely on a region proposal stage before the final classification and bounding box regression stage, thus being called two stage object detection models.

One-Stage object detection models try to detect objects without the region proposal stage, increasing the detection speed and achieving real time object detection.

### You Only Look Once (YOLO)

While models such as Faster R-CNN use a pretrained object classification backbone on region proposals and bounding box regression separately **YOLO** approaches object detection as a single regression problem by predicting the class and bounding box coordinates in the same layer and by processing the input image **only once**.

To do this, YOLO splits the image into an **S x S grid**, for each grid cell, it predicts **B bounding boxes centered at that cell** encoded as x, y, w, h, where (x, y) are the coordinates for the **center of the bounding box**, and (w, h) represent its width and height, respectively. For each bounding box, YOLO predicts one **confidence score** that reflects how confident the model is that the box contains an object, which is similar to the “objectness” score from the RPN. This score is equal to the highest IOU of that box with any ground truth box. Finally, at each cell the model also predicts **C class probabilities**, which are multiplied by the confidence score at test time to get a final detection probability. Final output boxes are filtered using the usual method of **non-max suppression** and their class is the highest scoring class probability in the grid cell where the box is centered.

YOLO was trained with the parameters S=7, B=2 on the Dataset Pascal VOC, which has 20 labelled classes, so C=20 and the final output tensor shape was 7 x 7 x (5 x 2 + 20) = 7 x 7 x 30.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figure 25: YOLO divides the original resized image into a S X S grid, predicts B bounding boxes and C class probabilities for each cell and a confidence score for each box. These predictions are encoded as an S x S x (5 x B + C) tensor.

YOLO is implemented as a convolutional neural network with no pretrained backbone. It has 24 convolutional layers followed by 2 fully connected layers, with the final layer generating the output as a 7 x 7 x 30 tensor.

Diagrama

Descripción generada automáticamente

Figure 26: The YOLO network architecture. The image only goes through the convolutional network once.

As YOLO has no pretrained backbone, the first 20 layers, followed by an average pooling layer and a fully connected layer, were first trained on the classification dataset **Imagenet** as a classification model, and then those learned features were used on the final YOLO model to learn object detection faster. The process of using pretrained models and changing the final layers to get a different output is called **Transfer Learning** or **fine tuning** and is useful to get good results with only a few trainable layers as the general features were learned on the pretrained ones.

By avoiding splitting the problem into two stages, YOLO achieves an inference speed of **45 frames per second**, beating Faster R-CNN by a big margin and achieving real time object detection with good detection accuracy of **63.4 mean Average Precrision (mAP)**, while **Faster R-CNN had more than 70** **mAP**.

The main limitation of YOLO is that it has spatial constrains, **as each grid cell can only predict one class** and two bounding boxes, struggling with small objects that appear in groups such as flocks of birds. YOLO also struggles with bounding box sizes and aspect ratios that has not learned on its training dataset, as it **learns the boxes coordinates from scratch**, instead of using anchor boxes like Faster R-CNN.

### YOLOv2

YOLOv2 (or YOLO900) improves the original YOLO accuracy and speed by tuning the neural network layers, training the network with randomized image rescaling, adding a residual layer from a bigger feature map to the final one and adding **batch-normalization layers,** which **standardize** the layer outputs of a batch, making them have a mean of 0 and a standard deviation of 1.

But the biggest change in YOLOv2 is the adoption the **anchor box** system from Faster-RCNN (RPN) instead of having the network predict the boxes coordinates from scratch. YOLOv2 also **decouples the class prediction from the cell location**, predicting it for each anchor box along with the objectness score instead. As the original YOLO only predicted 2 boxes per cell grid on a 7x7 grid, it predicted a total of 98 bounding boxes. With anchor boxes, YOLOv2 predicts more than a thousand.

Instead of setting anchor box sizes using handpicked scales and aspect ratios, a **k-means algorithm** is run during test time on the test dataset to find the most common bounding box shapes, using the k-means centroids as the anchor-box sizes. YOLOv2 also tunes the anchor box system by making it predict the box locations relative to the location of their grid cell instead of the full image.

Imagen que contiene Forma

Descripción generada automáticamente

Figure 27

YOLOv2 has a modified backbone from YOLO, called **Daknet-19** is trained on the image classification dataset ImegeNet with **1000 different classes**. Using this pretrained network before training the model for detection after changing the last layers makes the model able to detect objects of classes that are not included on the detection dataset.

Tabla

Descripción generada automáticamente

Figure 28: Darknet-19 architecture with the image classification head

### YOLOv3

YOLOv3 improves YOLOv2 mainly by using a new backbone, called **Darknet-53**, which makes use of **residual layers** and does predictions at **different scales** in a similar way to **Feature Pyramid Networks** (FPN).

Tabla

Descripción generada automáticamente

Figure 29: Darknet-53 architecture with the image classification head

Like YOLOv2, this iteration uses anchor boxes with predicted sizes using k-means. YOLOv3 will predict 3 bounding boxes at 3 different scales, so this time, 9 centroids are used as pre-chosen anchor box dimensions, called priors, and are divided evenly across 3 arbitrarily chosen scales. The last backbone feature map, which has the lower scale, will go through added detection convolution layers to predict 3 anchors boxes, where each has 4 coordinates, 1 objectness score and 80 class probabilities, resulting in a tensor of dimensions N x N x [3 \* (4 + 1 + 80)] where N x N is the number of grid cells the image is divided into.

After the first set of detected boxes, the feature map from two layers before that result is upsampled by 2 and is added to a feature map from earlier in the network, which goes through the detection layers and predicts three more bounding boxes, resulting in a tensor twice the size of the previous one. This process is repeated with a previous feature map and a result of 9 bounding boxes is predicted.

YOLOv3 achieves an inference time of less than 50 milliseconds, achieving between 30 and 50 fps on videos, while having a mean average precision mAP on the COCO dataset (also called COCO AP) between 28 and 33. At the time of its release, YOLOv3 was slightly less precise than the state-of-the-art detection models (in terms of precision) such as Faster R-CNN, but was a lot faster.

Gráfico, Diagrama

Descripción generada automáticamente

Figure 30: Comparison between precision and inference time of detection models in respect to YOLOv3

### YOLOv4

By the time YOLOv4 was released (2020), most object detector models had a similar structure where there was a backbone, neck, dense prediction and sparse prediction. The **neck** are layers between backbone and head that carry out the collection of feature maps at different stages of the backbone, having bottom-up and top-down paths like an FPN. The **dense prediction** stage is where bounding boxes are predicted from the layers of different scales of the neck, like **RPN** for two stage detectors and **YOLO** for one stage The **sparse prediction** stage is exclusive to two stage detectors, where the dense predictions (**ROI**s) go through the final detection layer, like in Fast R-CNN.

Gráfico, Gráfico radial

Descripción generada automáticamente

Figure 31

YOLOv4 groups its improvement methods into two categories. The first one, called **bag of freebies,** includes methods that increase the model accuracy without increasing the inference cost by only affecting the training stage. **Data augmentation and Regularization** methods are part of this group. Data augmentation makes the model less prompt to overfit on the training data, as the data is changing randomly with random transformations such as rotations, change of brightness, size, etc. YOLOv4 uses **CutMix**, which cuts a random region from an image and fills them with a patch from other one, and **Mosaic** data augmentation, which takes blocks of 4 different images and merges them into one. Another data augmentation technique used by YOLOv4 is Self-Adversarial Training (SAT), which modifies images in a similar way to Generative Adversarial Networks.

Perro parado junto a un gato

Descripción generada automáticamente con confianza mediaImagen que contiene foto, diferente, bicicleta, estacionado

Descripción generada automáticamente

Figure 32: Examples of CutMix (left) and Mosaic (right) data augmentation. Source: Google Images

There are techniques like data augmentation but applied to feature maps instead of the original image that serve as **regularization** methods, which function is to avoid overfitting on the training data. On popular method in deep learning is **DropOut**, where random pixels from feature maps are converted to 0 (black), or **DropBlock**, where blocks of pixels are set as 0 instead of individual ones. YOLOv4 uses DropBlock regularization. **Class label smoothing** is another method of regularization, which consists of altering the class prediction results to prevent having one class probability being much larger than the others.

Un dibujo de un perro

Descripción generada automáticamente con confianza media

Figure 33: Examples of DropOut (b) and DropBlock (c) regularization methods on extracted feature maps from an image (a)

Another element from the bag of freebies is Cross Mini-Batch normalization, which is a modified version of batch normalization where the mean and the standard deviation are computed, normalized and acummulated to subsequent mini-batches inside the same batch.

The last element of the bag of freebies is the loss function for bounding box regression. Traiditional object detectors use the Mean Squared Error (MSE) of the 4 bounding box coordinates. However, this method treeats each coordinate as an independent variable and does not take into account the integrity of the object. To fix this issue, **loss functions based on** **IoU** are proposed. YOLOv4 uses **Complete IoU Loss (CIoU)**, which takes into account the IoU, the distance between center points and the aspect ratio similarity between the predicted and the ground truth box.

Dibujo de una persona

Descripción generada automáticamente con confianza baja

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

Figure 34: CIoU loss function (top), where ρ(b, bgt) is the Euclidean Distance between the boxes central points and, c is the diagonal length of the smallest enclosing box covering the two boxes, α is a positive trade-off parameter, and v measures the consistency of aspect ratio (bottom).

The other group of improvement methods is called **bag of specials**, and include methods that improve the model precision while increasing inference time by a small ammount.

One of the modules in the bag of specials is the **enhancement of the** **receptive field**, which is the size of the input image pixels that a final feature map output value depends on. For example, the result of one convolutional iteration with a 3x3 kernel in the original image has a 3x3 receptive field. If one feature has a larger receptive field, it means its information contains a bigger context of the image.

Gráfico

Descripción generada automáticamente

Figure 35: Example of receptive fields with 3x3 kernel size convolutions. Green cells in Layer 1 represent the receptive field of the green cell in layer 2, and green and yellow cells from Layers 1 and 2 represent the receptive field of the yellow cell in Layer 3.

**Spatial Pyramid Pooling (SPP)** is one of the methods that enhances the receptive field. SPP is similar to ROI Pooling but with multiple scale outputs, it uses a feature map and divides it into multiple N x N grids with different values of N, and then uses max-pooling on the pixels inside each separate cell, flattens each grid output and concatenates them in a single one dimensional vector. As a one dimensional output is not fit for using it as a feature map, YOLOv4 uses a modificed version of SPP by converting it into a concatenation of max-pooling outputs of k x k sized kernels where k = {1, 5, 9, 13} and stride 1. This results in multiple feature maps of the same size but with different receptive field sizes. The SPP module is added after the final layer of the backbone.

Escala de tiempo

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

Figure 36: Representation of Spatial Pyramid Pooling (SPP) (left). YOLOv4 modifies SPP to have feature maps as output (right).

Like **FPN**, YOLOv4 uses bottom-up paths and top-down paths with lateral connections. Instead of using FPN like YOLOv3, YOLOv4 uses a modified version of **Path Aggregation Networks (PANet),** that includes another bottom-up path after the top-down path with lateral connections, called **bottom-up path augmentation**. YOLOv4 modifies PAN lateral connections to **concatenation** instead of addition. The PAN takes outputs from different layers of the backbone as well as the SPP output as its smallest scale input (the top of the top-down path), and **the final feature maps from each PAN stage are fed to a YOLOv3 head** that uses 3 anchor boxes on each stage to predict a total of 9 bounding boxes per pixel, each with their objectness score and class prediction.

Forma, Polígono

Descripción generada automáticamente

Figure 37: Original PAN aggregation method (left) and the YOLOv4 PAN modification (right). Source: []

An additional block is added to the neck, the **Spatial Attention Module (SAM),** which makes makes the use of the attention mechanism. Attention is referred as a mechanism in deep learning where one feature “looks” at previous inputs. For example, in a neural network that translates sentences from one language to another, the output translation of a certain word might depend on previous words, so the network has an attention mechanism where it considers the importance of previous words for that input, usually in the form of a vector with the same size as the input size, with the values representing the importance ratio between 0 and 1. Spatial Attention translates this mechanism to images or feature maps, where each pixel has its attention value.

SAM does this by concatenating the results of a max-pooling and average-pooling layers over the input, and applying a convolution with a sigmoid activation function, resulting in the spatial wise 2D attention map, which is multiplied with the original 3D input, having each channel value multiplied by the attention value of its (x,y) position, to get the final result. YOLOv4 modifies SAM by avoiding the max-pooling average-pooling layers, and instead using one convolution layer with the output of the same size as the input and with the sigmoid activation to get a **3D attention map**, which multiplies each pixel of the input by the attention pixel at its (x, y, z) position. This modified SAM block is added in the **PAN** module after each lateral connection between the bottom-up path augmentation and the previous top-down path.

Imagen que contiene reloj, objeto

Descripción generada automáticamente

Figure 38: The original SAM (top) and modified SAM for YOLOv4 (bottom). Source: []

Another element in the bag of specials is the change of the **activation function** for most layers from ReLU, which outputs 0 for every negative input, to **Mish**, which can make the gradient more efficiently propagated without much extra computational cost and proved to be more effective for YOLOv4. The **Mish activation function** is the following: **f(x) = x\*tanh(softplus(x))**, where the softplus function is the following: **softplus(x) = ln(1+ex).**

The last element in the bag of specials is the modification of the **Non-Max Suppression (NMS)** algorithm. YOLOv4 uses **DIoU NMS** instead, which uses **Distance IoU (DIoU)** to filter out boxes instead of normal IoU. DIoU is the **CIoU** formula without the aspect-ratio parameter.

Texto

Descripción generada automáticamente con confianza media

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

Figure 39: DIoU formula (top) and DIoU NMS (bottom). Source: []

Diagrama, Esquemático

Descripción generada automáticamente

Figure 40: Inside the YOLOv4 SPP + PAN neck and YOLOv3 head. Source: corrected image personally from []

YOLOv4 uses the same backbone as YOLOv3, the DarkNet-53, but it modifies it by using **Cross-Space Spatial connections (CSP),** which splits feature maps into two copies, has each copy go through convolutional layers and concatenates the results of both copies afterwards.The backbone is then renamed to **CSPDarkNet-53**.



Figure 41: Architecture of CSPDarknet-53. It has 5 CSP blocks that contain a different number of residual blocks. The outputs of the last 3 CSP blocks is fed to the PAN neck, having the last output go through the SPP first. Source: []

YOLOv4 has the mindset of being able to be trained and tested on single standard GPUs while achieving real time inference and became the start of the art of real-time detectors when it released in 2020, achieving **43.5% mAP** on COCO dataset at a real time speed of **65 FPS** on a Tesla V100 GPU.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Figure 42: Comparison of YOLOv4 and other state of the art object detectors at the time of its release.

### YOLOv5

What started as a **YOLOv4** **Pytorch** implementation with few changes has now become the most popular real-time detector. The nature of YOLOv5 is quite different from previous models, as **it is not a research project and does not have a research paper**, instead, it has a whole company behind its **continuous development** and is being constantly updated with tweaks to the architecture or adding support with different third-party software. YOLOv5 also features models of different “size” options, from faster and less precise models to slower and more precise ones that uses more different feature map scales for anchor box prediction.

Diagrama

Descripción generada automáticamente con confianza media

Figure 43: Size options for YOLOv5 with their COCO precision and inference speed on a Tesla V100 GPU. Source []

The continuous development and improvement of YOLOv5 makes it difficult to describe its changing architecture with detail, which can be checked in the source code, but **its general structure is the** **same as YOLOv4** with small changes to layer placement and parameters. The bigger options of YOLOv5 at the 6.0 version include **1 extra output layer**, changing the downsample and upsample rates to use 4 feature maps with lateral connections and predicts boxes at **4 different scales** instead of 3.

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente

Figure 44: The general architecture of YOLOv5 is the same from YOLOv4. Source []

YOLOv5 replaces the **Mish** activation function with **SILU**, which is equal to the **sigmoid function** multiplied by its input, giving a similar output to RELU, has a modified SPP block, called **SPPF,** uses different formulas for box coordinates calculation and uses a different method for anchor box templates assignment to grid cells by having cells use ground truth boxes centered at the edge of an adjacent cell. YOLOv5 also uses more data augmentation methods

Diagrama, Esquemático

Descripción generada automáticamente

Figure 45: YOLOv5 6.1 detailed architecture. More details on YOLOv5 6.0 Can be found at the source: <https://github.com/ultralytics/yolov5/issues/6998>

Interfaz de usuario gráfica, Gráfico

Descripción generada automáticamente

Other YOLO variants

Scaled-YOLO

YOLOR

Other One-Stage object detection methods

SDD

Retinanet

EfficientDet

Faster RCNN fps results

My pc = 6 fps with fast model (min\_size=512)

Idiada pc = 14.5 fps with fast model (min\_size=512)

Mask RCNN fps results

My pc = 2.455 fps with fast model (min\_size=512)

Idiada pc = 3.145 fps with fast model (min\_size=512)

## Evaluation of Object Detection models

Although this project focuses on the inference speed of object detection models to apply them on real time videos, it is also important to evaluate how accurate those models are at detecting objects.

For normal image classification, metrics such as **Accuracy, Precision and Recall** are usually used. **Accuracy** is the most straight forward metric as it simply measures the percentage of correct predictions by comparing them to the ground truth. For **object detection**, precision and recall are more important.

### Precision

Glossary

**CNN**: Convolutional Neural Network, a neural network with convolutional layers.

**FPN**: Feature Pyramid Network, a neural network with bottom-up and bottom-down feature map pyramids with lateral connections

**IoU**: Intersection over Union

**mAP or AP**: mean Average Precision, an evaluation method for object detection models that computes the area under the precision-recall curve.

**NMS**: Non-Max Supression, filtering out bounding boxes with lower IOU with the same ground truth box.

**R-CNN**: Region based CNN, a two-stage object detection model that uses CNN and ROIs

**ROI**: Region of Interest, a rectangle fragment from an image where there are potential objects.

**RPN**: Region Proposal Network, a neural network that predicts ROIs

**YOLO**: You Only Look Once, a one stage object detection model that only processes an image once.

Other Object Detection architectures:

Cascade R-CNN

Single shot Detection

Feature Pyramid Networks

EfficientDet

RetinaNet

2 Stage

Sliding window

Selective Search

RCNN

Faster RCNN

Mask RCNN

1 Stage

YOLO

Transformer? SWIN

Sources:

R-CNN paper: <https://arxiv.org/pdf/1311.2524.pdf>

Fast R-CNN paper: <https://arxiv.org/pdf/1504.08083.pdf>

Faster R-CNN paper: <https://arxiv.org/pdf/1506.01497.pdf>

Faster R-CNN analysis <https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46>

Mask R-CNN paper: <https://arxiv.org/pdf/1703.06870.pdf>

Feature Pyramid Networks (FPN) paper <https://arxiv.org/pdf/1612.03144.pdf>

YOLO paper: <https://arxiv.org/pdf/1506.02640.pdf>

YOLOv2 paper: <https://arxiv.org/pdf/1612.08242v1.pdf>

YOLOv3 paper: <https://arxiv.org/pdf/1804.02767.pdf>

YOLOv4 paper: <https://arxiv.org/pdf/2004.10934.pdf>

CIoU Loss paper: <https://arxiv.org/pdf/1911.08287.pdf>

YOLOv4 architecture images: <https://github.com/AlexeyAB/darknet/issues/5891#issuecomment-695117608>

CSPnet paper: <https://arxiv.org/pdf/1911.11929.pdf>

YOLOv5 v5.0 architecture images: <https://blog.csdn.net/Q1u1NG/article/details/107511465>

YOLOv5 6.0 architecture and other details: <https://github.com/ultralytics/yolov5/issues/6998>

Performance Analysis: YoloV5 vs YoloR: <https://towardsai.net/p/l/performance-analysis-yolov5-vs-yolor>

Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs: <https://www.mdpi.com/1424-8220/22/2/464>

YOLOv5 focus layer: <https://github.com/ultralytics/yolov5/discussions/3181m1>

Non-max supression: <https://towardsdatascience.com/intuition-and-implementation-of-non-max-suppression-algorithm-in-object-detection-d68ba938b630>

<https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

YOLOV4 Research review <https://www.thinkautonomous.ai/blog/?p=introduction-to-yolov4-research-review>

<https://towardsdatascience.com/object-detection-with-convolutional-neural-networks-c9d729eedc18>

mean average precision mAP

<https://www.youtube.com/watch?v=FppOzcDvaDI>

<https://kharshit.github.io/blog/2019/09/20/evaluation-metrics-for-object-detection-and-segmentation>

Roi Pooling

<https://deepsense.ai/region-of-interest-pooling-explained/>

Mean Average Precision:

<https://blog.paperspace.com/mean-average-precision/amp/>

Guias para usar Faster RCNN y Mask RCNN con pytorch:

<https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html>

<https://debuggercafe.com/faster-rcnn-object-detection-with-pytorch/>

<https://debuggercafe.com/custom-object-detection-using-pytorch-faster-rcnn/>

<https://debuggercafe.com/a-simple-pipeline-to-train-pytorch-faster-rcnn-object-detection-model/>