STATE-OF-THE-ART DEEP LEARNING MODELS FOR COMPUTER VISION IN SPACE MISSIONS

Alfonso Cavallo
Master SPICES
University of Bologna
alfonso.cavallo98@gmail.com

December 30, 2025

ABSTRACT

Image processing is a fundamental task in space missions for both ground-based and on-board systems. Computer Vision studies the capability of computer programs to process visual input such as images and videos. Traditional rule-based software methods for image processing have nowadays been replaced by data-driven Machine Learning models. This paper provides the reader with a technical overview of the working principle of Deep Learning approaches, in particular Computer Vision oriented Convolutional Neural Networks. Furthermore, it explores today's state-of-the-art neural architecture for object detections and how they are applied space missions.

1 Introduction to Machine Learning and Deep Learning

Ground-based space observation and space-borne remote sensing are fundamental capabilities in space missions for surveillance and exploration tasks. They are usually addressed by means of imaging sensors like cameras and radars, whose output is visual data such as an image or video. Computer Vision (CV) is the field of computer science that studies the capability of machines to process visual inputs [1] by means of software programs. Traditional software development is rule-based. Software developers start from requirements and have knowledge about the input and output domains of the program, and of the law that should process the data into the desired output. This paradigm has always been used for the realization of Artificial Intelligences (AI) and CV, making their logic and cognitive processes mostly interpretable. The recent evolution of hardware, in particular of Graphical Processing Units (GPU) unlocked an extreme high-performance parallel computing capability, enabling the execution of programs which could only be theorized in the past, such as Machine Learning (ML) models and training algorithms. According to IBM [2], ML is a subset of AI focused on the implementation of algorithms that are able to learn the models and relations from data, and use them for later inference, mostly for pattern recognition. This paradigm is substantially different from traditional software engineering. First of all, it is a data-driven approach, which means the AI get to learn how to accomplish a task without the explicit design of an expert. The first ML models were as simple as linear regression [3] and k-nearest neighbors [4]: elementary models with limited representational capabilities.

Nowadays, we are witnessing the Deep Learning (DL) revolution [5], in which algorithms with an enormously high representation capability, the Neural Networks (NN) [6], dominate the scene. Some trace the advent of this era to ImageNet challenge [7] in 2012. This technological boundary push comes at the price of the need for potentially huge amounts of significant data and processing capability. The obtainment of such assets is arguably the single most important problem of the ongoing industrial revolution. Nowadays, the AI field is just starting to address the passage from research to product, whenever possible. CV is one of the fields getting the most benefits from the DL revolution, where foundation models are starting to affirm themselves as concrete and deployable solutions rather than promising innovations.

This paper will introduce the fundamental working principles of Neural Networks and why they are replacing older traditional Computer vision methods. It will then delve deeper into the Convolutional Neural architecture and the state-of-the-art solutions for object detection and recognition: YOLO and ViT. Finally, some academic and industrial

research and solutions where such novel technologies are applied in space will be explored to understand the impact of the deep learning revolution on space missions.

2 Neural Network Modeling and Training

In the context of DL for CV, NNs are the most applied ML models. To better understand what a NN is, it is necessary to comprehend how a neuron, which is its fundamental block, works. An artificial neuron [20] is a bio-inspired construct that takes a feature vector in input and returns one single output which is an "activated" result of a linear regression performed on the input. In summary, it can be expressed as:

$$y = \sigma(\bar{w} \cdot \bar{x} + b) \equiv \sigma(W \cdot \bar{x}) \tag{1}$$

where y is the output; x is the multi-dimensional input; $\bar{w}$ and b are weights and bias of the linear regression; σ is a non-linear function called activation function; W is a simplified notation that includes both $\bar{w}$ and b. The features of x can be anything continuous or categorical that can be numerically quantifiable, and from which it is possible to infer the output. A neuron basically carries out a weighted sum, modeling a simple linear relation between the features and the output, with the addition of a non-linearity. Training a neuron is learning the parameters (weights and biases) that make it model a desired relation, which we provide the model with by examples rather than direct design. ML involves many different kinds of learning, but when it comes to neurons, it usually boils down to a form of supervised learning [8]. Supervised Learning involves the employment of labels, which are the expected output that the model should return for a given input data. In this frame, the optimization process is all about adapting the model to provide the desired output in response to that given input, trying to force it to learn a general relation that is also applicable to all data inside the dataset, and possibly even outside. Self-supervision [9], is a very specific paradigm where a processed version of the input is used as its own output label, and is recognized as half-way between supervised and unsupervised learning.

Among all the training algorithms, especially for supervised learning, Stochastic Gradient Descent (SGD) is surely the most famous. It involves the definition of a loss function to compute the error between the output of the neuron and the desired ground-truth label. Once the function is defined for one input data, it is possible to compute the gradient of the loss with respect to the parameters of the neurons. Basically, the gradient represents the direction toward which, by moving the parameter vector, the maximum error increase is achieved for that data sample. Intuitively, to move them into the opposite direction is to reduce such error, hence modelling the correct relationship. Formally, we can express this learning process with the following update formula:

$$W_t = W_{t-1} - \lambda \cdot \nabla_W \mathcal{L}(\hat{y}, y) \tag{2}$$

where W_t are the parameters after the update; W_{t-1} are the parameters before the update; λ is the learning rate, a coefficient defining how strong should the weight update be; ∇_W is the gradient operator in the parameters W; $\mathcal{L}$ is the loss function; $\hat{y}$ and y are the prediction and the ground truth, respectively. Trivially, the activation function has to be differentiable in the parameters domain. Indeed, huge relevant literature studies the optimization process and better optimizers, but still SGD represents the foundation of most modern research.

However, in its naive representation, one neuron has a limited representational capability. It is worth noting that this is not always a drawback, as simple models have a tendency to model general and simple relationships. Complex models, on the other side, can be induced to fit the data by approximating undesirably specific laws that are inapplicable outside of the training set, in a phenomenon called overfitting. Still, complex relations require complex modeling. To unlock more representational power, combining multiple neurons into one big network, a NN indeed, is necessary. A NN is a sequence of layers, each of which is made of multiple neurons. The neuron of each layer is connected to all the neurons of the next layer, such that the output can become input for all of them in a densely connected network. A simple NN is made of at least three layers, as also illustrated in Figure 1:

1. **Input Layer**: it is a virtual layer, where each neuron's output is actually one feature of the input;
2. **Hidden Layer**: it is the layer responsible for processing the input (or intermediate, whether multiple hidden layers are concatenated) features;
3. **Output Layer**: it is the layer responsible for aggregating the results of all the hidden regressions into the desired output structure.

with this in mind, a NN is just a parametric model, where the parameter set includes all the weights and biases of all the neurons. The choice of this structure as a model is justified by two main reasons. First of all, it is inspired by the

functioning of the human brain, which cognitive capabilities are well-known to us, and lastly because the Universal Approximation Theorem [17] states that, under certain assumptions, the aforementioned NN model is a universal function approximator with an arbitrary accuracy. However, such theorem says nothing about how many neurons are required in the hidden layer, most likely a prohibitive amount from a computational point of view. In this context, Deep Learning is a concept of NN based on the concatenation of multiple hidden layers, each part of a progressive feature refinement process aimed at learning increasingly complex relations in the data. Those are also called a Multi-Layer Perceptron (MLP). This complexity gives a hard time computing gradients. Let us assume the following error:

$$\mathcal{L}(\hat{y}, y) = \mathcal{L}(h_{L-1}, y) \tag{3}$$

where $\mathcal{L}$ is the loss function; y is the ground truth; $\hat{y} = h_{L-1}$ is the predicted output, which is also the output of the L-th hidden layer. We need to compute the gradient in each of the weights w_l for all the layers $l \in 0, 1, \ldots, L-1$. It is first necessary to do a forward propagation and compute all the intermediate hidden representations before performing what is called a backpropagation of the error to optimize the parameters. We start with an element-wise derivation in the weights of the last layer:

$$\frac{\partial \mathcal{L}(h_{L-1}, y)}{\partial W_{L-1}} \tag{4}$$

The computation is not immediate as the W_{L-1} term is not explicit in the error formula. That is where the derivation chain rule comes in rescue:

$$\frac{\partial \mathcal{L}(h_{L-1}, y)}{\partial h_{L-1}} \frac{\partial h_{L-1}}{\partial W_{L-1}} = \frac{\partial \mathcal{L}(h_{L-1}, y)}{\partial h_{L-1}} \frac{\partial \sigma(W_{L-1} \cdot h_{L-2})}{\partial W_{L-1}} \tag{5}$$

This formulation is much easier to solve as h_{L-1} is a direct function of W_{L-1}. Let us now propagate the derivative to the $(L-2)$-th layer.

$$\frac{\partial \mathcal{L}(h_{L-1}, y)}{\partial W_{L-2}} = \frac{\partial \mathcal{L}(h_{L-1}, y)}{\partial h_{L-1}} \frac{\partial h_{L-1}}{\partial h_{L-2}} \frac{\partial h_{L-2}}{\partial W_{L-2}} \tag{6}$$

We have propagated the gradient to the parameters in the $(L-2)$-th but the same rule can be iteratively applied down to the first layer. Once the entire backpropagation is completed, all the NN parameters are updated at the same time, similarly to how it is done in Equation 2. An NN data-driven training typically requires performing this update iteratively for each sample up until the entire dataset is processed, and what is called an "epoch" is completed. Usually, multiple epochs are required before the NN approximation is good enough. This evolution is usually monitored by means of performance metrics over a second dataset called the Validation Set, different from the Training Set, which instead contains the examples used to optimize the parameters. A Test Dataset is eventually employed to measure the definitive and final performance of the model.

In summary, a NN is a network of densely connected neurons that takes in input some features and models a mathematical law through which it processes them into one desired output. This model is parametrized, and its parameters, the weights and biases of all the neurons, are optimized by gradient backpropagation, an algorithm based on SGD. Backpropagation aims to minimize the error between the current output of the NN and the actual desired output over all the examples of a training dataset, in an iterative process. In the end, the model should have fitted to satisfy a relation between the input data and the output data. Such a law should possibly be general enough to be applicable also outside of the Training Set, which can be verified on the external Validation Set and Test Set.

3 Artificial Vision - Convolutional Neural Networks

The purpose of the Computer Vision (CV) or AV is to elaborate, analyze, and interpret images and videos to acquire meaningful information. As a video can be encoded in a sequence of image frames, everything boils down to image processing. In traditional CV, images are usually structured as bidimensional matrices of pixels, where the axes encode the spatial alignment of one pixel along the width and height of the picture. When the image is grayscaled, that is enough to encode the pixel intensity as a number; otherwise, a new axis is required to encode the color of the pixel on multiple channels. This is usually done in three channels that may be RGB , HSV or any other color space [10], but in space missions hyper-spectral images can reach up to thousands of wavelength channels.

Images are complex structures made of recurring morphological patterns. These signals typically manifest as spatial correlations across neighboring pixels and are translation invariant, meaning they recur regardless of their position into the picture. Those properties, as well as many others, can be employed by deterministic algorithms to detect recurring patterns. NNs do not inherently exploit those characteristics. To better understand how their architecture can be adjusted to better handle CV tasks, it is first necessary to understand how CV traditionally performs image processing.

In image processing, there exist three kinds of operators:

- **Point-wise Operators**: functions that are applied on each pixel;
- **Global Operators**: functions that process each pixel based on global information;
- **Neighborhood Operators**: functions that process each pixel based on information gathered on a local neighborhood.

Point-wise operators are typically used for thresholding and spotting inter-channel dependencies, while Global operators are meant to aggregate global information from all the pixels. On the other side, neighborhood operators are delegated to detect local patterns that interconnect close pixels, and are usually implemented as convolutional filters. A convolutional filter can be expressed as a sliding window with a given size, moving over all the pixels in the image. In this window, the element-wise product of the region of the image inside the window and a matrix the same size, called Kernel, is performed. Everything is then reduced to via summation to a single value that is applied to the pixel the window is centered in. It follows the mathematical formalization:

$$(K * I)(x, y) = \sum_{w=-a}^{a} \sum_{h=-b}^{b} K(w, h) \cdot I(x + w, y + h) \tag{7}$$

where $I(x, y) \in \mathbb{R}^C$ is the value on the C channels of the pixel whose coordinates are x, y; K, following the same notation, is the filter Kernel, which can be seen as a small image of the size of the window; a and b are half the width and height of the moving window. Intuitively, the convolution operator is a sort of bidimensional dot product that moves spatially over different windows of the image, that is to say is applied to all the pixels. The dot product is a widely known similarity metric, so it will return high values where there is a strong correlation between the Kernel and the window of the original image. Moving the same Kernel across the image leverages the translation invariance property of semantic structures, while the Kernel encodes the expected local cross-pixel correlation.

In the context of convolutional filtering, the Kernels are the models that can be designed according to a priori knowledge. They have been widely used to look for well-known morphological features, like for facial traits in human pictures. However, in complex cases where pictures are subject to a wide range of morphological recurring patterns and non-collaborative optical conditions in general, manual modeling is prohibitive. This is where NN comes into play to help characterize such structures by deeply analyzing recurrences in the data. However, for as it was presented in Section 2, an MLP approximates a function that takes in input all the features and process them all the way to one or multiple outputs. In CV terms, this is more of a global operator that has a limited scope and barely exploits the aforementioned image morphological properties. On the other side, Convolutional Neural Networks (CNN) follow an image-aware NN design that is able to exploit well-known CV image properties to improve the performance in neural image processing.

A CNN is a neural network in which each neuron is not densely connected with all the neurons in the previous layer but only gets to look at a limited receptive field, which is a spatial window over the input image. This means the linear regression of a neuron can be seen as a Kernel whose parameters are learned through training. CNN uses convolution like in Equation 7, to scans all the input image horizontally and vertically, with two given strides. For such a reason, the output of this operation is a new image with the following size for both width and height:

$$w_O = \frac{w_I - w_K}{s_w} + 1 \tag{8}$$

$$h_O = \frac{h_I - h_K}{s_h} + 1 \tag{9}$$

where w_I and h_I are the width and height of the input image; w_I and h_I are the width and height of the receptive field (Kernel); s_w and s_h are the horizontal and vertical stride with which the window slides; w_I and h_I are the width and height of the output. Hence, the output of one layer in a CNN is another image with size (w_O, h_O) and one channel, called a feature map. In summary, a feature map is the output of one sliding neuron, that is conceptually the same as a layer made of neurons sharing the same weight. This weight sharings makes CNNs far more memory-efficient

that traditional NNs. Using multiple neurons introduces multiple feature maps, each of which could process different morphological structures, expanding the representational power of the model. Conceptually, it is not different from adding more neurons to one simple MLP. Each neuron is basically a neighborhood operator that reduces to a point-wise operator whether its receptive field size is $(w_K = 1, h_K = 1)$. It is worth remembering that neurons also use an activation function to add non-linearity to the input. This step can be just considered a separate point-wise non-linear operator that is applied to all the pixels after they have been processed into the output feature maps.

The output of each layer of neurons is a multi-channel bidimensional latent image that can further be processed by a series of next layers. Thus, the deep learning approach can also be applied to CNN. Eventually, the output of the network will be a latent image with a large number of channels and a small spatial size. This is a compression of what kernel filtering considered a meaningful pattern and may require to be processed into a single output, for example, the classification label of an object. Usually, a traditional dense layer is placed at the end of the chain to aggregate everything as a global operator.

It is worth noting that CNNs exploit all the aforementioned operators, from point-wise to global, to achieve a processing that is morphology-aware. Furthermore, it also uses other traditional operations, like pooling, padding, and normalization, to regularize intermediate representations and avoid the problem of overfitting.

Today, CNNs are some of the most widely tested architecture in both research and industry, and their working principles are at the foundation of modern state-of-the-art solutions, either directly or indirectly.

One of the most important tasks in CV for which CNNs are state-of-the-art solutions is object detection. Object detection addresses the problem of both spotting the presence of one object in an image and locating its position inside of it. Kernel-based filtering, whether traditional or neural like CNN, can be used to scan the images and output high probability values in pixels where an object is present. The task of determining whether one pixel belongs to a certain object or not is called image segmentation. To perform this task is to find the silhouette of the objects. A more commonly used formalization of the problem is Bounding Box (BB) detection, as BB are usually less fine-grained than the silhouette and enough to solve most problems. A BB is characterized by any four properties that can model a square and its discrete location, like, for instance, horizontal coordinate, vertical coordinate, width, and height.

However, objects can appear in a large range of possible poses with highly variable box sizes and locations. For a long time, BB were detected by sliding models with different spatial sizes. A CNN would provide a high probability if the window it covered was indeed capturing a certain object. Post-processing would have then been performed to prune and merge possible BB into more correct representations [11]. Nonetheless, this process was inefficient as spanning through convolution over all the possible window sizes is very computationally intensive. To address this gap, solutions like YOLO and ViT were developed and are the contemporary foundation models for object detection. Additionally, such models can be tuned for many applications like object recognition and identification on top of the detection, which are more related to understand specific objects characteristics and identifiable traits.

4 You Only Look Once (YOLO)

The innovation of YOLO architecture, which stands for “You Only Look Once" lies right behind its name. Different from previous CNN solutions, YOLO aims to find BB in one single inference step, without attempting different window sizes, but by directly outputting the right BB. Across the years YOLO witnessed a continuous evolution and maintenance, leading to the most recent stable architecture, YOLO v11 [23]. This architecture is based on three subsequent modules:

1. **Backbone**: based on CNN layers, its purpose is to extract meaningful convolutional features out of multi-channel and grayscale images.
2. **Neck**: based on a mix of CNN layers and concatenations, its purpose is to further refine the extracted features and prepare them to be used for the detection of patterns at different scales. It also mixes up the layer’s outputs such that each scale-specific feature is not completely unaware of the processing at other scales.
3. **Head**: based on a linear CNN, its purpose is to perform linear regression to output BB and object class information.

All of them are based on low-level neural building blocks, which are introduced along with their technical implementation and purpose in the following list:

- **Conv**: CNN convolutional blocks that work as described in Section 3. They compress the information spatially according to Equations 8 and 9;
- **Bottleneck**: made of two consecutive Conv blocks with 3×3 kernels with 1×1 padding (dummy corner pixels) to keep the same spatial input size in the output. They are called bottlenecks as they compress the

channel size in between the Conv blocks in order to accelerate processing. The output and the input of the block are finally concatenated and returned, in order to preserve the original information;

- **C3K**: A 3×3 Conv block with 1×1 padding followed by a variable length chain of bottlenecks. Everything is concatenated before being finally processed by a 1×1 Conv with no padding, which aggregates different channel information;
- **C3K2**: A Conv block followed by a variable-length chain of C3K. The output and input are finally concatenated before being processed by a 1×1 Conv block. This high-level block, made of Conv and Bottlenecks, aims to process and refine the structural features of the image;
- **Max Pooling**: a Max Pooling block is a variable-size kernel that uses a maximum operator instead of a linear regression. It serves mostly to rescale the size of an image and help make the network more robust and less noise-sensitive;
- **Spatial Pyramid Pooling Fast (SPPF)**: A 1×1 Conv block followed by a variable length chain of Max Pooling layers with a padding size that keeps the input size in output, such that the output of each Max Pooling can be concatenated and a final 1×1 block can process all the channels. Its purpose is to obtain representations at different scales and combine them all into a summarized representation.
- **Multi-Head Self Attention (MHSF)**: This block works on triplets: key K, query Q, and value V. Each pixel has its (k, q, v) such that (K, Q, V) are matrices where the first dimension is N, the number of pixels in the image. The number of triplets generated is defined as a "number of heads". The attention operation can be expressed as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V \tag{10}$$

  The following enumeration unpacks it step by step:

  1. An attention matrix is computed by performing the matrix pair-wise dot product KQ^T for all the possible combinations of pixel couples. The result is a $N \times N$ "attention matrix" A, where each cell represents how much "attention" shall be given to a specific pixel according to the global context and with respect to the entire row.
  2. Attention values are normalized according to the size of the Q and K vector size d_k by dividing over $\sqrt{d_K}$.
  3. A $softmax$ is applied row-wise to normalize all the row probabilities and have them correctly sum up to 1;
  4. Finally, the attention values are used as multiplicative weights by performing the matrix product between A and V to get the final feature values.

  This operation is performed for each head, which should conceptually focus on computing attention for different kinds of relations. Attention is deeply detailed in [27];
- **Partial Self Attention (PSA)**: After MHSF is performed, the output is concatenated with the input through a skip layer before being processed through a traditional dense NN layer. Then a 1×1 Conv processes the channels, and another 1×1 Conv linearly projects the output and concatenates it with the output of the first 1×1 block.
- **C2SPA**: A linear Conv projects the channels of each pixel into a larger dimensionality and splits them into multiple triplets of vectors: key k, query q, and value v (k and q must have the same size d_K). Then, a chain of 2 Attention blocks processes the input, and the output is eventually concatenated with the Attention input to be finally processed by a 1×1 Conv block. The purpose of this high-level block is to extract the importance of each pixel with respect to the entire image context.
- **Head**: this block uses a 1×1 Conv block to extract a BB in a $4 + N_{classes}$ long vectorial representation. The last $N_{classes}$ values represent the probability that the BB centered in a given pixel refers to an object of a certain class for each of the $N_{classes}$ classes. The first 4 features represents the (x_{rel}, y_{rel}) coordinates center of the BB in the image and both the width and height (w_{rel}, h_{rel}) of the image. All the values are normalized in $[0, 1]$ and relative to the actual width and height (w, h) of the image.

Figure 2 illustrates the entire architecture of YOLOv11 while Figure 3 provides the reader with a comprehensive view of all the building blocks. As anticipated, this entire processing is carried out once per picture. The returned possible BBs are eventually processed with non-maximum suppression and other determinstc pruning in order to get rid of useless guesses and merge the correct ones into perfectly fitting boxes.

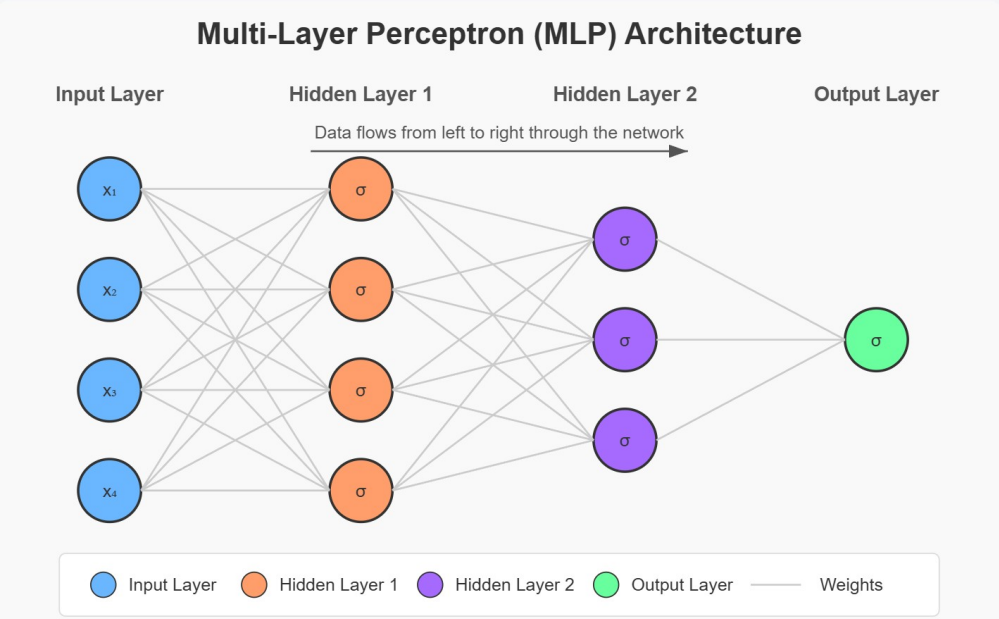

Figure 1: Multi-Layer Perceptron architecture.

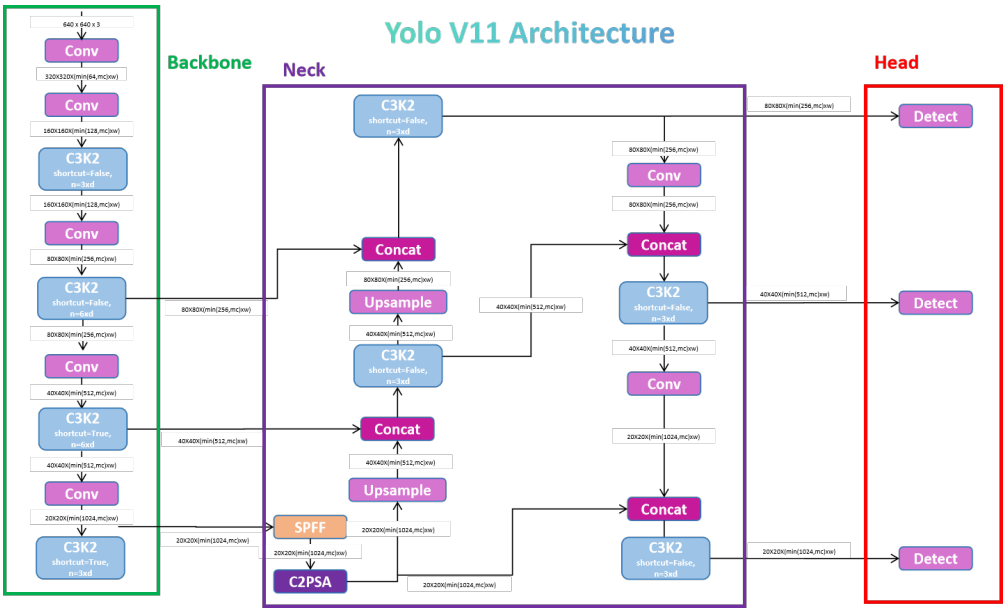

Figure 2: YOLOv11 architecture overview.

This powerful ML solution is released with various size and hyperparameters configurable to be tailored for specific use cases. Indeed, neuron weights have to be trained through a labeled dataset, where the correct BB have been manually annotated. It is also optimized with edge deployment in mind and can reach high levels of efficiency in terms of inference speed and memory occupation.

5 Vision Transformers

Vision Transformers (ViT) are one of the most recent innovation in CV object detection and image segmentation [18]. Once the attention mechanism introduced in Section 4 is understood, ViT has a much easier architecture based on a different paradigm. While YOLO is mostly based on Convolutional neural layers, which slide over the image and look for structural features just like Kernel filtering, ViT is entirely attention-based. The scope of this novel paradigm is identifying morphologies as correlations between images regions, no matter how sparse and far they are. To this aim, the following process summarizes how the algorithm works:

1. The image is spatially divided into cells, usually square-shaped, each of which includes a set of neighbouring pixels;
2. The pixels of one patch and their channel are all flattened into one feature vector;
3. The positional information that uniquely identifies the patch is embedded into the patch by adding one position encoding vector, which is usually a predefined sinusoidal pattern or a learned parameter;
4. Each patch feature vector is projected with multiple linear regression in a key, query, and value feature vector;
5. Attention is performed over keys, queries, and values to extract contextual features that take into consideration the global picture and the role that each pixel plays in the context and with respect to all the other;

Figure 4 effectively illustrates the architecture. ViT is natively a feature extraction NN. In order to obtain BB, regression heads are usually concatenated on top of the process to further process the output with NN. Some of these approaches use, for example, simple MLPs to complete the task. This architecture have been found to achieve better performance in terms of quality of the output thanks to its global-context-aware approach in contrast with the much more local convolutional filtering performed by YOLO [26]. However, attention is also well known for having a quadratic time complexity with respect to the input's spatial size, making it less suitable for edge applications. As described in Section 4, YOLO introduced attention blocks over time, and eventually, the CV field will converge to a flexible trade-off solution.

6 Artificial Vision in Space Missions

The use of AI models able to analyze and understand the content of traditional and multi-spectral images is widely studied for space missions. A YOLOv11-based method is presented in [30] to address the problem of in-orbit surveillance for debris and non-cooperative spacecrafts in close-proximity operations. YOLO-Dynamic is proposed in [29] for ground-based near-earth debris and asteroid detection. HB-YOLO was proposed in [28] for remote sensing, in particular, object detection and tracking from orbit. Similarly, YOLO-OSD, proposed in [21], addresses marittime surveillance mostly for commercial, highly technological platforms. In [19] uses and applications of YOLO for multispectral images are analyzed in general.

On the other side, also attention-based video has recently caught the attention of space research. The task of remote sensing, in particular image scene recognition, has been studied by [14], [24], and [15]. A review of the application of ViT for such tasks has been proposed in [16]. Deployment-oriented computational optimization of ViT models has been presented in [25] for in-orbit processing.

It is also finally worth citing TerraMind, presented in [22] as a generative multi-purpose model with the ambitious scope of building a multimodal remote sensing foundation model. Just like ViT, it is attention-based, and its processing follows the same working principles, but it is trained on a wide range of different tasks and a huge dataset of heterogeneous data.

Furthermore, industrial organizations and government space agencies are starting to use AI-based CV systems to enable cognitive processing. Some examples include ClearCharlie [12] developed by AIKO, "Detect & Classify" by a collaboration between ICEYE and SATIM, and Phifire AI, committed from ESA to Thales Alenia Space [13]. Hence, AI-based CV is rapidly becoming an application in the new space economies, from small and relatively new actors to well-established public and private organizations.

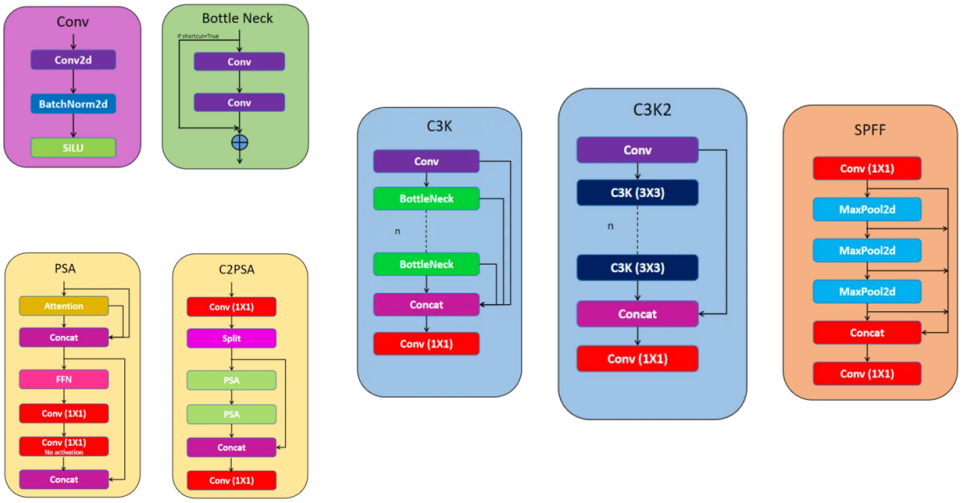

Figure 3: Building Blocks of YOLOv11.

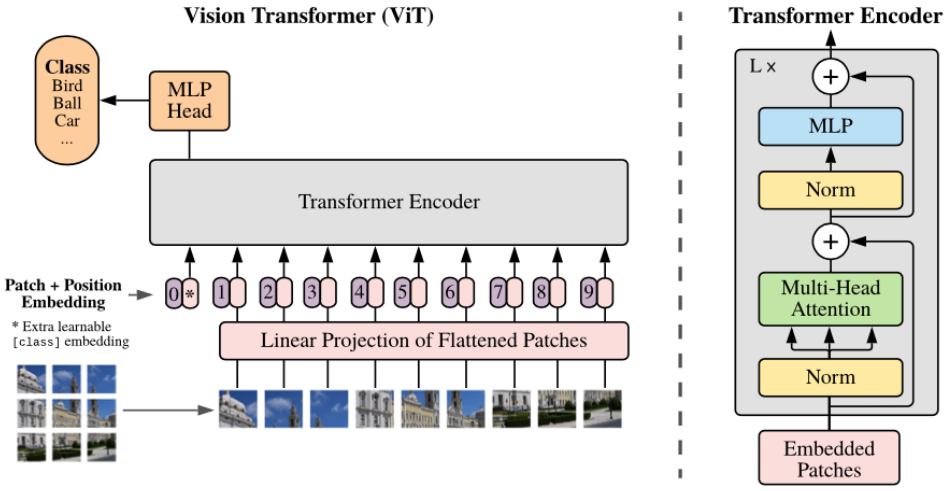

Figure 4: Vision Transformer architecture

7 Conclusion

In conclusion, CV moved gradually from traditional processing to neural processing, with a huge evolution of CNN into today's state-of-the-art solution like YOLO, and the discovery of novel approaches like ViT, which are now consolidated. Research is just starting addressing the usage of such models for space missions, even delving into optimization aspects, to support the concrete deployment of future technologies. At the same time, the rapid pace of the new space economy is starting to deploy the first commercial solutions for CV based on ML solutions.

References

[1] URL: https://www.ibm.com/think/topics/computer-vision.

[2] URL: https://www.ibm.com/think/topics/machine-learning.

[3] URL: https://www.ibm.com/think/topics/linear-regression.

[4] URL: https://www.ibm.com/think/topics/knn.

[5] URL: https://www.ibm.com/think/topics/deep-learning.

[6] URL: https://www.ibm.com/think/topics/neural-networks.

[7] URL: https://www.pinecone.io/learn/series/image-search/imagenet/.

[8] URL: https://www.ibm.com/think/topics/supervised-learning.

[9] URL: https://www.ibm.com/think/topics/self-supervised-learning.

[10] URL: https://www.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html.

[11] URL: https://www.ultralytics.com/glossary/non-maximum-suppression-nms#mechanisms-of-suppression.

[12] URL: https://aikospace.com/products/clear-charles.

[13] URL: https://www.esa.int/Space_in_Member_States/Italy/Phsat-2_ottiene_due_nuove_app_basate_sull_Intelligenza_Artificiale.

[14] Yakoub Bazi et al. "Vision transformers for remote sensing image classification". In: *Remote Sensing* 13.3 (2021), p. 516.

[15] Meiqiao Bi et al. "Vision transformer with contrastive learning for remote sensing image scene classification". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 16 (2022), pp. 738–749.

[16] Meiqiao Bi et al. "Vision transformer with contrastive learning for remote sensing image scene classification". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 16 (2022), pp. 738–749.

[17] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[18] Alexey Dosovitskiy. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[19] James E Gallagher and Edward J Oughton. "Surveying You Only Look Once (YOLO) multispectral object detection advancements, applications and challenges". In: *IEEE Access* (2025).

[20] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[21] Muhammad Farhan Humayun et al. "YOLO-OSD: Optimized ship detection and localization in multiresolution SAR satellite images using a hybrid data-model centric approach". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 17 (2024), pp. 5345–5363.

[22] Johannes Jakubik et al. "Terramind: Large-scale generative multimodality for earth observation". In: *arXiv preprint arXiv:2504.11171* (2025).

[23] Rahima Khanam and Muhammad Hussain. "Yolov11: An overview of the key architectural enhancements". In: *arXiv preprint arXiv:2410.17725* (2024).

[24] Pengyuan Lv et al. "SCViT: A spatial-channel feature preserving vision transformer for remote sensing image scene classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022), pp. 1–12.

[25] Yanhua Pang et al. "RepSViT: An efficient vision transformer based on spiking neural networks for object recognition in satellite on-orbit remote sensing images". In: *IEEE Transactions on Geoscience and Remote Sensing* 62 (2024), pp. 1–16.

[26] Pratik Patel. "YOLO vs Vision Transformers: A Comparative Review of Object Detection Techniques". In: *Internation Journal of Multidisciplinary Research* (2025).

[27] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[28] Chaoran Yu et al. “Hb-yolo: An improved yolov7 algorithm for dim-object tracking in satellite remote sensing videos”. In: *Remote Sensing* 15.14 (2023), p. 3551.

[29] Haiying Zhang, Zhengyang Li, and Chunyan Wang. “YOLO-Dynamic: A detection algorithm for spaceborne dynamic objects”. In: *Sensors* 24.23 (2024), p. 7684.

[30] Yi Zhou et al. “Improved Space Object Detection Based on YOLO11”. In: *Aerospace* 12.7 (2025), p. 568.