YOLO: You Only Look 10647 Times

Christian Limberg[1][a], Andrew Melnik[2], Helge Ritter[2] and Helmut Prendinger[1]
[1]*National Institute of Informatics (NII), Tokyo, Japan*
[2]*Bielefeld University, Bielefeld, Germany*

Keywords: Object Detection, Explainable AI/ML, YOLO, You Only Look Once.

Abstract: In this work, we explore the You Only Look Once (YOLO) single-stage object detection architecture and compare it to the simultaneous classification of 10647 fixed region proposals. We use two different approaches to demonstrate that each of YOLO's grid cells is attentive to a specific sub-region of previous layers. This finding makes YOLO's method comparable to local region proposals. Such insight reduces the conceptual gap between YOLO-like single-stage object detection models, R-CNN-like two-stage region proposal based models, and ResNet-like image classification models. For this work, we created interactive exploration tools for a better visual understanding of the YOLO information processing streams: https://limchr.github.io/yolo_visu

1 INTRODUCTION

Much progress in detecting multiple objects in an image using deep neural networks can be attributed to the introduction of R-CNN (Girshick et al., 2014), SSD (Liu et al., 2016), and YOLO (Redmon et al., 2016) architectures. R-CNN and its improved versions like FasterRCNN (Ren et al., 2016) detect objects by first producing proposals for regions containing an object (Melnik et al., 2021), and then in a second stage these proposals get passed through a classifier network. YOLO and SSD work without a separate proposal stage by combining object detection and classification into one stage. Four YOLO architecture successors along with other one-stage models were proposed. They introduce improvements such as the usage of anchor boxes, separate pathways for different object sizes, deeper architectures, different activation functions and a variety of other tricks and tweaks (Redmon and Farhadi, 2016), (Redmon and Farhadi, 2018), (Bochkovskiy et al., 2020).

We argue that the performance of these systems can be understood as performing classification and regression tasks for a high number of fixed region proposals with their positions relating to the convolutional grid. To this end, we implement several interactive visualizations, that (i) show the inner processing of the network and (ii) provide detailed insights how YOLO achieves both high speed and high accuracy.

In this paper we obtain insights of the inner mechanics of YOLO by modifying two distinct visualization approaches that were originally developed for examining classification CNNs, and are here adapted and applied for the YOLO object detection model. The first approach produces a saliency measure motivated by GradCam (Selvaraju et al., 2017). The second approach is motivated by Inceptionism and DeepDream (Mordvintsev et al., 2015) and optimizes the input pattern that is fed into YOLO for matching a specific target output response.

Both approaches use gradients for visualizing characteristics of the network. In GradCam, we feed an image with a target object into the network and multiply activation and gradient in a particular intermediate layer. On the other hand, in DeepDream we are searching for optimal input patterns for minimizing a certain cost function on the prediction. In other words, the input image itself is optimized in such a way that the related output neuron response will get closer to a desired target value.

The core findings of examining the modified versions of GradCam and DeepDream are:

- YOLO's high-confidence grid cells are sparse;
- each of YOLO's grid cells is attentive to a variable and localized sub-region of the input image;
- YOLO's attention is variable and directly related to the output neuron of interest;
- optimal input samples can be generated that optimize specific target outputs. This can be used for explaining the trained features responsible for detecting objects of e.g. a specific class, position or shape;

[a] https://orcid.org/0000-0002-4903-3933

Limberg, C., Melnik, A., Ritter, H. and Prendinger, H.
YOLO: You Only Look 10647 Times.
DOI: 10.5220/0011677300003417
In *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - Volume 5: VISAPP*, pages 153-160
ISBN: 978-989-758-634-7; ISSN: 2184-4321

- we can understand YOLO as performing parallelized classification and regression tasks of many image sub-regions. These regions efficiently share most of their computation, resulting from a high amount of overlap in CNNs.

By a simple post-processing step, only a very sparse selection of YOLO's output neurons are considered for the actual prediction, and the result of most output neurons is simply ignored. For YOLO.v4, the detection of a picture leads to a total number of 10647 classification/regression pairs, obtained by summing up the squared grid shapes times the number of anchor boxes $((13^2 + 26^2 + 52^2) * 3)$. This number is dependent on the image dimensions used for train YOLO.v4 which is 416x416 pixels. It has to be adapted for other image dimensions, since the dimensions of the grids would also change, if we assume the same network architecture.

This article is organized as follows: In Section 2 we explain the YOLO.v4 architecture and discuss its main principles. In Section 3 we introduce our first interactive visualizations of YOLO. Then we introduce the adapted GradCam in Section 3.1 and DeepDream approach in Section 3.2. Finally, in Section 4, we draw our conclusion and point out some analogies of YOLO's computational strategy to computational structures found in biological vision, where a good trade-off between speed and accuracy is tantamount as well.

2 YOLO ARCHITECTURE

In our experiments, we used a TensorFlow implementation[1] of YOLO.v4 (Bochkovskiy et al., 2020) that uses the original weights. Other YOLO implementations can be found here: YOLO.v1-v3[2] and YOLO.v5[3]. While there are newer versions of YOLO out, YOLO.v4 was an optimal choice based on performance, architecture clarity and code-availability. However, our experiments should be reproducible for following YOLO versions.

The network architecture is divided into two parts (see Fig. 1): First, the image is processed by a backbone network for feature extraction and second, the YOLO head is calculating object bounding boxes.

[1]https://github.com/hunglc007/tensorflow-yolov4-tflite
[2]https://pjreddie.com/darknet/yolo/
[3]https://github.com/ultralytics/yolov5

2.1 Backbone Network

In YOLO.v4, darknet53 is used as a backbone for feature extraction. The network assumes input rgb-images of size $(416 : 416)$. It consists of 23 residual blocks and 77 convolutional layers. Five of the convolutional layers downsample the input by applying strides of 2. The backbone's output feature map and 2 intermediate outputs (after the fourth and fifth downsampling steps) are then passed into the YOLO.v4 head.

2.2 YOLO Head

The YOLO head consists of 31 convolutions with a stride of 1 and padding for ensuring the same output size. Also, the three different paths from the backbone are concatenated in the YOLO head at different points with upsampling and downsampling operations (see Fig. 1).

The YOLO.v4 head has 3 different output *pathways* supporting the detection of smaller, medium and larger sized objects.

The first output from the YOLO head is the small objects pathway after 2 upsampling operations and 2 concatenations with all paths of the backbone network. After another downsampling and concatenation with skip-connection, the medium objects pathway is defined and after a last downsampling paired with a concatenation, the final output of the large objects pathway is defined.

2.3 Anchor Boxes

Anchor boxes are predefined bounding box patterns used by YOLO to delineate regions for object candidates. Each of YOLO's three pathways uses 3 different anchor box patterns (9 in total, see Fig. 1 right column). Each pathway further provides a grid of different resolutions (52x52x255 for small, 26x26x255 for medium, and 13x13x255 for large sized objects). Each grid cell is estimating the 3 respective anchor boxes. Each anchor box has 85 channels (summing up to 255 channels per grid cell). Five of the 85 channels (`x`,`y`,`w`,`h`,`c`) represent the x- and y-displacement of the center of the object's bounding box, the width and height of the bounding box and a confidence value denoting that the specific anchor box is detecting an object. **`p`** denotes the remaining 80 channels that are representing a probability value for each of the 80 classes for the COCO dataset.

The YOLO architecture can also be trained on a dataset with an arbitrary number of classes. Then **`p`** will denote the number of classes in the dataset. For

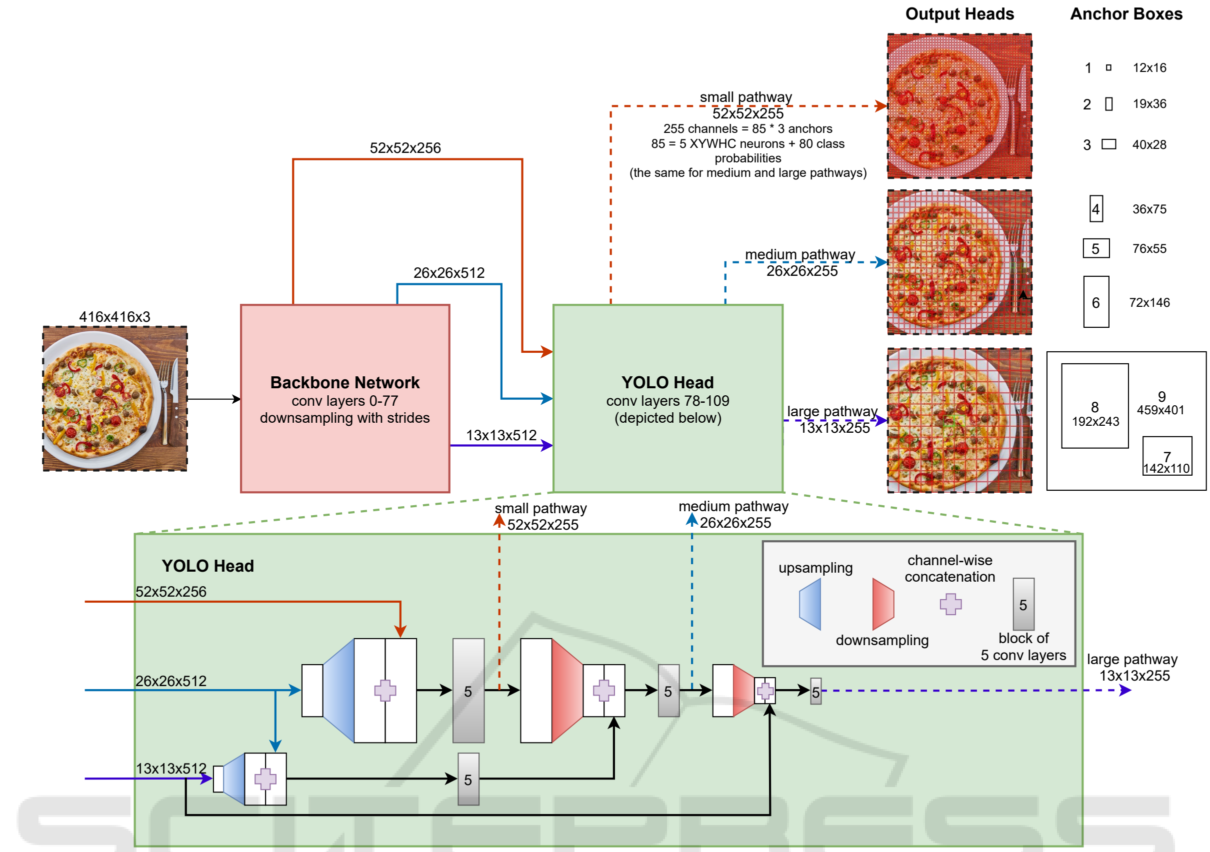

Figure 1: A simplified schematic of the YOLO.v4 network architecture.

example (Melnik et al., 2022) trained the YOLO architecture to detect faces and classify each detected face into 10 age classes (0-10 years old, 10-20, 20-30, 30-40, 40-50, 50-60, 60-70, 70-80, 80-90, 90-100).

For training, all bounding boxes from the COCO dataset were assigned to one (or several) of the YOLO anchor boxes, based on their Intersection over Union (IoU) value with the box patterns (see Section 2.4 for more details). Thus, after training, each of these 9 anchor boxes is trained to predict its own type of bounding box shape. Thus, the learnt anchor boxes work like templates for detecting objects of different sizes and shapes. There is e.g. an anchor box for detecting rather big, vertical-shaped objects and there is an anchor box for detecting smaller horizontal objects, etc. The YOLO architecture has 3 anchor boxes per grid cell, where the resolution of the grid changes with the 3 pathways. A confidence value near 0 indicates that no object was found located spatially within the grid cell. In Section 2.4 we further discuss how the training signal for the different anchor boxes, i.e. their representing neurons, is calculated.

2.4 Training

As explained in the previous section, the YOLO network is trained to map a 416x416x3 input array into three head output pathways (grids of shape 52x52x255, 26x26x255, and 13x13x255) to represent the input at appropriate discretization outputs for "small", "medium" and "large" objects, with each grid cell specifying three 85-dimensional channels for encoding 3 object bounding boxes. A grid cell can also opt not to represent its maximum of three boxes[4]: each channel that represents a box indicates this by setting its `c`-variable to 1, thereby assigning meaning for the remaining 4+80 `x,y,w,h` and **p** neurons. Otherwise, if `c`=0, these components are not to be interpreted as specifying anything.

With this representation convention, any dataset with images of annotated axis-aligned bounding boxes that represent the outline of objects can be straightforwardly translated into target values for

[4]Note that the maximum of three representable boxes per channel is not in any deep way related to the number of three pathways - it is perfectly thinkable to specify YOLO architectures where these numbers are chosen to differ.

channel components of the three output pathways. For each image and each bounding box label, the 3 differently sized grid cells at the spatial center of the object are identified. Each grid cell can be represented by 3 different anchor boxes. For training the network, the anchor boxes getting a positive training signal (described below) have to be identified.

The anchor boxes that have an IoU greater than 0.3 compared to the annotated object bounding boxes from the dataset are chosen to get a positive training signal. If there is no anchor box fulfilling this criteria, the anchor box with the largest IoU is selected to get the positive training signal. All positive anchor boxes are getting non-zero encoded values in the respective positions of the target vector used for training. They are getting a confidence value of 1, encoded values for `x`,`y`,`w`,`h` of the respective object bounding box, and a soft 1-hot-encoded vector representing the object class as **p**. All other fields in the target vector are set to 0's for "no object present".

With the so-defined target values the architecture can be straightforwardly trained to model the input-output relationship in the training data. This architecture can be compared to an ensemble of output paths, where each path is trained to detect the most appropriate bounding boxes (Bach et al., 2020).

3 VISUALIZATION

For a deeper inquiry into the nature of the pathway/grid cells representation that emerges under this training in YOLO, we implemented an interactive visualization of all pathways/grid cells for several images (see Fig. 2).

The number of detectable objects is limited to 10647. However, also in a quite busy image with many small objects covering the whole space (e.g. Fig. 2 image 2), this number isn't reached by far. Since only a few anchor boxes get a non-zero training signal, also the number of active (or high confident) anchor boxes in a prediction step is rather sparse. In fact, most of the anchor boxes are low-confidence and just "thrown away". The few anchor boxes that have a high confidence are post processed and build the final detection result.

Fig. 3 depicts the confidence mapping of a prediction. By shifting the input image a few pixels, one can see how the anchor boxes' confidences are also shifting and an anchor box of a neighboring grid cell "takes over".

We find that a maximum of 4 grid cells have a high confidence to detect a certain object, but most of the time only 1 or 2 grid cells are active. Fig. 2

Figure 2: With our interactive visualization, the full grid layers of the YOLO.v4 network can be depicted for several images. The YOLO architecture has 3 different pathways for recognizing objects of different sizes. The recognition heads are located in 2d-grids of different resolutions. Each grid element can detect underlying objects based of 3 possible anchor box shapes. Each anchor box refines estimates of the x- and y-position, the width and the height, a confidence value and a probability vector of each class used for training. The bounding boxes are labeled with the predicted class, the certainty value and an index of the displayed anchor box (we depict only the most confident anchor box out of the 3 possible). Object proposals with a high certainty are colored blue. The interactive version of this plot can be accessed via https://limchr.github.io/yolo_visu/index.html#fig2.

further shows that all of these active grid cells have fairly similar bounding boxes, so as a post-processing step an ordinary non-maximum suppression can be applied, choosing the bounding box with the highest confidence.

3.1 Saliency-Based Analysis of YOLO Layers

In this section we present an analysis of the responses of YOLO's output pathways in terms of their "saliency pattern" of an intermediate convolutional layer. To compute a suitable saliency measure, we build on the GradCam approach (Selvaraju et al., 2017), which calculates gradients based on an out-

Figure 3: Shifting the actual input image below the grid cells makes apparent how the confidences of the anchor boxes (blue means high confidence) are shifted and neighboring grid cells get activated. The interactive version of this plot can be accessed via https://limchr.github.io/yolo_visu/index.html#fig3.

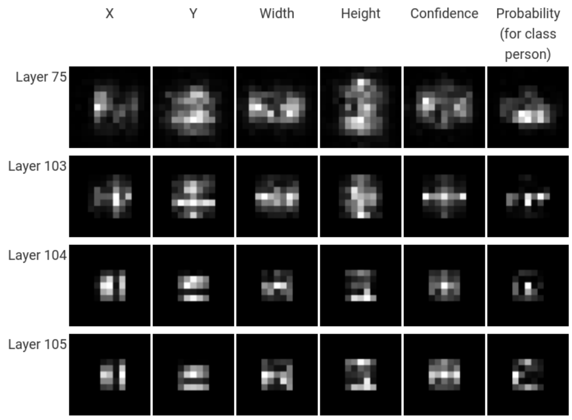

Figure 4: Our adapted Detection GradCam visualizes the saliency map of a single output neuron. The columns represent the saliency maps of the x-shift, y-shift, width, height, confidence and probability neuron of a selected grid cells of the large pathway (13x13 grid). As rows, we depict saliency maps for convolutional layers 75, 103, 104 and 105. Each plot represents the saliency map averaged over 15 images (15x13x13) having class "person" under the corresponding grid cell's position. The interactive version of this plot can be accessed via https://limchr.github.io/yolo_visu/index.html#fig4.

put neuron under consideration (e.g., typically a class-neuron of the one-hot encoded output layer of a classification CNN).

To produce a scalar that indicates the importance of a particular feature channel, the original algorithm first averages the gradients at the (usually last) convolutional layer over the spatial dimensions of the layer's channels. The scalar is then multiplied with the actual activation values of neurons. The result is an importance-weighted saliency map. The spatial pattern of these values across the convolutional layer and thus the corresponding input image shows which input region is accountable for the response on the selected output neuron.

For applying this technique to YOLO, we have to make several changes. We can not use the very last convolutional layer because of the different architecture of the network (the last convolutional layers don't have that much saliency information because of the grid outputs). Therefore, we use an intermediate layer for computing saliency maps.

We found out that we can generate much more descriptive saliency maps by multiplying the output of the intermediate layer element-wise with the plain gradients of this layer (e.g. calculated from the `w`-neuron or the `h`-neuron of one specific grid cell/anchor box) and average the output channels. But still, the so-obtained result is pretty noisy. We did another trick for getting a clearer pattern: We query several images from the COCO dataset that have an instance of a particular class (e.g. "person") located at a particular spatial image position, i.e. where the underlying YOLO grid cell would get a positive training signal. We pass 15 of these images through our adapted GradCam algorithm and average the result images for getting a saliency map for this grid cell. By repeating the process for all grid cells (omitting border grid cells since in the dataset are too few samples having persons located in the border areas), we get a clearer visualization and we can see how the saliency of YOLO will shift as we hover over the image (see Fig. 4).

The figure shows that the saliency is focused below the corresponding grid cell's position. Further, it depicts that the saliency map of the `w`-neuron and `x`-neuron has a rather wider activation, while the `h`-neuron's and `y`-neuron's saliency map has a rather vertical activation, i.e. the detection is more sensitive to these areas. The activation of the `p`-neuron, which is representing the class "human" in **p**) and especially the `c`-neuron is more focused to the center.

3.2 Optimization-Based Analysis of YOLO Input Patterns

Back in 2014/2015, the Inceptionism and DeepDream approaches (Mordvintsev et al., 2015) showed that input images can be optimized for maximizing a particular intermediate or output neuron of the network. Those input images then show patterns that the respective neurons are responsive to (Olah et al., 2017). The optimization process is achieved by forward-passing an image into the network and back-propagating the gradients back into the input layer. The input layer, or rather say, the subsequently changing input image, is then modified by adding the normalized gradient multiplied by a optimization rate. The weights of the neural network, however, are frozen and do not change.

The main difference regarding detection models is that we usually want to optimize for multiple output neurons, since we can not only estimate the class of the object but also its shape and position, which relates to additional regression problems. In other words, we not only want to maximize the stimulus of a particular neuron of a one-hot encoded output layer by gradient ascent, we are using a gradient descent approach for optimizing several of YOLO's output neurons to be a specific target value (e.g. the object's height should be 200 pixels). Doing this, we can require the target object to have a specific size and position in the image simultaneously with optimizing for specific features/classes.

In our experiments, we are starting the optimization process with a grey image. The gradients are calculated by considering all neurons of a specific grid cell/anchor box to be a pre-defined target value. We set the confidence neuron `c` and the target class' `p`-neuron of this anchor box to be optimized to be 1, all other neurons in $\mathbf{p}$ are optimized to be 0. The `x`,`y`,`w`,`h`-neurons can be set to be optimized to a specific target value.

The target matrices have the shape of the 3 output pathways and gradients are computed as the difference of target matrix and actual output values. ({13x13,26x26,52x52}x3x85). Other than directly back-propagating these gradients through the network, they are first multiplied by a *multiplier mask* that is weighting each output neuron. If we set the a multiplier in that mask to a high value, the gradient of the respective output neuron would be more influential to the optimization process. A value of 0 in the multiplier mask would neglect the respective neuron. We thereby give the `c`- and `p`-neuron of the positive anchor box a weight of 10 each - making them more influential regarding the update process. In our experiments we set the target confidences of all other anchor boxes to 0 with a small multiplier. If we set the multiplier of all other anchor boxes to 0, i.e. ignoring them for calculating the gradient, the process would converge faster to the target class but may also include other classes at random positions as well. As loss function we use a weighted sum of the L_1-loss loss and the total variation score for reducing noise.

Figure 5: Video demonstrating the optimization process of several classes of the COCO data set by our proposed "Deep Detection Dream" approach. The video can be accessed via https://limchr.github.io/yolo_visu/index.html#fig5.

In classification networks for reducing noise often shift and/or rotation operations are applied to the input image (Olah et al., 2017). However, we found out that in detection networks this will affect the detection position so that the optimization process is not converging well. Using the described method "Deep Detection Dream" we created optimized input images for several classes of the COCO data set used to train YOLO. In Fig. 5 several of these optimization processes are visualized for the center grid cell of the large object pathway. The object's target width and height are 200 pixels each, this is 41% of the image width/height.

Next, the capabilities to generate objects at a certain image position are illustrated. In Fig. 6 we are optimizing an object for every grid cell. The result is strongly resembling Fig. 4 and supports our claim that each grid cell is attentive to its underlying image area.

Not only the object class and position can be optimized towards a desired value but also the object's dimensions, i.e. its width and height. This can be explored by another interactive visualization in Fig. 7.

It can be seen that a non-matching configuration of output head and `w`,`h` target values is affecting the result quality, i.e. generating small objects with the large object head does not result in a well recogniz-

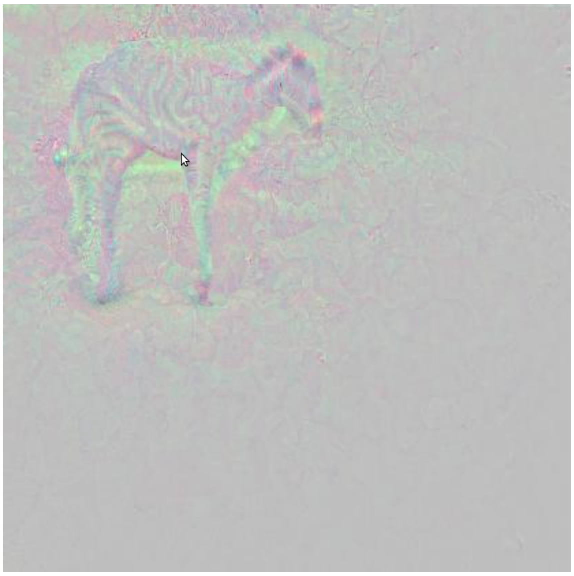

Figure 6: By our proposed "Deep Detection Dream" approach, we can generate objects in specific spatial image regions. The static print figure can only show for a single position specification the associated sensitivity region. The interactive version, accessible via https://limchr.github.io/yolo_visu/index.html#fig6, allows to explore the sensitivity region as a function of an interactively chosen position.

Figure 7: By our proposed detection dream approach, we can generate objects with specific attributes like in this case an adjustable height. The interactive version of this plot can be accessed via https://limchr.github.io/yolo_visu/index.html#fig7.

able images. Other experiments also showed, that if the samples of a particular object class are usually small in the images of the training data base, those classes cannot reconstructed well with the big anchor boxes neither.

However, Fig. 7 extends our statement to the effect that the attentive image region is variable in size and also shape.

4 CONCLUSION

The visualization of saliency patterns and optimized input images of YOLO reveal that YOLO acts like parallelized classification CNNs: each anchor box's saliency is pointed to an underlying subarea of the image and on this subarea classification and regression tasks are focused. These tasks are locally interdependent (the dimensions and location of a box are determined by its contents and vice versa). However, this tight spatial coupling focuses the propagation of the gradients and their interactions, creating a strong "spatial hierarchy"-bias that makes learning and subsequent processing very efficient.

This is not dissimilar to the information processing in human visual cortex (Sheth and Young, 2016; Melnik et al., 2018). In the primary visual cortex V1, features are extracted and passed to secondary visual cortex V2, where the information is split into a dorsal and ventral stream for localizing (dorsal) and classifying (ventral) objects. The dorsal stream is more explorative, showing a wider activation in the biological saliency map for localizing objects and movements in the scenery, where the ventral stream's activation focuses more on the center of the object (see (Sheth and Young, 2016) Fig. 2). We can see similar properties also in Fig. 4, when comparing columns `x`,`y`,`w`,`h` with `c`,**`p`**: `x`,`y` define the relative position of the detected object and `w`,`h` represent the dimensions of the object. I.e. the four neurons estimate where the object is, while `c` and **`p`** determine if there is an object present, and the object's class (what is it?). The saliency map activations of `x`,`y`,`w`,`h` is rather wide, focusing on the object borders (`x`,`w` for horizontal and `y`,`h` for vertical border areas) for determining where exactly it is, and the activation of `c`,`p` is more narrow and focused to the center, comparable to the dorsal and ventral stream of the biological model.

The nature of an artificial neural network, and especially a CNN, is that it consists of many parallel operations for one layer. This relates to a simple matrix multiplication. GPUs are built for this purpose and modern GPUs can do many matrix calculations in a very short amount of time. So it seems natural to exploit this feature and just "throw away the uninteresting results", i.e. the low confidence detections.

Compared to early 2-stage detectors, which had intermediate steps for selecting, e.g. region proposals which were then fed into a second neural network, the advantage is a massive speed increase (and only a minor loss in accuracy) since this intermediate steps take time since they are running on the CPU.

In this article, we demonstrated and visualized the inner mechanics of the YOLO architecture. Our key message is that YOLO is not really “looking once”, but a lot more often. Because of a clever exploitation of Artificial Neural Network structures, which make it possible to share most of the computation between regions and also allow to easily parallelize the computations on a GPU, this can be very fast and efficient.

Our findings might be used for future developments in architecture design or for evaluating trained models. Interesting future work include the improvement of these visualization techniques. Also, the proposed detection dream approach might be used to determine how much information about a training image is actually saved “within the weights of the network” by setting the target output to the actual prediction output of the model and optimize from a gray image. Also, different other constraints can be added to the optimization loss. By including the distance to a color histogram into the loss function, the reconstructed images might can be improved to have a more realistic color distribution.

ACKNOWLEDGEMENTS

This work was supported by a fellowship within the IFI program of the German Academic Exchange Service (DAAD).

REFERENCES

Bach, N., Melnik, A., Rosetto, F., and Ritter, H. (2020). An error-based addressing architecture for dynamic model learning. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 617–630. Springer.

Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.

Melnik, A., Akbulut, E., Sheikh, J., Loos, K., Buettner, M., and Lenze, T. (2022). Faces: Ai blitz xiii solutions. *arXiv preprint arXiv:2204.01081*.

Melnik, A., Harter, A., Limberg, C., Rana, K., Sünderhauf, N., and Ritter, H. (2021). Critic guided segmentation of rewarding objects in first-person views. In *German Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 338–348. Springer.

Melnik, A., Schüler, F., Rothkopf, C. A., and König, P. (2018). The world as an external memory: the price of saccades in a sensorimotor task. *Frontiers in behavioral neuroscience*, 12:253.

Mordvintsev, A., Olah, C., and Tyka, M. (2015). Inceptionism: Going deeper into neural networks.

Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*. https://distill.pub/2017/feature-visualization.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Redmon, J. and Farhadi, A. (2016). Yolo9000: Better, faster, stronger.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement.

Ren, S., He, K., Girshick, R., and Sun, J. (2016). Faster r-cnn: Towards real-time object detection with region proposal networks. In *arXiv*.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626.

Sheth, B. R. and Young, R. (2016). Two visual pathways in primates based on sampling of space: exploitation and exploration of visual information. *Frontiers in integrative neuroscience*, 10:37.