Risiko! Object Detection: BranchyYOLO

Michele Sprocatti[1], **Alberto Pasqualetto**[2], **Lorenzo Serafini**[3]

Department of Information Engineering, University of Padua
Via Gradenigo 6, 35131 Padova, Italy
{michele.sprocatti[1], alberto.pasqualetto.2[2], lorenzo.serafini.1[3]}@studenti.unipd.it

Abstract

In this paper we present *Branchy Yolo*: a new architecture for object detection applied to "Risiko!" board game. This architecture is a deep learning model developed on the same ground ideas of YOLO (Redmon et al. 2015) but with the intention of reducing the number of parameters to speed up the computation at inference time in order to use the model in low-end devices. During the paper, we also present an ablation of YOLOv9-C (Wang and Liao 2024) and we compare the performances of the two models. We also present a possible real application of BranchyYOLO and how to improve it to have better performance.

Introduction

The main idea is to create a Deep Neural Network that can perform object detection of "Risiko!" pieces with fewer parameters, while maintaining a good accuracy compared to YOLOv9 (Wang and Liao 2024) in order to be used on the go with a mobile app.

Risiko! game

"Risiko!" is the Italian variant of the strategic board game known internationally as "Risk": it is set in a world war environment and the aim is to conquer the world by defeating the other players or to reach some objectives.

The game is played on a board divided into territories, each of which is represented by a polygonal region. All players have a set of pieces: tanks and flags (which represent 10 tanks); armies are placed on the board and represent the player's power in a territory. Each player has a color (yellow, red, green, blue, purple, and black) and the pieces are colored accordingly.

It is important to know how many tanks and flags are in a territory, so we need to be able to count them. The task of counting the pieces in each territory can be performed by a deep learning model trained for object detection.

Method

YOLO

The most popular object detection models are based on Convolutional Neural Networks (CNNs) and in particular on the YOLO architecture (Redmon et al. 2015). The YOLO architecture is a one-stage detector that makes predictions in a single forward pass of the network and combines the different outputs of features extracted to make the prediction. The latest version of YOLO at the time of this project's development is YOLOv9 (Wang and Liao 2024); on 23 May 2024, YOLOv10 was released (Wang et al. 2024).

Dataset

The train dataset's images are generated by randomly rotating and scaling 3D models of tank and flag pieces and then superimposing the result multiple times over a set of background images. Each piece is also colored with one of the six colors of the game. The main background dataset is MSR-CORID (Microsoft Research Cambridge Object Recognition Image Database) (Microsoft Research Cambridge 2005), which contains images of natural scenes with objects in them. We also added some specific backgrounds like tablecloths with flowers and the Risiko! board to enhance the similarity between the synthetic images and the real ones since we observed that the model often confused the pieces with tablecloths' motifs.

To reflect the uneven distribution of the pieces in the game, we generated the dataset with a higher number of tanks than flags, with a ratio of about 2:1 as shown in figure 1.

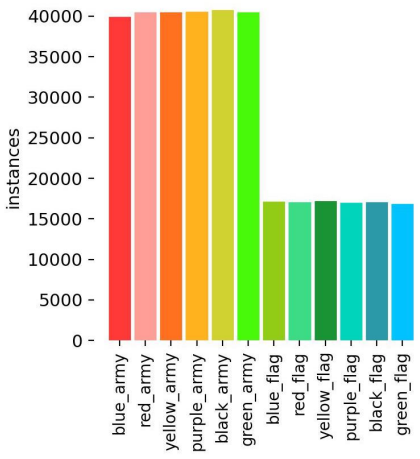


Figure 1: Labels distribution across the synthetic dataset

In order to train the deep neural network a lot of data is needed but, due to storage limits, we only were able to manage about 10,000 images.

We also used a dataset consisting of real images: this dataset is composed of 40 images taken from the game board, with the pieces annotated by hand.

The dataset was split into three parts: training, validation, and test, as shown in table 1.

	Train	Validation	Test
Synthetic	7,000 (70%)	1,500 (15%)	1,500 (15%)
Real	28 (70%)	0 (0%)	12 (30%)

Table 1: Dataset split

Hardware and Software

The model was trained on systems with A40 GPU (48 GB of VRAM) (Department of Information Engineering cluster @ UniPD).

The test phase has been done on systems with T4 GPU (16 GB of VRAM) (Google Colab).

The used software is YOLOv9 PyTorch official implementation (WongKinYiu 2024).

Ablation study

First of all, we investigated a possible ablation of YOLOv9-C (Wang and Liao 2024). The ablation starts from the idea of reducing the number of parameters of the network while maintaining good performances on Risiko! pieces detection. The idea which drove the ablation is to focus on the detection of small and medium size objects since the pieces of the game have small and medium size.

Model changes

After studying the architecture of YOLOv9-C (Wang and Liao 2024), we decided to simplify the model by reducing the number of parameters specifically in the parts that are responsible for detecting large objects both in the neck and in the auxiliary part of the model.

The result of this ablation can be seen in figure 7. This light version of YOLOv9-C uses 41 million parameters so, compared to 51 million of the original YOLOv9-C, we achieved a 20% reduction.

Training

We trained this model for 300 epochs on the synthetic dataset only and then we evaluated the performance on the real images, but also on the test split of the synthetic. Since we have a lot of parameters we are in danger of overfitting but, looking at the validation performances during training, we can say that the model does not overfit because its metrics are monotonically increasing as can be seen in figure 2.

BranchyYOLO

In order to reduce the number of parameters and speed up the computation at inference time, we developed a neural network with the same idea of YOLOv9.

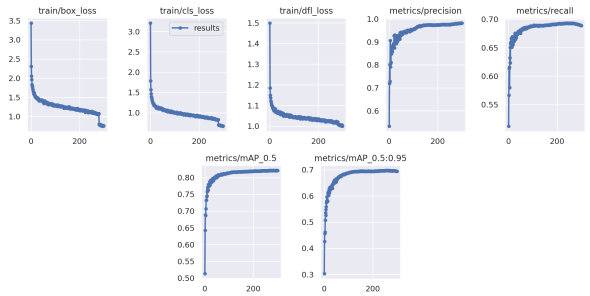


Figure 2: Results of training of the ablated model on the synthetic dataset only

The architecture is based on the idea of having a backbone with 2 branches, each one can learn a different representation of features:

Left Branch: small convolution kernel and average pooling

Right Branch: big convolution and max pooling

The two features are then concatenated and passed to an `SPPELAN`.

The neck consists of a series of upsampling and convolutional layers, more precisely `RepNCSPELAN4`s.

Then we make the final prediction. We selected the LeakyReLU as activation function with parameter 0.1 since it solves the problem of vanishing gradient and its gradient is computed faster than SiLU that is used in the original YOLOv9-C.

We still apply the promising Feature Pyramid concept to combine multi-scale feature maps by providing as output of the backbone also two intermediate results, with higher resolution. These ones are concatenated at the end of the neck and then they are used to do the final predictions.

The full model architecture can be seen in figure 8.

We strove to keep the complexity of the model as small as possible and we achieved a reduction from 51 million (YOLOv9-C) to 17 million parameters, a 67% reduction.

Model	Parameters
BranchyYOLO	17 million
Ablated YOLOv9-C	41 million
Yolov9-c	51 million

Table 2: Comparison between the number of parameters in the different models

Results

The training consisted of 300 epochs with a batch size of 20.

The training has been performed twice to study the performance of the model in two different scenarios:

1. Train validation and test on the synthetic dataset only
2. Train validation and test on both synthetic and real datasets

Training and validation metrics show that the model can detect very well synthetic images without overfitting them

as can be seen in figures 3 and 4 since the loss related to the validation set is decreasing during the entire training (validation was performed after each epoch).

In fact, the model behaves very well on the test datasets as can be seen in the confusion matrices in figures 6a and 6c.

But we can observe that the model is not able to generalize well on real images, as shown in the confusion matrices in figures 6b and 6d. This is because the model has been trained only on synthetic images and the real images differ too much from the former ones.

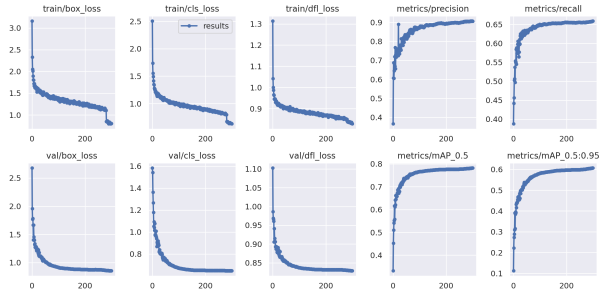


Figure 3: Results of training on the synthetic dataset only

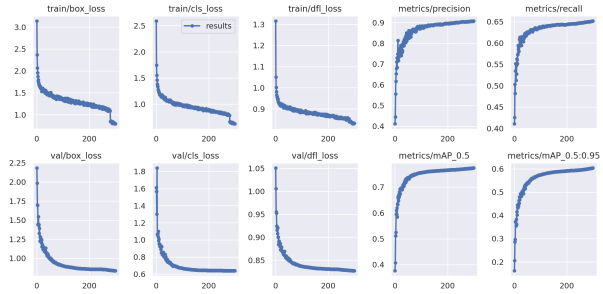


Figure 4: Results of training on both synthetic and real datasets

While a small performance increase can be noticed between tests on the two scenarios, the real images dataset is too small (only 40 images) compared to the synthetic dataset (10,000 images) to make a significant difference.

The model would perform better if trained on a larger dataset even if the new images are all generated, this is motivated by the table 3 where you can see the performance improvement passing from 1000 to 10000.

	Synthetic		Real	
Size	mAP50	mAP50-95	mAP50	mAP50-95
1k	0.696	0.469	0.106	0.042
10k	0.787	0.623	0.163	0.075

Table 3: Comparison between the metrics of the model trained on 10,000 images and on 1,000 images

In all the confusion matrices (figures 6a, 6b, 6c, and 6d) we can see that the model performs better when it has to recognize tanks compared to when it has to detect flags, this is probably related to the fact that in our dataset we have a very large number of tanks compared to the number of flags as presented in section Dataset; another culprit can be identified in the bounding boxes of the flags, which contain more background than the bounding boxes of the tanks so they are more difficult to be discerned from the background.

Also, we observe that the model creates some confusion between tanks and flags of the same color.

Comparison with ablated model

Since the ablated model described in section Ablation study has more parameters than BranchyYOLO, it can generalize better onto the real images, so we obtain better performances, but in the test split of the synthetic dataset we obtain similar performances. The performance metrics about all the classes in our dataset are reported in table 4 and table 5.

	Synthetic dataset			
Model	P	R	mAP50	mAP50-95
Ablated YOLO	0.976	0.68	0.81	0.686
BranchyYOLO	0.911	0.654	0.787	0.623

Table 4: Comparison between metrics of ablated YOLO and BranchyYOLO onto the synthetic dataset

	Real dataset			
Model	P	R	mAP50	mAP50-95
Ablated YOLO	0.395	0.186	0.236	0.118
BranchyYOLO	0.37	0.128	0.163	0.075

Table 5: Comparison between metrics of ablated YOLO and BranchyYOLO onto the real dataset

The confusion matrices relative to this train are presented below (figure 6e and figure 6f).

Comparison inference times

Table 6 displays the comparison of the inference times for the three different models. BranchyYOLO has an inference time that is halved compared to YOLOv9-C, but it is also smaller compared to the ablated YOLOv9-C: this comes from the fact that the model has much fewer parameters.

Model	Pre-process	Inference	NMS[1]
BranchyYOLO	0.6ms	12.9ms	7.9ms
Ablated YOLOv9-C	0.1ms	19.8ms	2.3ms
YOLOv9-C	0.2ms	27.7ms	10.7ms

Table 6: Comparison of inference time

Future works

The authors think that BranchyYOLO can be further improved by using a larger dataset for training. This dataset

[1]Non-Maximum Suppression

must contain more ”real images” in order to achieve better performance and also the number of flags must be incremented in contrast to the idea of having more tanks than flags proposed in section Dataset.

A possible application of this model can be a simple mobile app that, given the camera input, allows the user to select the two regions in a match, roll the dice, and tell which region wins the battles; this can be done by detecting flags and the tanks inside the two selected regions. An example of the app's mockup can be seen in figure 5.

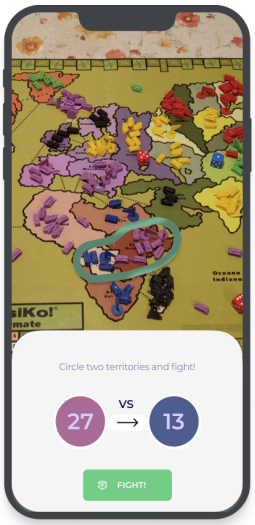


Figure 5: App Mockup

Conclusions

In this paper, we presented two possible architectures for object detection applied to the Risiko! game. The first one is an ablation of YOLOv9-C which behaved well compared to the original model, the second one is a new architecture called BranchyYOLO: a very small architecture, with a new concept. The latter is a deep learning model developed starting from YOLO architecture trying to reduce the number of parameters in order to speed up the computation at inference time. This new model implements a backbone with two branches that should learn different representations of features to try to enhance the performance of the model.

Also, we presented a possible real application of BranchyYOLO and how to improve it to have better performance.

The aim of speeding up the inference is facing the problem of having a mobile app that can be used on the go to play the game on low-end devices. We achieved the speed up of the computation (almost halved) due to the strong reduction of the number of parameters (67% less).

This model, together with an implementation, can be used by people with color blindness to help them play the game. Some color-blind people were interested in the project when we presented it to them.

References

Chang, H.-S.; Wang, C.-Y.; Wang, R. R.; Chou, G.; and Liao, H.-Y. M. 2023. YOLOR-Based Multi-Task Learning. *arXiv preprint arXiv:2309.16921*.

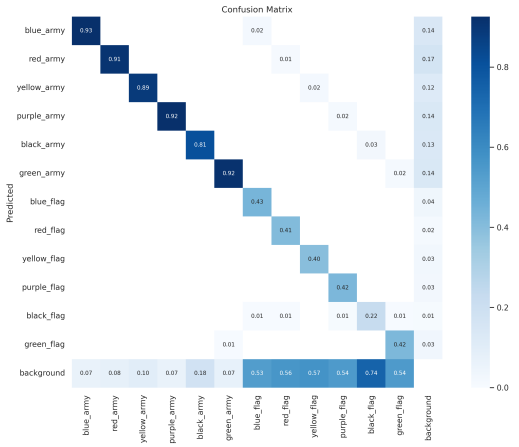
Microsoft Research Cambridge. 2005. Microsoft Research Cambridge Object Recognition Image Database (MSCORID). https://www.microsoft.com/en-us/download/details.aspx?id=52644. Accessed: 08/06/2024.

Redmon, J.; Divvala, S. K.; Girshick, R. B.; and Farhadi, A. 2015. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, abs/1506.02640.

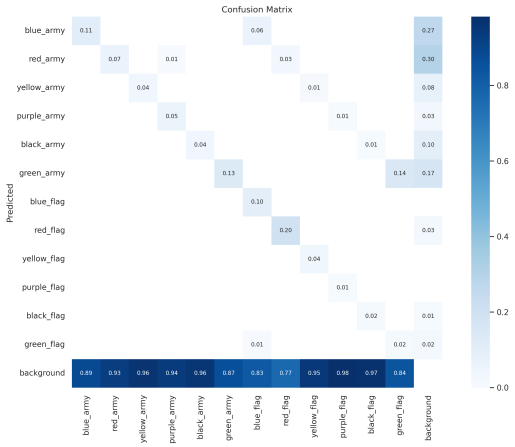
Wang, A.; Chen, H.; Liu, L.; Chen, K.; Lin, Z.; Han, J.; and Ding, G. 2024. YOLOv10: Real-Time End-to-End Object Detection. arXiv:2405.14458.

Wang, C.-Y.; and Liao, H.-Y. M. 2024. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information.

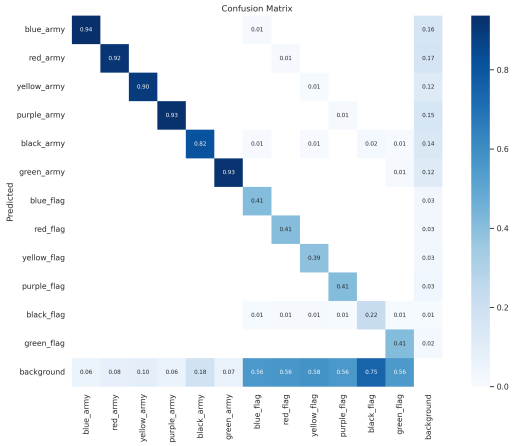
WongKinYiu. 2024. Implementation of paper - YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. https://github.com/WongKinYiu/yolov9. Accessed: 15/05/2024.



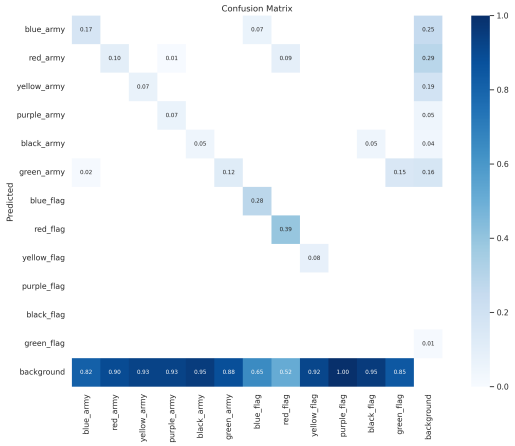
(a) Confusion matrix of the test on synthetic test dataset only; first approach



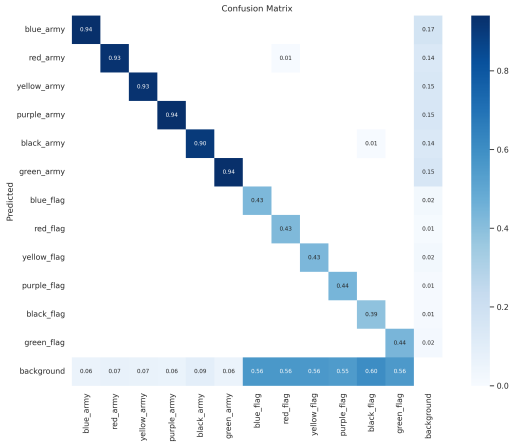
(b) Confusion matrix of the test on real test dataset only; first approach



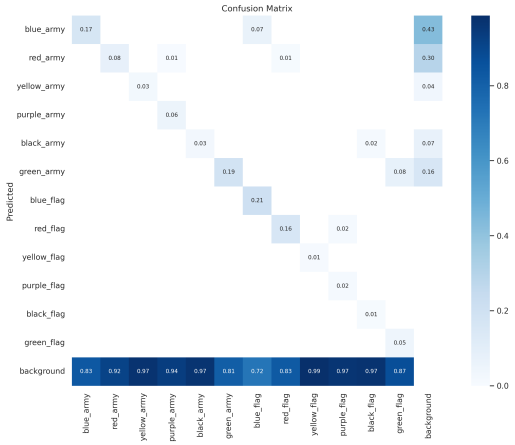
(c) Confusion matrix of the test on synthetic test dataset only; second approach



(d) Confusion matrix of the test on real test dataset only; second approach



(e) Confusion matrix of the test on synthetic test dataset only; ablated YOLO



(f) Confusion matrix of the test on real test dataset only; ablated YOLO

Figure 6: Confusion matrices

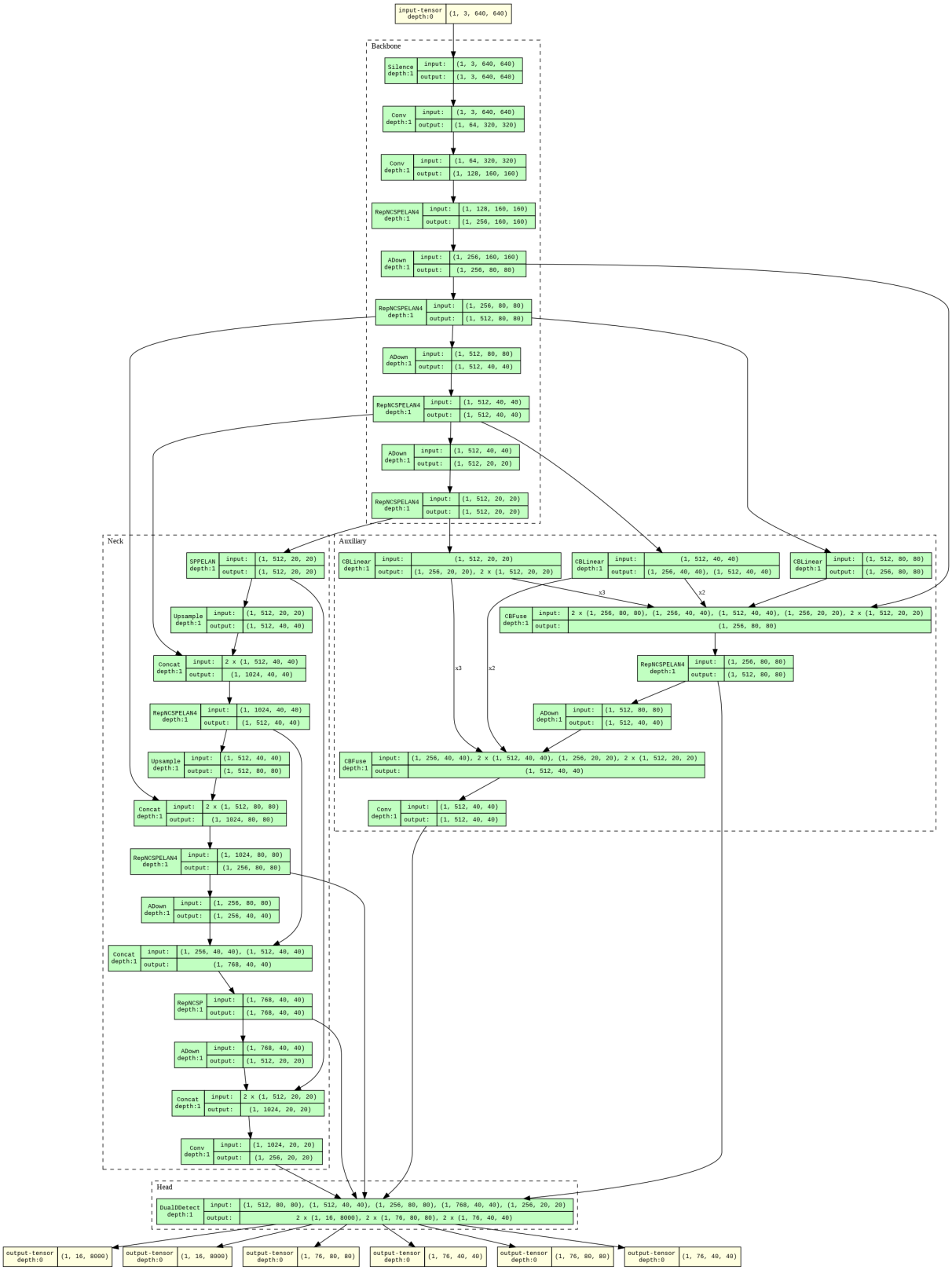


Figure 7: Model Architecture: Ablated YOLOv9-C

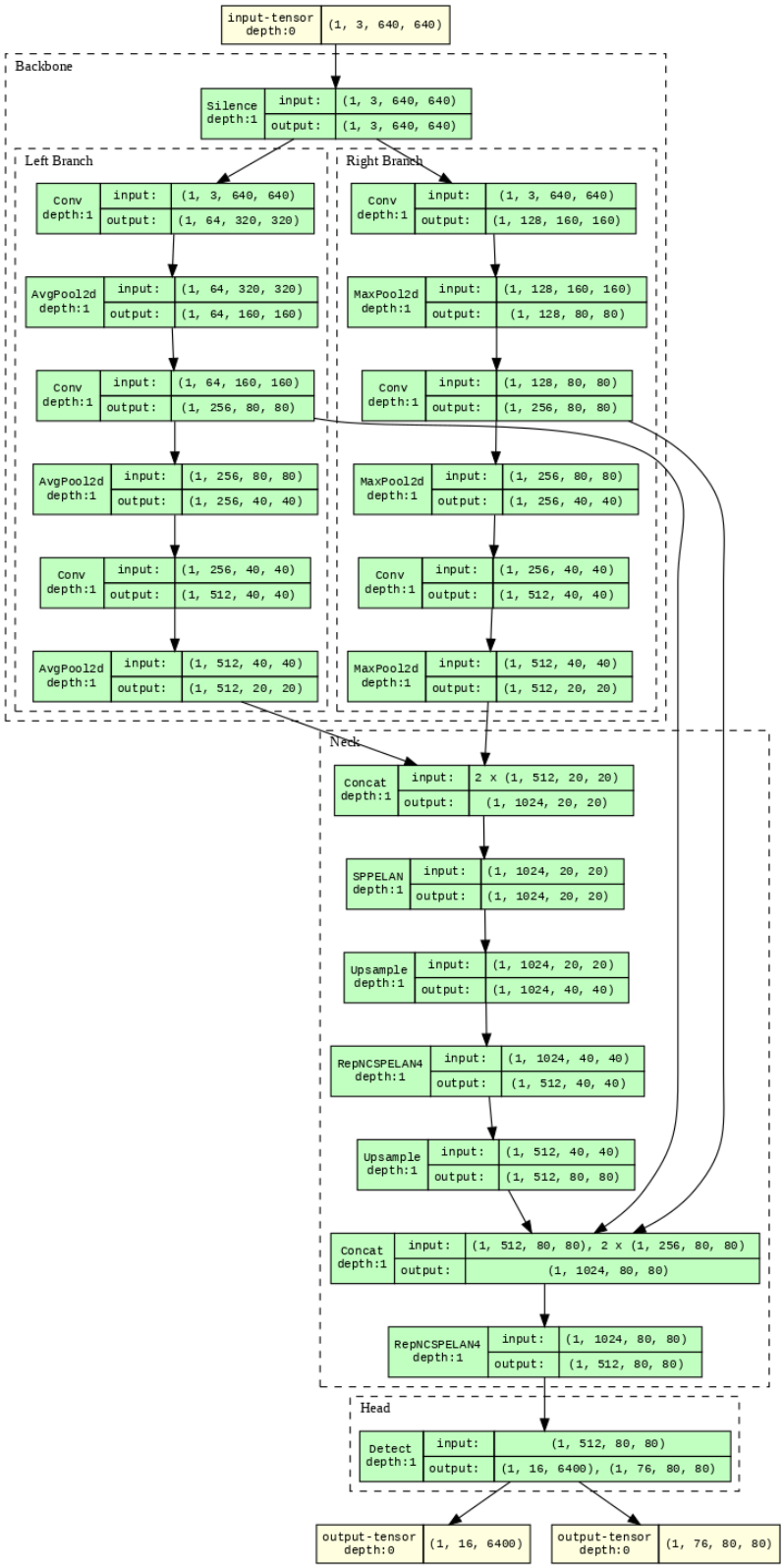


Figure 8: Model Architecture: BranchyYOLO