You Only Learn One Representation: Unified Network for Multiple Tasks

Chien-Yao Wang[1], I-Hau Yeh[2], and Hong-Yuan Mark Liao[1]
[1]Institute of Information Science, Academia Sinica, Taiwan
[2]Elan Microelectronics Corporation, Taiwan
kinyiu@iis.sinica.edu.tw, ihyeh@emc.com.tw, and liao@iis.sinica.edu.tw

Abstract

People "understand" the world via vision, hearing, tactile, and also the past experience. Human experience can be learned through normal learning (we call it explicit knowledge), or subconsciously (we call it implicit knowledge). These experiences learned through normal learning or subconsciously will be encoded and stored in the brain. Using these abundant experience as a huge database, human beings can effectively process data, even they were unseen beforehand. In this paper, we propose a unified network to encode implicit knowledge and explicit knowledge together, just like the human brain can learn knowledge from normal learning as well as subconsciousness learning. The unified network can generate a unified representation to simultaneously serve various tasks. We can perform kernel space alignment, prediction refinement, and multi-task learning in a convolutional neural network. The results demonstrate that when implicit knowledge is introduced into the neural network, it benefits the performance of all tasks. We further analyze the implicit representation learnt from the proposed unified network, and it shows great capability on catching the physical meaning of different tasks. The source code of this work is at : https://github.com/WongKinYiu/yolor.

1. Introduction

As shown in Figure 1, humans can analyze the same piece of data from various angles. However, a trained convolutional neural network (CNN) model can usually only fulfill a single objective. Generally speaking, the features that can be extracted from a trained CNN are usually poorly adaptable to other types of problems. The main cause for the above problem is that we only extract features from neurons, and implicit knowledge, which is abundant in CNN, is not used. When the real human brain is operating, the aforementioned implicit knowledge can effectively assist the brain to perform various tasks.

Implicit knowledge refers to the knowledge learned in a subconscious state. However, there is no systematic def-
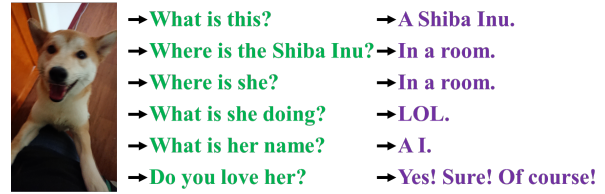

Figure 1: Human beings can answer different questions from the same input. Our aim is to train a single deep neural network that can serve many tasks.

inition of how implicit learning operates and how to obtain implicit knowledge. In the general definition of neural networks, the features obtained from the shallow layers are often called explicit knowledge, and the features obtained from the deep layers are called implicit knowledge. In this paper, we call the knowledge that directly correspond to observation as explicit knowledge. As for the knowledge that is implicit in the model and has nothing to do with observation, we call it as implicit knowledge.

We propose a unified network to integrate implicit knowledge and explicit knowledge, and enable the learned model to contain a general representation, and this general representation enable sub-representations suitable for various tasks. Figure 2.(c) illustrates the proposed unified network architecture.

The way to construct the above unified networks is to combine compressive sensing and deep learning, and the main theoretical basis can be found in our previous work [16, 17, 18]. In [16], we prove the effectiveness of reconstructing residual error by extended dictionary. In [17, 18], we use sparse coding to reconstruct feature map of a CNN and make it more robust. The contribution of this work are summarized as follows:

1. We propose a unified network that can accomplish various tasks, it learns a general representation by integrating implicit knowledge and explicit knowledge, and one can complete various tasks through this general representation. The proposed network effectively improves the performance of the model with a very small amount of additional cost (less than one ten thousand of the amount of parameters and calculations.)

(a) Multi-purpose, multi-analyzer, multi-discriminator.

(b) Multi-purpose, single analyzer, multi-discriminator.
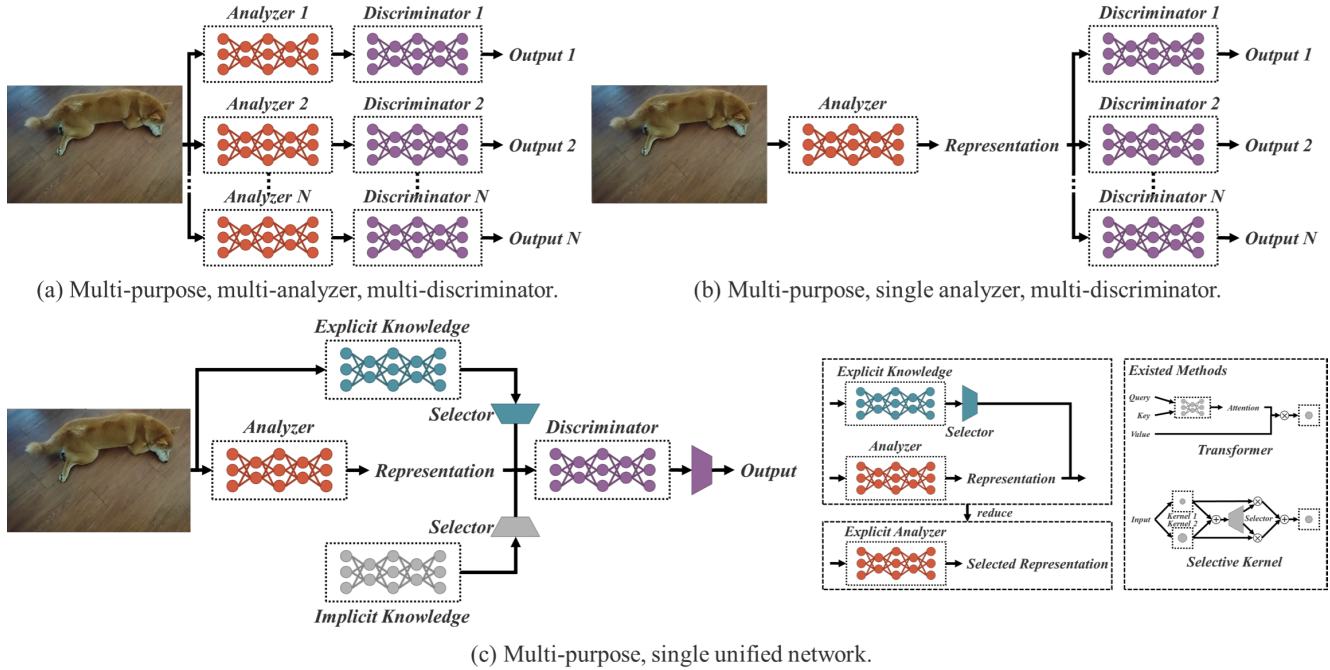
(c) Multi-purpose, single unified network.

Figure 2: Multi-purpose NN architectures. (a) distinct models for distinct tasks; (b) shared backbone, different heads for different tasks; and (c) our proposed unified network: one representation with explicit knowledge and implicit knowledge for serving multiple tasks.

2. We introduced kernel space alignment, prediction refinement, and multi-task learning into the implicit knowledge learning process, and verified their effectiveness.

3. We respectively discussed the ways of using vector, neural network, or matrix factorization as a tool to model implicit knowledge, and at the same time verified its effectiveness.

4. We confirmed that the proposed implicit representation learned can accurately correspond to a specific physical characteristic, and we also present it in a visual way. We also confirmed that if operators that conform to the physical meaning of an objective, it can be used to integrate implicit knowledge and explicit knowledge, and it will have a multiplier effect.

5. Combined with state-of-the-art methods, our proposed unified network achieved comparable accuracy as Scaled-YOLOv4-P7 [15] on object detection and the inference speed has been increased 88%.

2. Related work

We conduct a review of the literature related to this research topic. This literature review is mainly divided into three aspects: (1) explicit deep learning: it will cover some methods that can automatically adjust or select features based on input data, (2) implicit deep learning: it will cover the related literature of implicit deep knowledge learning and implicit differential derivative, and (3) knowledge modeling: it will list several methods that can be used to integrate implicit knowledge and explicit knowledge.

2.1. Explicit deep learning

Explicit deep learning can be carried out in the following ways. Among them, Transformer [14, 5, 20] is one way, and it mainly uses query, key, or value to obtain self-attention. Non-local networks [21, 4, 24] is another way to obtain attention, and it mainly extracts pair-wise attention in time and space. Another commonly used explicit deep learning method [7, 25] is to automatically select the appropriate kernel by input data.

2.2. Implicit deep learning

The methods that belong to the category of implicit deep learning are mainly implicit neural representations [11] and deep equilibrium models [2, 3, 19]. The former is mainly to obtain the parameterized continuous mapping representation of discrete inputs to perform different tasks, while the latter is to transform implicit learning into a residual form neural networks, and perform the equilibrium point calculation on it.

2.3. Knowledge modeling

As for the methods belonging to the category of knowledge modeling, sparse representation [1, 23] and memory networks [22, 12] are mainly included. The former uses exemplar, predefined over complete, or learned dictionary to perform modeling, while the latter relies on combining various forms of embedding to form memory, and enable memory to be dynamically added or changed.

3. How implicit knowledge works?

The main purpose of this research is to conduct a unified network that can effectively train implicit knowledge, so first we will focus on how to train implicit knowledge and inference it quickly in the follow-up. Since implicit representation $\mathbf{z}_i$ is irrelevant to observation, we can think of it as a set of constant tensor $Z = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_k\}$. In this section we will introduce how implicit knowledge as constant tensor can be applied to various tasks.

3.1. Manifold space reduction

We believe that a good representation should be able to find an appropriate projection in the manifold space to which it belongs, and facilitate the subsequent objective tasks to succeed. For example, as shown in Figure 3, if the target categories can be successfully classified by the hyperplane in the projection space, that will be the best outcome. In the above example, we can take the inner product of the projection vector and implicit representation to achieve the goal of reducing the dimensionality of manifold space and effectively achieving various tasks.
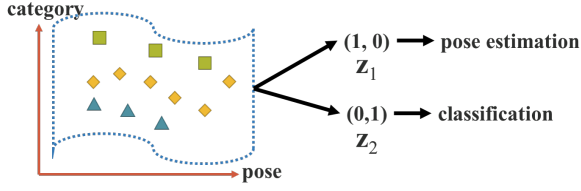

Figure 3: Manifold space reduction.

3.2. Kernel space alignment

In multi-task and multi-head neural networks, kernel space misalignment is a frequent problem, Figure 4.(a) illustrates an example of kernel space misalignment in multi-task and multi-head NN. To deal with this problem, we can perform addition and multiplication of output feature and implicit representation, so that Kernel space can be translated, rotated, and scaled to align each output kernel space of neural networks, as shown in Figure 4.(b). The above mode of operation can be widely used in different fields, such as the feature alignment of large objects and small objects in feature pyramid networks (FPN) [8], the use of knowledge distillation to integrate large models and small models, and the handling of zero-shot domain transfer and other issues.
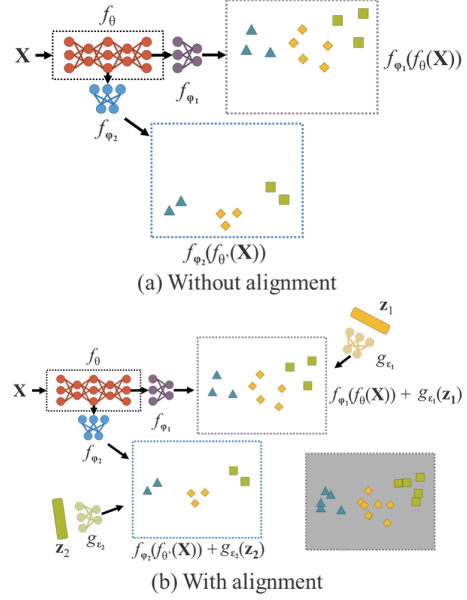

(a) Without alignment

(b) With alignment

Figure 4: Kernel space alignment.

3.3. More functions

In addition to the functions that can be applied to different tasks, implicit knowledge can also be extended into many more functions. As illustrated in Figure 5, through introducing addition, one can make neural networks to predict the offset of center coordinate. It is also possible to introduce multiplication to automatically search the hyperparameter set of an anchor, which is very often needed by an anchor-based object detector. Besides, dot multiplication and concatenation can be used, respectively, to perform multi-task feature selection and to set pre-conditions for subsequent calculations.
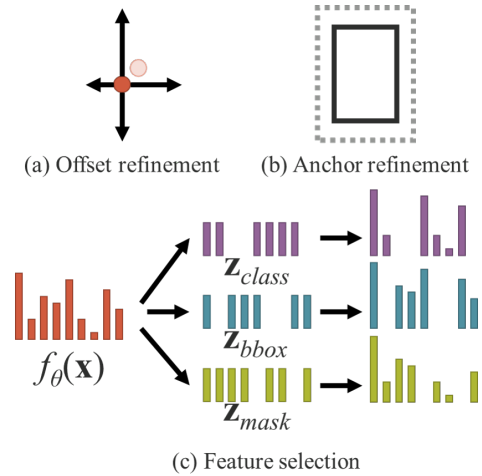

(a) Offset refinement   (b) Anchor refinement

(c) Feature selection

Figure 5: More functions.

4. Implicit knowledge in our unified networks

In this section, we shall compare the objective function of conventional networks and the proposed unified networks, and to explain why introducing implicit knowledge is important for training a multi-purpose network. At the same time, we will also elaborate the details of the method proposed in this work.

4.1. Formulation of implicit knowledge

Conventional Networks:

For the object function of conventional network training, we can use (1) to express as follows:

$$y = f_\theta(\mathbf{x}) + \epsilon$$
$$\text{minimize } \epsilon \tag{1}$$

where $\mathbf{x}$ is observation, θ is the set of parameters of a neural network, f_θ represents operation of the neural network, ϵ is error term, and y is the target of given task.

In the training process of a conventional neural network, usually one will minimize ϵ to make $f_\theta(\mathbf{x})$ as close to the target as possible. This means that we expect different observations with the same target to be a single point in the sub space obtained by f_θ, as illustrated in Figure 6.(a). In other words, the solution space we expect to obtain is discriminative only for the current task t_i and invariant to tasks other than t_i in various potential tasks, $T \setminus t_i$, where $T = \{t_1, t_2, ..., t_n\}$.

For general purpose neural network, we hope that the obtained representation can serve all tasks belonging to T. Therefore, we need to relax ϵ to make it possible to find solution of each task at the same time on manifold space, as shown in Figure 6.(b). However, the above requirements make it impossible for us to use a trivial mathematical method, such as maximum value of one-hot vector, or threshold of Euclidean distance, to get the solution of t_i. In order to solve the problem, we must model the error term ϵ to find solutions for different tasks, as shown in Figure 6.(c).

(a) Minimize ε (b) Relax ε
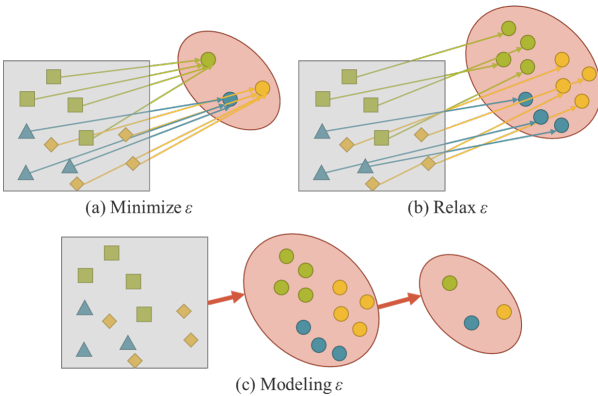
(c) Modeling ε

Figure 6: Modeling error term.

Unified Networks:

To train the proposed unified networks, we use explicit and implicit knowledge together to model the error term, and then use it to guide the multi-purpose network training process. The corresponding equation for training is as follows:

$$y = f_\theta(\mathbf{x}) + \epsilon + g_\phi(\epsilon_{ex}(\mathbf{x}), \epsilon_{im}(\mathbf{z}))$$
$$\text{minimize } \epsilon + g_\phi(\epsilon_{ex}(\mathbf{x}), \epsilon_{im}(\mathbf{z})) \tag{2}$$

where ϵ_{ex} and ϵ_{im} are operations which modeling, respectively, the explicit error and implicit error from observation $\mathbf{x}$ and latent code $\mathbf{z}$. g_ϕ here is a task specific operation that serves to combine or select information from explicit knowledge and implicit knowledge.

There are some existing methods to integrate explicit knowledge into f_θ, so we can rewrite (2) into (3).

$$y = f_\theta(\mathbf{x}) \star g_\phi(\mathbf{z}) \tag{3}$$

where $\star$ represents some possible operators that can combine f_θ and g_ϕ. In this work, the operators introduced in Section 3 will be used, which are addition, multiplication, and concatenation.

If we extend derivation process of error term to handling multiple tasks, we can get the following equation:

$$F(\mathbf{x}, \theta, \mathbf{Z}, \Phi, Y, \Psi) = 0 \tag{4}$$

where $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_T\}$ is a set of implicit latent code of T different tasks. Φ are the parameters that can be used to generate implicit representation from $\mathbf{Z}$. Ψ is used to calculate the final output parameters from different combinations of explicit representation and implicit representation.

For different tasks, we can use the following formula to obtain prediction for all $\mathbf{z} \in \mathbf{Z}$.

$$d_\Psi(f_\theta(\mathbf{x}), g_\Phi(\mathbf{z}), y) = 0 \tag{5}$$

For all tasks we start with a common unified representation $f_\theta(\mathbf{x})$, go through task-specific implicit representation $g_\Phi(\mathbf{z})$, and finally complete different tasks with task-specific discriminator d_Ψ.
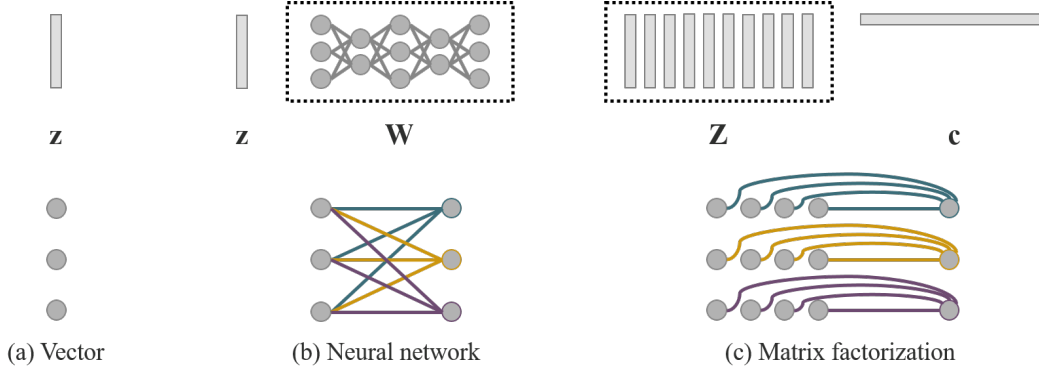
Figure 7: We proposed to use three different ways for modeling implicit knowledge. The top row shows the formation of these three different modeling approaches, and the bottom row shows their corresponding mathematical attributes. (a) Vector: single base, and each dimension is independent with another dimensions; (b) Neural Network: single or multiple basis, and each dimension is dependent to another dimensions; and (c) Matrix factorization: multiple basis, and each dimension is independent with another dimensions.

4.2. Modeling implicit knowledge

The implicit knowledge we proposed can be modeled in the following ways:

Vector / Matrix / Tensor:

$$\mathbf{z} \tag{6}$$

Use vector $\mathbf{z}$ directly as the prior of implicit knowledge, and directly as implicit representation. At this time, it must be assumed that each dimension is independent of each other.

Neural Network:

$$\mathbf{Wz} \tag{7}$$

Use vector $\mathbf{z}$ as the prior of implicit knowledge, then use the weight matrix $\mathbf{W}$ to perform linear combination or non-linearization and then become an implicit representation. At this time, it must be assumed that each dimension is dependent of each other. We can also use more complex neural network to generate implicit representation. Or use Markov chain to simulate the correlation of implicit representation between different tasks.

Matrix Factorization:

$$\mathbf{Z}^{\mathbf{T}}\mathbf{c} \tag{8}$$

Use multiple vectors as prior of implicit knowledge, and these implicit prior basis $\mathbf{Z}$ and coefficient $\mathbf{c}$ will form implicit representation. We can also further do sparse constraint to $\mathbf{c}$ and convert it into sparse representation form. In addition, we can also impose non-negative constraint on $\mathbf{Z}$ and $\mathbf{c}$ to convert them into non-negative matrix factorization (NMF) form.

4.3. Training

Assuming that our model dos not have any prior implicit knowledge at the beginning, that is to say, it will not have any effect on explicit representation $f_\theta(\mathbf{x})$. When the combining operator $\star \in \{addition, concatenation\}$, the initial implicit prior $\mathbf{z} \sim N(0, \sigma)$, and when the combing operator $\star$ is $multiplication$, $\mathbf{z} \sim N(1, \sigma)$. Here, σ is a very small value which is close to zero. As for $\mathbf{z}$ and ϕ, they both are trained with backpropagation algorithm during the training process.

4.4. Inference

Since implicit knowledge is irrelevant to observation $\mathbf{x}$, no matter how complex the implicit model g_ϕ is, it can be reduced to a set of constant tensor before the inference phase is executed. In other words, the formation of implicit information has almost no effect on the computational complexity of our algorithm. In addition, when the above operator is multiplication, if the subsequent layer is a convolutional layer, then we use (9) below to integrate. When one encounters an addition operator, and if the previous layer is a convolutional layer and it has no activation function, then one use (10) shown below to integrate.

$$\begin{aligned} \mathbf{x}_{(l+1)} &= \sigma(W_l(g_\phi(\mathbf{z})\mathbf{x}_l) + b_l) \\ &= \sigma(W_l^{'}(\mathbf{x}_l) + b_l), \text{where } W_l^{'} = W_l g_\phi(\mathbf{z}) \end{aligned} \tag{9}$$

$$\begin{aligned} \mathbf{x}_{(l+1)} &= W_l(\mathbf{x}_l) + b_l + g_\phi(\mathbf{z}) \\ &= W_l(\mathbf{x}_l) + b_l^{'}, \text{where } b_l^{'} = b_l + g_\phi(\mathbf{z}) \end{aligned} \tag{10}$$

5. Experiments

Our experiments adopted the MSCOCO dataset [9], because it provides ground truth for many different tasks, including [1]**object detection**, [2]**instance segmentation**, [3]**panoptic segmentation**, [4]**keypoint detection**, [5]**stuff segmentation**, [6]**image caption**, [7]**multi-label image classification**, and [8]**long tail object recognition**. These data with rich annotation content can help train a unified network that can support computer vision-related tasks as well as natural language processing tasks.

5.1. Experimental setup

In the experimental design, we chose to apply implicit knowledge to three aspects, including [1]**feature alignment for FPN**, [2]**prediction refinement**, and [3]**multi-task learning in a single model**. The tasks covered by multi-task learning include [1]object detection, [2]multi-label image classification, and [3]feature embedding. We choose YOLOv4-CSP [15] as the baseline model in the experiments, and introduce implicit knowledge into the model at the position pointed by the arrow in Figure 8. All the training hyper-parameters are compared to default setting of Scaled-YOLOv4 [15].
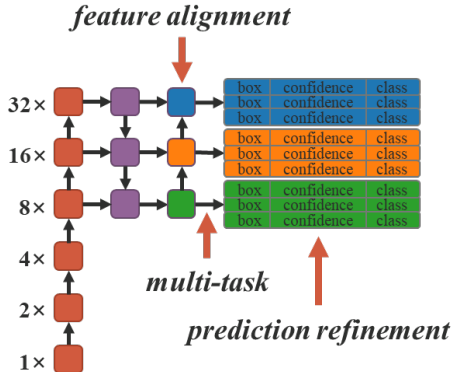

Figure 8: Architecture.

In Section 5.2, 5.3, and 5.4, we use the simplist vector implicit representation and addition operator to verify the positive impact on various tasks when implicit knowledge is introduced. In Section 5.5, we will use different operators on different combinations of explicit knowledge and implicit knowledge, and discuss the effectiveness of these combinations. In Section 5.6, we shall model implicit knowledge by using different approaches. In Section 5.7, we analyze the model with and without introduce implicit knowledge. Finally in Section 5.8, we shall train object detectors with implicit knowledge and then compare the performance with state-of-the-art methods.

5.2. Feature alignment for FPN

We add implicit representation into the feature map of each FPN for feature alignment, and the corresponding experiment results are illustrated in Table 1. From these results shown in Table 1 we can say: After using implicit representation for feature space alignment, all performances, including AP_S, AP_M, and AP_L, have been improved by about 0.5%, which is a very significant improvement.

Table 1: Ablation study of feature alignment.

Model	AP^{val}	AP_{50}^{val}	AP_{75}^{val}	AP_S^{val}	AP_M^{val}	AP_L^{val}
baseline	47.8%	66.3%	52.1%	30.1%	52.5%	62.0%
+ iFA	**47.9%**	**66.6%**	**52.3%**	**30.6%**	**53.1%**	**62.6%**

[*] baseline is YOLOv4-CSP-fast, tested on 640×640 input resolution.
[*] FA: feature alignment.

5.3. Prediction refinement for object detection

Implicit representations are added to YOLO output layers for prediction refinement. As illustrated in Table 2, we see that almost all indicator scores have been improved. Figure 9 shows how the introduction of implicit representation affects the detection outcome. In the object detection case, even we do not provide any prior knowledge for implicit representation, the proposed learning mechanism can still automatically learn (x, y), (w, h), (obj), and $(classes)$ patterns of each anchor.

Table 2: Ablation study of prediction refinement.

Model	AP^{val}	AP_{50}^{val}	AP_{75}^{val}	AP_S^{val}	AP_M^{val}	AP_L^{val}
baseline	47.8%	66.3%	52.1%	30.1%	52.5%	62.0%
+ iPR	**47.8%**	**66.5%**	**52.1%**	**30.3%**	**53.3%**	61.5%

[*] baseline is YOLOv4-CSP-fast, tested on 640×640 input resolution.
[*] PR: prediction refinement.

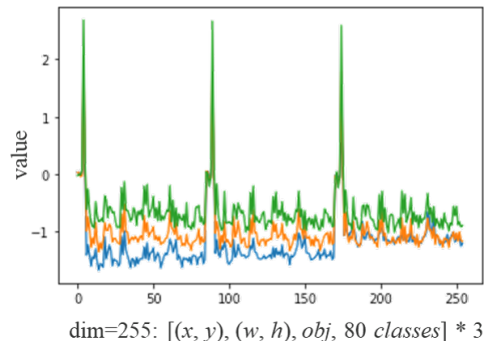dim=255: [(x, y), (w, h), *obj*, 80 *classes*] * 3

Figure 9: Value of learned implicit representation for prediction refinement.

5.4. Canonical representation for multi-task

When one wants to train a model that can be shared by many tasks at the same time, since the joint optimization process on loss function must be executed, multiple parties often pull each other during the execution process. The above situation will cause the final overall performance to be worse than training multiple models individually and then integrating them. In order to solve the above problem, we propose to train a canonical representation for multi-tasks. Our idea is to augment the representation power by introducing implicit representation to each task branch, and the effects it causes are listed in Table 3. As the data illustrated in Table 3, without the introduction of implicit representation, some index scores improved after multi-task training, and some dropped. After introducing implicit representation to joint detection and classification (JDC), in the model category corresponding to + iJDC, we can clearly see that the overall index score has increased significantly, and it has surpassed the performance of single-task training model. Compared to when implicit representation was not introduced, the performance of our model on medium-sized objects and large-sized objects has also been improved by 0.3% and 0.7%, respectively. In the experiment of joint detection and embedding (JDE), because of the characteristic of implicit representation implied by feature alignment, the effect of improving the index score is more significant. Among the index scores corresponding to JDE and + iJDE listed in Table 3, all index scores of + iJDE surpass the index that does not introduce implicit representation. Among them, the AP for large objects even increased by 1.1%.

Table 3: Ablation study of multi-task joint learning.

Model	$\mathbf{AP}^{val}$	$\mathbf{AP}^{val}_{50}$	$\mathbf{AP}^{val}_{75}$	$\mathbf{AP}^{val}_{S}$	$\mathbf{AP}^{val}_{M}$	$\mathbf{AP}^{val}_{L}$
baseline	48.0%	66.8%	52.3%	30.0%	53.0%	62.7%
JDC	47.7%	**66.8%**	51.9%	**30.8%**	52.4%	61.6%
+ iJDC	**48.1%**	**67.1%**	52.2%	**31.1%**	52.7%	62.3%
JDE	**48.1%**	66.7%	**52.4%**	30.7%	53.2%	61.9%
+ iJDE	**48.3%**	**66.8%**	**52.6%**	30.7%	**53.4%**	**63.0%**

* baseline is YOLOv4-CSP [15], tested on 640×640 input resolution.
* JD{C, E}: joint detection & {clssification, embedding}.

(a) input (b) '+' addition (c) '×' multiplication (d) '⊕' concatenation
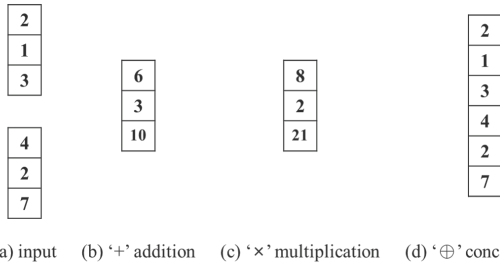
Figure 10: Implicit modeling with (a) addition, (b) multiplication, and (c) concatenation operators.

5.5. Implicit modeling with different operators

Table 4 shows the experimental results of using different operators shown in Figure 10 to combine explicit representation and implicit representation. In the implicit knowledge for feature alignment experiment, we see that addition and concatenation both improve performance, while multiplication actually degrades performance. The experimental results of feature alignment are in full compliance with its physical characteristics, because it must deal with the scaling of global shift and all individual clusters. In the implicit knowledge for prediction refinement experiment, since the operator of concatenation ill change the dimension of output, we only compare the effects of using addition and multiplication operators in the experiment. In this set of experiments, the performance of applying multiplication is better than that of applying addition. Analyzing the reason, we found that center shift uses addition decoding when executing prediction, while anchor scale uses multiplication decoding. Because center coordinate is bounded by grid, the impact is minor, and the artificially set anchor owns a larger optimization space, so the improvement is more significant.

Table 4: Ablation study of different operators.

Model	$\mathbf{AP}^{val}$	$\mathbf{AP}^{val}_{50}$	$\mathbf{AP}^{val}_{75}$	$\mathbf{AP}^{val}_{S}$	$\mathbf{AP}^{val}_{M}$	$\mathbf{AP}^{val}_{L}$
baseline	47.8%	66.3%	52.1%	30.1%	52.5%	62.0%
+ iFA	**47.9%**	**66.6%**	**52.3%**	**30.6%**	**53.1%**	**62.6%**
× iFA	47.4%	65.8%	51.6%	29.6%	52.2%	**62.1%**
⊕ iFA	47.8%	66.5%	52.2%	30.3%	52.9%	62.3%
+ iPR	47.8%	66.5%	52.1%	30.3%	53.3%	61.5%
× iPR	**48.0%**	**66.7%**	**52.3%**	29.8%	**53.4%**	61.8%

* baseline is YOLOv4-CSP-fast, tested on 640×640 input resolution.
* {+, ×, ⊕}: {addition, multiplication, concatenation}.

Based on the above analysis, we designed two other set of experiments – {× iFA*, × iPR*}. In the first set of experiments – × iFA*, we split feature space into anchor cluster level for combination with multiplication, while in the second set of experiments – × iPR*, we only performed multiplication refinement on width and height in prediction. The results of the above experiments are illustrated in Table 5. From the figures shown in Table 5, we find that after corresponding modifications, the scores of various indices have been comprehensively improved. The experiment shows that when we designing how to combine explicit and implicit knowledge, we must first consider the physical meaning of the combined layers to achieve a multiplier effect.

Table 5: Ablation study of different operators.

Model	$\mathbf{AP}^{val}$	$\mathbf{AP}^{val}_{50}$	$\mathbf{AP}^{val}_{75}$	$\mathbf{AP}^{val}_{S}$	$\mathbf{AP}^{val}_{M}$	$\mathbf{AP}^{val}_{L}$
baseline	47.8%	66.3%	52.1%	30.1%	52.5%	62.0%
× iFA*	**47.9%**	**66.6%**	52.0%	**30.5%**	**52.6%**	**62.3%**
× iPR*	**48.1%**	**66.5%**	52.1%	30.1%	**53.3%**	61.9%

* baseline is YOLOv4-CSP-fast, tested on 640×640 input resolution.

5.6. Modeling implicit knowledge in different ways

We tried to model implicit knowledge in different ways, including vector, neural networks, and matrix factorization. When modeling with neural networks and matrix factorization, the default value of implicit prior dimension is twice that of explicit representation dimension. The results of this set of experiments are shown in Table 6. We can see that whether it is to use neural networks or matrix factorization to model implicit knowledge, it will improve the overall effect. Among them, the best results have been achieved by using matrix factorization model, and it upgrades the performance of AP, AP_{50}, and AP_{75} by 0.2%, 0.4%, and 0.5%, respectively. In this experiment, we demonstrated the effect of using different modeling ways. Meanwhile, we confirmed the potential of implicit representation in the future.

Table 6: Ablation study of different modeling approaches.

Model	AP^{val}	AP_{50}^{val}	AP_{75}^{val}	AP_S^{val}	AP_M^{val}	AP_L^{val}
baseline	47.8%	66.3%	52.1%	30.1%	52.5%	62.0%
+ *i*FA	**47.9%**	**66.6%**	**52.3%**	**30.6%**	**53.1%**	**62.6%**
+ *wi*FA	47.8%	66.4%	52.0%	**30.8%**	**52.8%**	61.9%
+ *ic*FA	**48.0%**	**66.7%**	**52.6%**	30.3%	**53.2%**	**62.5%**

* baseline is YOLOv4-CSP-fast, tested on 640×640 input resolution.
* $\{i, wi, ic\}$: {vector, neural network, matrix factorization}, see 4.2.

5.7. Analysis of implicit models

We analyze the number of parameters, FLOPs, and learning process of model with/w/o implicit knowledge, and show the results in Table 7 and Figure 11, respectively. From the experimental data, we found that in the model with implicit knowledge set of experiments, we only increased the amount of parameters and calculations by less than one ten thousandth, which can significantly improve the performance of the model, and the training process can also converge quickly and correctly.

Table 7: Information of model with/without implicit knowledge.

Model	AP^{val}	# parameters	MFLOPs
baseline 1	47.8%	52908989	117517.2952
implicit 1	**48.0%**	**52911546** (+0.005%)	**117519.4372** (+0.002%)
baseline 2	51.4%	37262204	326256.1624
implicit 2	**51.9%**	**37265016** (+0.008%)	**326264.7304** (+0.003%)

* baseline 1 is YOLOv4-CSP-fast, tested on 640×640 input resolution.
* baseline 2 is YOLOv4-P6-light, tested on 1280×1280 input resolution.
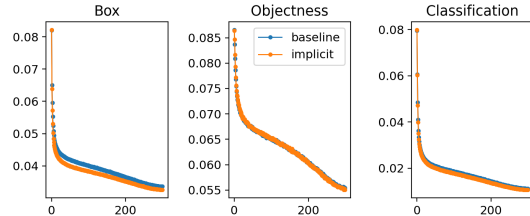* implicit $\{1, 2\}$ are baseline $\{1, 2\}$ with + *i*FA, × *i*PR.

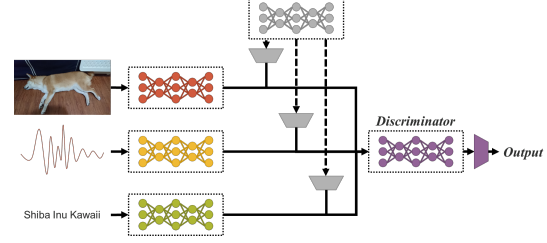Figure 11: Learning curve of model with/w/o implicit knowledge.

Figure 12: Multimodal unified netwrok.

5.8. Implicit knowledge for object detection

Finally, we compare the effectiveness of the proposed method with object detection's state-of-the-art methods. The benefits of introducing implicit knowledge are shown in Table 8. For the entire training process, we follow the scaled-YOLOv4 [15] training process, that is, train from scratch 300 epochs first, and then fine-tune 150 epochs. Table 9 illustrates the comparisons with the state-of-the-art methods. One thing worth noting is that our proposed method does not have additional training data and annotations. By introducing the unified network of implicit knowledge, we still achieve results that are sufficient to match the state-of-the-art methods.

Table 8: Benefit from implicit knowledge.

Model	AP^{val}	AP_{50}^{val}	AP_{75}^{val}	AP_S^{val}	AP_M^{val}	AP_L^{val}
baseline	51.4%	69.5%	56.4%	35.2%	55.8%	64.6%
implicit	**51.9%**	**69.8%**	**56.8%**	**36.0%**	**56.3%**	**65.0%**
fine-tuned implicit	**52.5%**	**70.5%**	**57.6%**	**37.1%**	**57.2%**	**65.4%**

* baseline is YOLOv4-P6-light, tested on 1280×1280 input resolution.
* implicit is baseline with + *i*FA, × *i*PR.

Table 9: Comparion of state-of-the-art.

Method	pre.	seg.	add.	AP^{test}	AP_{50}^{test}	AP_{75}^{test}	FPS^{V100}
YOLOR (ours)				55.4%	73.3%	60.6%	30
ScaledYOLOv4 [15]				55.5%	73.4%	60.8%	16
EfficientDet [13]	✓			55.1%	74.3%	59.9%	6.5
SwinTransformer [10]	✓	✓		57.7%	–	–	–
CenterNet2 [26]	✓		✓	56.4%	74.0%	61.6%	–
CopyPaste [6]	✓	✓	✓	57.3%	–	–	–

* pre. : large dataset image classification pre-training.
* seg. : training with segmentation ground truth.
* add. : training with additional images.

6. Conclusions

In this paper, we show how to construct a unified network that integrates implicit knowledge and explicit knowledge, and prove that it is still very effective for multi-task learning under the single model architecture. In the future, we shall extend the training to multi-modal and multi-task, as shown in Figure 12.

7. Acknowledgements

References

[1] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006. 2

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 2

[3] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2

[4] Yue Cao, Jiarui Xu, Stephen Lin, Fangyun Wei, and Han Hu. GCNet: Non-local networks meet squeeze-excitation networks and beyond. In *Proceedings of the IEEE International Conference on Computer Vision Workshop (ICCV Workshop)*, 2019. 2

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 213–229, 2020. 2

[6] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. *arXiv preprint arXiv:2012.07177*, 2020. 8

[7] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 510–519, 2019. 2

[8] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017. 3

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755, 2014. 6

[10] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021. 8

[11] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2

[12] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. 2

[13] Mingxing Tan, Ruoming Pang, and Quoc V Le. EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 8

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 2

[15] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-YOLOv4: Scaling cross stage partial network. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2, 6, 7, 8

[16] Chien-Yao Wang, Seksan Mathulaprangsan, Bo-Wei Chen, Yu-Hao Chin, Jing-Jia Shiu, Yu-San Lin, and Jia-Ching Wang. Robust face verification via bayesian sparse representation. In *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 1–4, 2016. 1

[17] Chien-Yao Wang, Andri Santoso, Seksan Mathulaprangsan, Chin-Chin Chiang, Chung-Hsien Wu, and Jia-Ching Wang. Recognition and retrieval of sound events using sparse coding convolutional neural network. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 589–594, 2017. 1

[18] Chien-Yao Wang, Tzu-Chiang Tai, Jia-Ching Wang, Andri Santoso, Seksan Mathulaprangsan, Chin-Chin Chiang, and Chung-Hsien Wu. Sound events recognition and retrieval using multi-convolutional-channel sparse coding convolutional neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 28:1875–1887, 2020. 1

[19] Tiancai Wang, Xiangyu Zhang, and Jian Sun. Implicit feature pyramid network for object detection. *arXiv preprint arXiv:2012.13563*, 2020. 2

[20] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021. 2

[21] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7794–7803, 2018. 2

[22] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *International Conference on Learning Representations (ICLR)*, 2015. 2

[23] John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(2):210–227, 2008. 2

[24] Minghao Yin, Zhuliang Yao, Yue Cao, Xiu Li, Zheng Zhang, Stephen Lin, and Han Hu. Disentangled non-local neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 191–207, 2020. 2

[25] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. ResNeSt: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020. 2

[26] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Probabilistic two-stage detection. *arXiv preprint arXiv:2103.07461*, 2021. 8
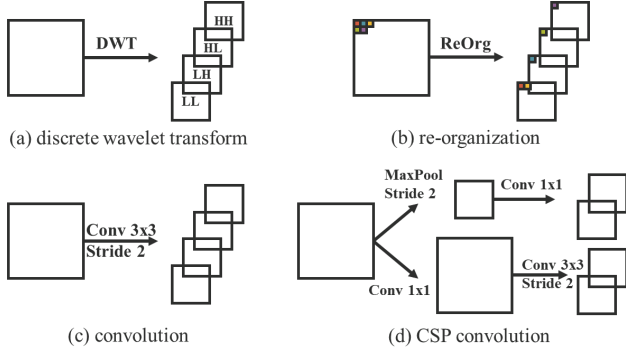
A. Appendix

(a) discrete wavelet transform

(b) re-organization

(c) convolution

(d) CSP convolution

Figure A1: We use four kind of down-sampling modules in this work, including (a) discrete wavelet transform (DWT): https://github.com/fbcotter/pytorch_wavelets, (b) re-organization (ReOrg): https://github.com/AlexeyAB/darknet/issues/4662#issuecomment-608886018, (c) convolution, and (d) CSP convolution used in CSPNet: https://github.com/WongKinYiu/CrossStagePartialNetworks/tree/pytorch.

(a) Stem A

(b) Stem B

(c) Stem C
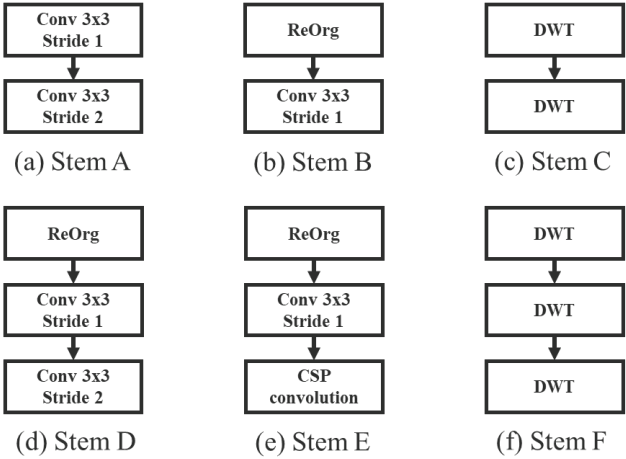
(d) Stem D

(e) Stem E

(f) Stem F

Figure A2: We use down-sampling modules in Figure A1 to form stem blocks: (a) Stem A is used in YOLOv4-CSP, (b) Stem B is used in YOLOv4-CSP-fast, (c) Stem C is used in YOLOv4-CSP-SSS, (d) Stem D is proposed by 10.5281/zenodo.4679653 and called focus layer, it is used in YOLOv4-P6-light, YOLOR-P6, and YOLOR-W6, (e) Stem E is used in YOLOR-E6 and YOLOR-D6, and (f) Stem F is used in YOLOv4-CSP-SSSS.

(a) Topology I

(c) Topology III

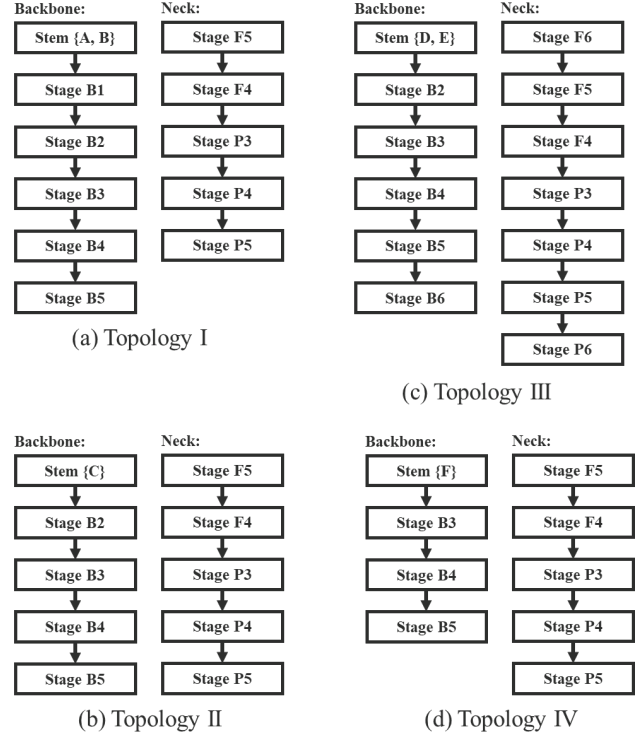(b) Topology II

(d) Topology IV

Figure A3: Models in this paper can be mapped to three four of architecture topology. Due to Stem C, D, and E contain two down-sampling modules, models used those stem blocks has no Stage B1 in the backbone, for same reason Stem F has no Stage B1 and B2.

• YOLOv4-CSP belongs to Topology 1, the architecture is described in Scaled-YOLOv4 paper.

• YOLOv4-CSP-fast is modified from YOLOv4-CSP, we replace Stem A in YOLOv4-CSP by Stem B to form YOLOv4-CSP-fast.

• YOLOv4-CSP-SSS belongs to Topology 2, Stem C is used in this model. The topology after Stage B2 is as same as YOLOv4-CSP, and width scaling factor and depth scaling factor are set as 0.5 and 0.33, respectively. We then using SiLU activation to replace all Mish activation in the model.

• YOLOv4-CSP-SSSS is modified from YOLOv4-CSP-SSS, Stem C in YOLOv4-CSP-SSS is replaced by Stem F in this model. Due to the stem block contains three down-sampling modules, YOLOv4-CSP-SSSS belongs to topology IV.

• YOLOv4-P6-light belongs to Topology 3, it uses Stem D and base channels are set as {128, 256, 384, 512, 640}. To optimize the gradient propagation, we apply CSP fusion first in B* stages and the repeat number of B2 to B6 are set as {3, 7, 7, 3, 3}.

• YOLOR-P6 has same architecture as YOLOv4-P6-light, we replace all Mish activation in YOLOv4-P6-light by SiLU activation.

• YOLOR-W6 is wider YOLOR-P6, base channels are set as {128, 256, 512, 768, 1024}.

• YOLOR-E6 expands the width of YOLOR-W6, the width scaling factor is set as 1.25, and all of convolution down-sampling modules are replaced by CSP convolution.

• YOLOR-D6 is deeper YOLOR-E6, the repeat number of B2 to B6 are set as {3, 15, 15, 7, 7}

Table A1: Lightweight models with implicit knowledge.

Model	YOLOR	Size	$\mathbf{FPS}_{batch32}^{TitanRTX}$	FLOPs	# parameters	$\mathbf{AP}^{val}$	$\mathbf{AP}_{50}^{val}$	$\mathbf{AP}_{75}^{val}$	$\mathbf{AP}_{S}^{val}$	$\mathbf{AP}_{M}^{val}$	$\mathbf{AP}_{L}^{val}$
Y4-SSS		640	720	17.9G	9290077	38.8%	57.8%	42.1%	21.3%	44.0%	52.4%
Y4-SSS	✓	640	712	17.9G	9291738	39.3%	58.1%	42.5%	21.7%	44.4%	52.8%
					+0.018%	**+0.5%**	**+0.3%**	**+0.4%**	**+0.4%**	**+0.4%**	**+0.4%**
Y4-SSSS		640	806	16.1G	9205693	36.3%	54.8%	39.3%	17.7%	40.7%	49.7%
Y4-SSSS	✓	640	791	16.1G	9207354	36.8%	55.1%	39.7%	18.9%	41.4%	50.8%
					+0.018%	**+0.5%**	**+0.3%**	**+0.4%**	**+1.2%**	**+0.7%**	**+1.1%**
U5R5-S		640	569	17.0G	7266973	36.7%	55.4%	39.8%	22.2%	41.9%	46.2%
U5R5-S	✓	640	563	17.0G	7268634	37.3%	56.5%	40.5%	21.1%	42.7%	47.7%
					+0.023%	**+0.6%**	**+1.1%**	**+0.7%**	**-1.1%**	**+0.8%**	**+1.5%**

[*] Y4: YOLOv4-CSP, U5R5: 10.5281/zenodo.4679653; FPS: model inference only.
• **YOLOR**-Y4-SSSS get 0.1% better AP than U5R5-S with 39% faster inference speed.

Table A2: Large models with implicit knowledge.

Model	Size	$\mathbf{FPS}_{batch32}^{TitanRTX}$	FLOPs	# parameters	$\mathbf{AP}^{val}$	$\mathbf{AP}_{50}^{val}$	$\mathbf{AP}_{75}^{val}$	$\mathbf{AP}_{S}^{val}$	$\mathbf{AP}_{M}^{val}$	$\mathbf{AP}_{L}^{val}$
YOLOR-P6	1280	72	326.2G	37265016	52.5%	70.6%	57.4%	37.4%	57.3%	65.2%
YOLOR-W6	1280	66	454.0G	79873400	54.0%	72.1%	59.1%	38.1%	58.8%	67.0%
YOLOR-E6	1280	39	684.0G	115909400	54.6%	72.5%	59.8%	39.9%	59.0%	67.9%
YOLOR-D6	1280	31	936.8G	151782680	55.4%	73.5%	60.6%	40.4%	60.1%	68.7%
Y4-P6	1280	34	718.4G	127530352	54.4%	72.7%	59.5%	39.5%	58.9%	67.3%
U5R5-S6	1280	139	69.6G	12653596	43.3%	61.9%	47.7%	29.0%	48.0%	53.3%
U5R5-M6	1280	93	209.6G	35889612	50.5%	68.7%	55.2%	35.5%	55.2%	62.0%
U5R5-L6	1280	67	470.8G	77218620	53.4%	71.1%	58.3%	38.2%	58.4%	65.7%
U5R5-X6	1280	36	891.6G	141755500	54.4%	72.0%	59.1%	40.1%	59.0%	67.2%

[*] Y4: YOLOv4-CSP, U5R5: 10.5281/zenodo.4679653; FPS: model inference only.
• Y4-P6 get better AP than U5R5-X6 with 24% less computation and 11% fewer #parameters.
• **YOLOR**-E6 get better AP than Y4-P6 with 5% less computation, 10% fewer #parameters, and 15% faster inference speed.

Table A3: More comparison.

Model	Size	$\mathbf{FPS}^{V/R}$	FLOPs	# parameters	$\mathbf{AP}^{test}$	$\mathbf{AP}_{50}^{test}$	$\mathbf{AP}_{75}^{test}$	$\mathbf{AP}_{S}^{test}$	$\mathbf{AP}_{M}^{test}$	$\mathbf{AP}_{L}^{test}$
YOLOR-P6	1280	49 / 48	326G	37M	52.6%	70.6%	57.6%	34.7%	56.6%	64.2%
YOLOR-P6D	1280	49 / 48	326G	37M	53.0%	71.0%	58.0%	35.7%	57.0%	64.6%
YOLOR-W6	1280	47 / 44	454G	80M	54.1%	72.0%	59.2%	36.3%	57.9%	66.1%
YOLOR-E6	1280	37 / 27	684G	116M	54.8%	72.7%	60.0%	36.9%	58.7%	66.9%
YOLOR-D6	1280	30 / 22	937G	152M	55.4%	73.3%	60.6%	38.0%	59.2%	67.1%
ST-L (HTC++)	–	–	1470G	284M	57.7%	–	–	–	–	–
C2 (R2)	1560	– / 5	–	–	56.4%	74.0%	61.6%	38.7%	59.7%	68.6%
Y4-P5	896	41 / –	328G	71M	51.8%	70.3%	56.6%	33.4%	55.7%	63.4%
Y4-P6	1280	30 / –	718G	128M	54.5%	72.6%	59.8%	36.8%	58.3%	65.9%
Y4-P7	1536	16 / –	1639G	287M	55.5%	73.4%	60.8%	38.4%	59.4%	67.7%
P2	640	50* / –	–	–	50.3%	69.0%	55.3%	31.76%	53.9%	62.4%

Model	Size	$\mathbf{FPS}^{V/R}$	FLOPs	# parameters	$\mathbf{AP}^{val}$	$\mathbf{AP}_{50}^{val}$	$\mathbf{AP}_{75}^{val}$	$\mathbf{AP}_{S}^{val}$	$\mathbf{AP}_{M}^{val}$	$\mathbf{AP}_{L}^{val}$
ST-T (MRCNN)	–	15.3 / –	745G	86M	50.5%	69.3%	54.9%	–	–	–
ST-S (MRCNN)	–	12.0 / –	838G	107M	51.8%	70.4%	56.3%	–	–	–
ST-B (MRCNN)	–	11.6 / –	982G	145M	51.9%	70.9%	56.5%	–	–	–
ST-B (HTC++)	–	–	1043G	160M	56.4%	–	–	–	–	–
ST-L (HTC++)	–	–	1470G	284M	57.1%	–	–	–	–	–
C2 (DLA)	640	– / 38	–	–	49.2%	–	–	–	–	–

[*] ST: SwinTransformer, C2: CenterNet2, Y4: YOLOv4-CSP, P2: PP-YOLOv2.
[*] HTC: Hybrid Task Cascade, R2: Res2Net, MRCNN: Mask R-CNN, DLA: Deep Layer Aggregation.
[*] FPS: end-to-end batch one inference speed of V100/TitanRTX, FPS value with * indicates speed of model inference only.
• **YOLOR**-P6D means joint train **YOLOR**-P6 model with **YOLOR**-D6 model.
• **YOLOR**-D6 get 0.9% better AP than Y4-P6 with almost same inference speed.
• **YOLOR**-D6 get 88% faster inference speed than Y4-P7 with almost same AP.