Plugin Networks for Inference under Partial Evidence

Michal Koperski[*]
Tooploox
michal.koperski@tooploox.com

Tomasz Konopczyński[*]
Heidelberg University
Tooploox

Rafał Nowak
Institute of Computer Science
University of Wroclaw
Tooploox

Piotr Semberecki
Wroclaw University of Science and Technology
Tooploox

Tomasz Trzciński
Warsaw University of Technology
Tooploox

Abstract

In this paper, we propose a novel method to incorporate partial evidence in the inference of deep convolutional neural networks. Contrary to the existing, top-performing methods, which either iteratively modify the input of the network or exploit external label taxonomy to take the partial evidence into account, we add separate network modules ("Plugin Networks") to the intermediate layers of a pre-trained convolutional network. The goal of these modules is to incorporate additional signal, i.e. *information about known labels, into the inference procedure, and adjust the predicted output accordingly. Since the attached plugins have a simple structure, consisting of only fully connected layers, we drastically reduced the computational cost of training and inference. Also, the proposed architecture allows propagating information about known labels directly to the intermediate layers to improve the final representation. Extensive evaluation of the proposed method confirms that our Plugin Networks outperform the state-of-the-art in a variety of tasks, including scene categorization, multi-label image annotation, and semantic segmentation.*

1. Introduction

Visual recognition tasks, *e.g.* scene categorization or multi-label image annotation, have attracted a significant amount of research interest in recent years [26, 11, 28, 5]. One of the reasons, which sparked this attention was the availability of evaluation datasets created for benchmarking given visual tasks, such as ImageNet [2], VOC Pascal [4], or COCO [17]. Although sensible for comparison purposes, single-task evaluation protocols are often far from real-life use-cases, where additional information, *e.g.* related to location or time of photo capture, is available.

The partial information (*partial evidence*) about an image, which may be available during inference, can improve the accuracy of pre-trained networks [11, 26]. More specifically, we assume that a set of labels corresponding to a given image is known during inference, while the task at hand is to improve the performance of the model on the original task, *e.g.* image classification, object detection, and semantic segmentation. This corresponds to a real life application, where, for instance, we know that the image was captured in a forest or in a cave, which drastically reduces the likelihood of detecting a skyscraper. Similarly, information that a given object appears in an image can greatly improve its localization or segmentation. Since partial evidence can be available in multiple forms and modalities, the main prediction system, *e.g.* a convolutional neural network (CNN), is trained to perform a general purpose prediction with no assumption about the existence of partial evi-

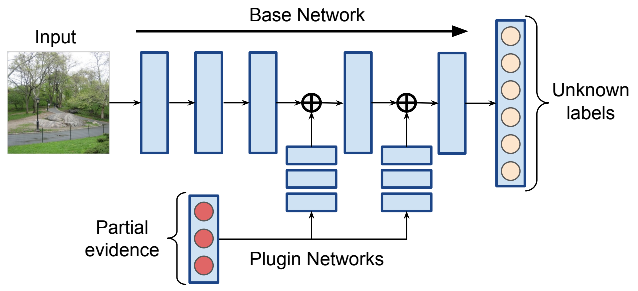

Figure 1: Plugin Networks — neural networks attached to the intermediate layers of a pre-trained convolutional neural network, allow exploiting partial evidence labels at the inference to predict unknown labels with higher accuracy. This simple, yet effective approach significantly reduces training and inference time, while outperforming competitive results on three challenging benchmarks.

[*]Equal contribution

dence or lack thereof. Neural architectures such as CNNs are not modular, thus any modification such as new inputs (partial evidence) or new outputs (new tasks) are difficult to apply without repeating the full training procedure. Otherwise, phenomena such as the catastrophic forgetting may occur. Our objective is to enable the model to incorporate additional available information without re-training the main system while exploiting this information to increase the quality of predictions.

Several methods were proposed in the literature to address this problem, among them [11] and [26] are the most recent. In paper [11] authors proposed to exploit external taxonomy of the labels by modeling correspondences between scene attributes and categories, by feeding this data into the main neural network at inference. On the other hand, paper [26] introduces a feedback-prop approach that iteratively modifies input and network activations to ensure the response of the network corresponds to the distribution of known labels. Although these methods provide effective ways to exploit partial evidence, they require complex reasoning, that concerns relationships between labels or computationally expensive iterative adaptation mechanism.

In this work, we reduce the complexity and propose the Plugin Networks – a simple, yet effective main network extension that allows incorporating partial evidence during the inference. We show that by using a set of fully-connected (FC) side networks attached to intermediate layers of the main network (see Fig. 1), we are able to not only avoid costly optimization process but also exploit the assumption about the existence of partial evidence in the offline training stage. More specifically, the proposed Plugin Networks, connected to the backbone neural network, adjust their activations at the time of inference, depending on available known labels. Due to the simplicity of the Plugin Networks, their training converges quickly, while remaining robust to overfitting, as we show in this paper. The inference of the proposed model consists of a quick feed-forward propagation of the main model. Plugin Networks offer a significant speedup with respect to the state-of-the-art feedback-prop method [26]. Last but not least, the proposed Plugin Networks outperform all of the existing methods on three challenging benchmark applications: hierarchical scene categorization on the SUN397 dataset [28], multi-label image annotation on the COCO 2014 dataset [17], and semantic segmentation on Pascal VOC 2011 [4].

To summarize, our contributions are as follows:

- We propose novel neural network model extensions called Plugin Networks, which allows us to take partial information available during inference into account. Plugin Networks adjust the activations of the pre-trained base network. They are fast to train and efficient during inference.
- We show how to attach the proposed Plugin Networks to different types of neural network layers and investigate the influence of those variants on final results.
- We provide an extensive evaluation of the proposed approach on three challenging tasks: hierarchical scene categorization, multi-label image annotation, and scene segmentation.

We make our code available for the public[1].

In the remainder of this paper, we first give an overview of related publications. In Sec. 3, we formally introduce the proposed approach, explain how to use it, and discuss its properties. Sec. 4 provides an extensive evaluation of our method and we conclude this paper in Sec. 5.

2. Related Work

Using context in visual tasks: Exploiting additional contextual cues in visual recognition tasks gained a lot of attention from the computer vision community [3, 6, 13]. Contextual information related to semantics was used to improve object detection [20]. Social media meta-data was also used in a context of multilabel image annotation in [13]. Although, adding context proved to be successful in increasing the quality of visual recognition tasks, all of the methods mentioned above used the context in conjunction with the input uni-modal (visual) image during the training of the entire system. In this work, we propose a fundamentally different approach since the context (in the form of known labels) is learned only after the training of the main model is finished, and our approach allows us to extend this pre-trained model with additional information *a posteriori*. Rosenfeld *et al.* [24] proposed a method where detection and segmentation are conditioned on the presence of a given object category. To achieve it, they propose to use a set of linear modulators. This method shares common features such as offline training with the Plugin Networks, but a model capacity of linear modulators is not enough to learn complex functions. This leads to more complicated training procedures with data oversampling. Perez *et al.* [22] proposed FiLM (Feature-wise Linear Modulation) for visual question answering, where textual information serves as a context to a CNN model. In particular, FiLM introduces new layers (named FiLM blocks or Resblocks), which are incorporated into the base model. FiLM blocks are later modulated using an affine fusion operator. Plugin Networks, on the other hand, do not introduce any alternations to the base model architecture and directly modulate activations of the base model. Thanks to that, we do not introduce so much noise as the FiLM Resblocks to the base model, which results in more stable training.

Using label structure: Some authors proposed to model the co-occurrence of labels available at training time to improve recognition performance [19]. [1], on the other hand,

[1]https://github.com/tooploox/plugin-networks

uses a special structure to store the relations between the labels using a graph designed specifically to capture semantic similarities between the labels. Other forms of external knowledge can be found in [8] and [12], where they use the WordNet taxonomy of tags to increase the accuracy of their visual recognition systems. [13], [18] also used social media meta-data to improve the quality of the results obtained for image recognition tasks. Finally, [21] estimated entry-level labels of visual objects by exploiting image captions. Contrary to our method, the approaches mentioned above focus on finding the relationships between the labels and driving the training algorithm to encompass those structures. In this work, we do not explicitly model any label structures – the only input related to labels we give to the network is a set of known labels related to an image with no information about their relationship with the others.

Multi-task learning: Somehow related to our work is the thriving area of multi-task learning. Motivated by the phenomenon of catastrophic forgetting, multi-task learning tries to address the problem of lifelong learning and adaptation of a neural network to a set of changing tasks while preserving the network's structure. In [16], Lee *et al.* aim to solve this problem by the continuous matching of network distribution. In [23], the same problem is solved through residual adapters – neural network modules plugged into a network, similarly to our Plugin Networks – which are the only structures trained for the tasks while the base network remains untouched. Although we do not aim to solve multi-task learning problem in this work, our approach is inspired by the methods mentioned above, which focus on designing robust network architecture that can dynamically adjust to additional data point sources unseen during training.

Inference with Partial Evidence: Finally, the most relevant to the work presented in this paper are two methods proposed by Hu *et al.* [11] and Wang *et al.* [26]. Both of them address the problem of visual tasks in the presence of partial evidence.

Hu *et al.* [11] tackle this challenge by proposing a Structured Inference Neural Network (SINN). The SINN method is designed to discover the hierarchical structure of labels, but it can also be used in a partial evidence setup if labels in a given hierarchy are clamped at inference. However, the SINN model, which uses CNN and LSTM to discover label relations, has a large amount of learnable parameters, which makes model training difficult. To solve this issue, authors use the positive and negative correlations of labels as prior knowledge, which is inferred from the WordNet relations. We compare our method with SINN and show that we achieve significantly better performance with a much simpler model.

The FeedbackProp proposed by Wang *et al.* [26], uses an iterative procedure, which is applied at inference time. The idea is to modify network activations to maximize the probabilities of labels under the partial evidence. The method does not require to re-train the base model. However, due to the iterative procedure introduced at inference time, it requires more computational effort. Also, they introduced hyperparameters, like a number of iterations and learning rate to the inference phase. Finally, in the case of FeedbackProp, the partial evidence labels can only be a subset of labels that the base model can recognize. Our method, however, can accept any kind of labels as partial evidence. Moreover, our method introduces negligible computations and no extra parameters to the inference phase. The comparison shows that our method outperforms FeedbackProp while being significantly faster at the inference phase.

3. Plugin Networks

In this section, we first introduce the Plugin Networks and define them formally. We then describe how to attach the Plugin Networks to the existing base network at the linear and convolutional layers.

3.1. Definition

Let's assume that we have a CNN model $F(\mathbf{x}; \mathbf{w})$, where $\mathbf{x}$ is an input image and $\mathbf{w}$ are the parameters. The model F is already trained on some task (*e.g.* single or multi-label classification, scene segmentation). The parameters $\mathbf{w}$ were trained on input images X and input labels Y.

Now let us assume that some labels $\bar{Y}$ are available and known at inference time. In the following definitions without loosing generality, we will assume that only one Plugin Network is attached to the base model. We define the Plugin Network model F_p with parameters $\mathbf{w}_p$ as

$$\mathbf{r} = F_p(\bar{\mathbf{y}}; \mathbf{w}_p). \tag{1}$$

The model takes the partial evidence $\bar{\mathbf{y}} \in \bar{Y}$ as an input. The output $\mathbf{r}$ of the plugin can be attached to the output vector $\mathbf{z}$ of some layer of the base model F:

$$\tilde{\mathbf{z}} = \mathbf{z} \oplus \mathbf{r}, \tag{2}$$

where the sign $\oplus$ can have the following meaning:

- additive: $\tilde{\mathbf{z}} = \mathbf{z} + \mathbf{r}$,
- affine: $\tilde{\mathbf{z}} = \mathbf{r_a}\mathbf{z} + \mathbf{r_b}, \mathbf{r} = \mathbf{r_a} \parallel \mathbf{r_b}$,
- multiplicative: $\tilde{\mathbf{z}} = \mathbf{z} * \mathbf{r}$,
- residual: $\tilde{\mathbf{z}} = \mathbf{z} + \mathbf{z} * \mathbf{r}$,

where $\parallel$ is the concatenation operator.

In this way, the Plugin Network F_p adapts the output vector $\mathbf{z}$ of the base model F under the presence of available partial evidence. The eq. (2) defines how the Plugin Network F_p is attached to the base network F. Thus a joint model can be defined as:

$$\tilde{F}(\mathbf{x}, \bar{\mathbf{y}}; \mathbf{w}, \mathbf{w}_{p_i}). \tag{3}$$

In general, several Plugin Networks can be attached simultaneously to a number of layers of the base model F.

Note that the output of a Plugin Network $\mathbf{r}$ can be attached to either the output of a fully connected layer or a convolutional layer. In the following sections, we explain how both operations are performed in details.

3.2. Connection with Linear Layers

The Plugin Network, which is attached to the linear layer, has to compute the vector $\mathbf{r}$ of the same dimension as the vector $\mathbf{z}$. Then the operator $\oplus$ in eq. (2) is well defined. The adjusted vector $\tilde{\mathbf{z}}$ is then processed by the following layers of the base model. The value $\mathbf{z}$ is the output of a layer before a non-linear function (*e.g.* ReLU).

3.3. Connection with Convolutional Layers

When a Plugin Network is attached to a convolutional layer, it adjusts the feature map obtained from a given convolutional filter. Thus, it has to compute vector $\mathbf{r}$, which has to be of the same dimension as the number of channels in the tensor $\mathbf{z}$. Then all the considered operators $\oplus$ in eq. (2) have elementwise meaning:

$$\tilde{\mathbf{z}}_c = \mathbf{z}_c \oplus r_c = \begin{bmatrix} z_{11} & \cdots & z_{1j} \\ \vdots & \ddots & \\ z_{i1} & & z_{ij} \end{bmatrix}_c \oplus r_c, \tag{4}$$

where c indicates the number of channel.

Adding a scalar value to each channel of a feature map greatly reduces the number of Plugin Network parameters that have to be learned. Learning different values for each element of a feature map requires $w \times h \times c$ output values, where w, h stands for width and height of a feature map, respectively. Since we only add scalar value to each feature map, we require only c output values from a Plugin Network. Fig. 2 illustrates how Plugin Network is attached to convolutional layers.

3.4. Plugin Network Architecture

Overall, the Plugin Networks can be generalized to any model that can be trained with backpropagation. In our case, the Plugin Network is a FC neural network. Each FC layer is followed by a ReLU activation except for the last layer. We chose a fully connected architecture because partial evidence vector $\bar{\mathbf{y}}$ can be interpreted as a feature vector, which is used to compute a non-linear transform of outputs of the base model. This task can be well handled by a fully connected neural network.

3.5. Training

The eq. (3) defines joint model of a base network F with parameters $\mathbf{w}$ and the Plugin Network F_p with parameters $\mathbf{w}_p$. In the training procedure, we are optimizing only $\mathbf{w}_p$ parameters. Thus the base model F is not altered. To optimize the parameters of the Plugin Networks, the original loss function is used, *i.e.* the same loss function that was used to train the base model.

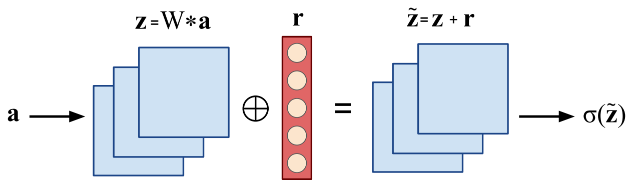

Figure 2: If we add Plugin Network to a convolutional layer, then each element of the output vector is added to a corresponding channel of feature maps. For instance, if convolutional layer has c channels, then output from the Plugin Network has also c elements.

3.6. Properties

One important property of the Plugin Networks is that the function, which modifies a base model, is trained in an offline phase (see Sec. 3.5). Thus, the testing phase requires only a single feed-forward propagation through the base model and the Plugin Networks. Thanks to the single forward pass, our model is fast, and forward propagation overhead of the Plugin Networks is negligible. Thus, our method is significantly faster than the model proposed in [26], where iterative optimization process is applied at the inference phase.

4. Experiments

In this section, we evaluate the Plugin Networks on three challenging computer vision tasks. We consider a hierarchical, multi-label classification and semantic segmentation problems. To stay consistent with the previous work on these subjects, we conduct the experiments in the same setups as [11, 24, 26]. Therefore, we use SUN397 [28, 27], COCO'14 [17] and Pascal VOC 2011 [4] datasets.

4.1. Hierarchical Scene Categorization

We apply our method on SUN397 dataset [28, 27]. The dataset is annotated with 3 coarse categories, 16 general scene categories, and 397 fine-grained scene categories. Our task is to classify fine-grained categories, given true values for coarse categories, as it was performed in Hu *et al.* [11] and Wang *et al.* [26]. Thus, coarse categories serve as the partial evidence. We follow the same experimental setup as [11, 26]: we split the dataset into train, validation, and test split with 50, 10, and 40 images per scene category. To allow fair comparison to [11, 26], we use the AlexNet [15] with Softmax trained on fine-grained categories. It will serve as the base model for the Plugin

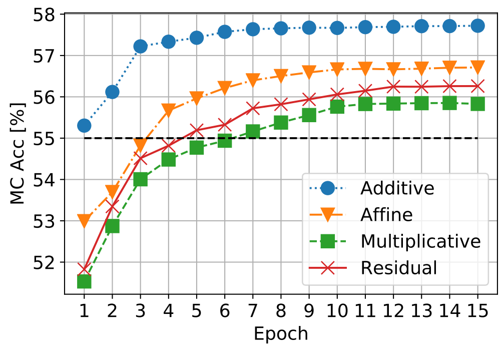

Figure 3: Plugin Networks – fusion operators ablation study on SUN397 dataset. Black dashed line marks state-of-the-art.

Layer	MC Acc
no plugin	53.15
conv1	54.08
conv2	53.88
conv3	53.75
conv4	54.30
conv5	54.86
conv3-5	56.88

Layer	MC Acc
fc1	53.90
fc2	56.47
fc3	**57.51**
fc1-3	57.08
conv3-5, fc1-3	57.16

Table 1: Performance of the Plugin networks w.r.t. the number of plugins and layers at which they are attached to the AlexNet. The comparison is done on SUN397 dataset. The performance is better if the plugin is attached to deeper layers. Plugins attached to FC layers perform better. The most effective is plugin attached to fc3 layer, although it outperforms models slightly where three and six plugins were attached.

Networks. To evaluate our method, we compute mean average precision (mAP), multi-class accuracy (MC Acc), and intersection-over-union accuracy (IoU Acc).

Ablation study: Plugin Network can be attached to different layers of the base model. Furthermore, one can attach more than one Plugin Network simultaneously. Now, we analyze different combinations of the above choices. The results (see Tab. 1) show that Plugin Network improves the performance of the base model regardless to which layer it is attached. On the other hand, the performance gain differs between chosen layers. Thus, a couple of observations can be drawn. It is more effective to attach a Plugin Network to an FC layer rather than a convolutional layer. Secondly, the Plugin Network connected to deeper layers tends to obtain better performance. The results are aligned with the intuition, that in case of the classification task, deeper layers carry more abstract information. Thus, the Plugin Network can converge to a better solution when attached to deeper layers. Moreover, in the classification task, spatial information is ignored. Thus modification of convolutional layers, which carry spatial information, is less important. Note, that in case of semantic segmentation task described in Sec. 4.3, it is more important to modify convolutional layers. These experiments show that the Plugin Networks are generic and can solve various tasks. We do not observe further performance improvements if more than one Plugin Network is attached simultaneously. In the case of the classification task, the Plugin Network mainly learn the relationship between partial evidence and output labels. Thus, fc3 outputs carry enough information to find such a relationship.

Layer	# of hidden layers				
	0	1	2	3	4
conv5	53.41	54.06	54.18	**54.28**	53.90
fc3	54.53	56.65	57.18	**57.51**	56.56

Table 2: Performance of the Plugin Network (MC Acc) with respect to different number of hidden layers on SUN397. We consider two cases. Former is the Plugin Network being attached to the conv5 layer, while the latter — attached to the fc3 layer. The experiment shows that simple linear transformation (no hidden layers) is not sufficient.

In the second ablation study, we consider different Plugin Network architectures. We evaluate the number of hidden layers of the Plugin Network. We check from zero to four hidden layers. The best results are obtained by the network with three hidden layers, which is still a quite shallow architecture that allows efficient training and does not add significant overhead in the inference; see Tab. 2. The results also show that a linear function (model with no hidden layers) has not enough capacity to adjust base model outputs.

In the third ablation study, we compare different fusion operators from Eq. 2. The results show that the best performance and convergence rate is obtained by the additive operator, which is a particular case of the affine operator where $\mathbf{r_a} = 1$; see. Fig. 3. Because we use the ReLU activation function, the translation operation performed by the additive operator is sufficient to alter channel activation values by translating them to either positive or negative half-plane. Scaling (in the multiplicative operator) can achieve similar effect when $\mathbf{r_a} \approx 0$, but negative $\mathbf{r_a}$ may switch the sign of activation values, resulting in amplification of unwanted responses. Finally, we can observe that all proposed fusion operators achieve state-of-the-art results. In all further experiments, we are using the additive operator, since it achieved the best performance consistently.

In the final model, we use AlexNet CNN + Softmax as the base model. The CNN was pretrained on the Places365 dataset [29]. The base model was chosen to allow a fair comparison with [26, 11]. We use the Plugin Network with three hidden layers, attached to the fc3 layer from the base model. The model was trained for 15 epochs using the Adam [14] optimizer with the Xavier initialization [7] with learning rate set to 10^{-3}, which was reduced to 10^{-4} and 10^{-5} after five and ten epochs.

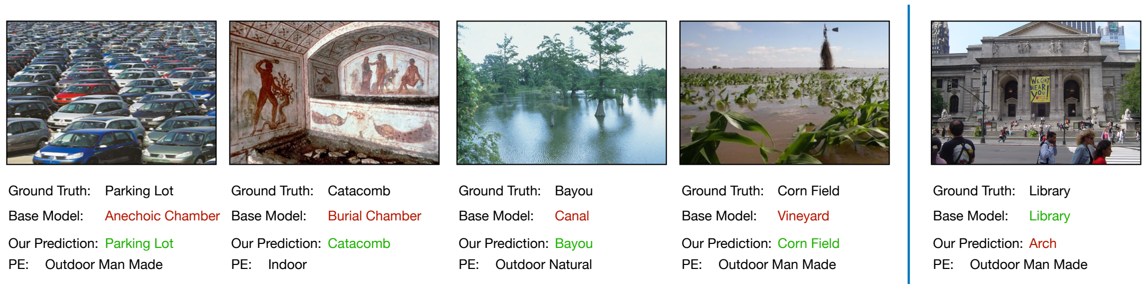

Figure 4: Visualization of results (best viewed in color). We pick five representative images from the SUN397 test set and visualize the predicted fine-grained categories from our method. We compare them with the predictions from the base model (CNN+Softmax). Correct predictions are marked in green, incorrect in red. Failure cases are shown in the rightmost column. "PE" stands for partial evidence.

	% of training examples				
	20	40	60	80	100
MC Acc	56.58	57.07	57.26	57.34	57.51
mAP	60.75	61.19	61.26	61.27	61.37
IoU	38.98	39.39	39.42	39.60	39.62

Table 3: Performance of the Plugin Network model trained on a fraction of training data. The results show that model trained even with 20% of available data, achieves state-of-the-art performance on all metrics. The results are reported on SUN397 dataset.

	MC Acc	mAP	IoU Acc
Base model [26]	52.83±0.24	56.17±0.21	35.90±0.22
SINN [11, 26]	54.30±0.35	58.34±0.32	37.28±0.34
F. Prop (LF) [26]	54.94±0.42	58.52±0.34	37.86±0.39
F. Prop (RF) [26]	55.01±0.35	58.70±0.26	37.95±0.33
Base (Ours)	53.30±0.29	56.36±0.21	34.39±0.31
Plugin Net (Ours)	**57.59±0.24**	**61.55±0.43**	**39.26±0.38**

Table 4: Plugin Network performance on SUN397 dataset. Our method outperforms the state-of-the-art on all reported metrics. To allow fair comparison, we also show the performance of the base model used in [26] as well as the performance of the base model trained by us.

Training analysis: In Tab. 3, we report the performance of our method w.r.t. to the percentage of available training data. We trained Plugin Network with 20%, 40%, 60%, and 80% percent of the training data. The results show that our model achieves state-of-the-art performance even when trained on 20% of the data.

Comparison with the state-of-the-art: In Tab. 4, we report the performance of our Plugin Network. The results are averaged over five runs to mitigate the randomness in validation set sampling, also standard deviation is computed. We report the performance of the base model, which does not use partial evidence as a reference. We also report the performance of SINN network [11] and FeedbackProp [26]. The results show that Plugin Networks outperform state-of-the-art methods in terms of MC Acc, mAP, and IoU; see. Tab. 4. Also, our method is easier to train than SINN model and allows faster inference than FeedbackProp.

Observations: In Fig. 4, we show five images from the SUN397 test set. For each example, we show: ground-truth label for fine-grained category, both classification results from the base and our model. We also report the coarse category (partial evidence). The examples show that our method can recover many errors thanks to the presence of partial evidence information. For instance, in top left example, partial evidence that "parking lot" belongs to "Outdoor man made" category helped to correct the classification error. The base model classified example as "Anechoic Chamber", which belongs to "Indoor" category. If we look at the "Anechoic Chamber" examples, one can notice that cars in the parking lot can mimic patterns of the "Anechoic Chamber" walls.

4.2. Multi-label Image Annotation

In the evaluation of our method for the multi-label image annotation task, we use the COCO 2014 dataset [17]. It contains 120,000 images, each annotated with five caption sentences. Again, for consistency, we follow the same experimental setup as [26]. Namely, we use the provided 82,783 training data instances as our training set, and randomly split the remaining provided validation data into 20,000 validation set and 20,504 test set images.

The task is to predict a predefined set of words explaining an image. The words are referred as visual concepts in Fang *et al.* in their visual concept classifier [5]. We define them as the 1,000 most frequent words in the captions of the COCO dataset. We use the same tokenization, lemmatization, and stop-word removals as Wang *et al.* [26]. As a result, each image is annotated by a vector of 1,000 elements corresponding to an occurrence of words in the captions.

For the task of reasoning under partial evidence, we randomly divide the target vector into a fixed 500 known and 500 unknown classes. The model performance is measured on the unknown set only, while the known set is used as the partial evidence. The base network is first trained as

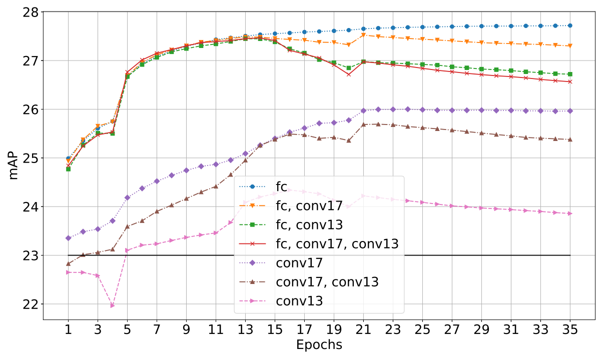

Figure 5: Ablation study of Plugin Network attachment layer. Base network is ResNet-18. Plugins are attached to the 17th and 13th conv layers and the last FC layer. The plugins architecture consists of two layered FC network with 500, and 2048 neurons. The solid black line indicates mAP achieved by ResNet-18 w/o Plugin Network. Attachment to the last fc layer only results in the highest improvement.

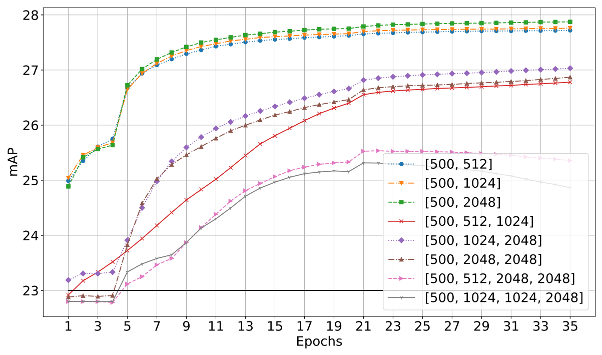

Figure 6: Ablation study of Plugin Network architecture. Base network is ResNet-18. Plugin network is attached to the last layer. Numbers in brackets indicate the number of neurons at consecutive layers. The solid black line indicates mAP achieved by ResNet-18 w/o Plugin Network. If the plugin has too many layers, it starts to overfit. We report the highest performance by using a two-layered network.

in Fang *et al.* [5] on the multi-labeled task using the entire 1,000 classes. It is done by minimization of the binary cross-entropy between the predicted and target vector of concepts. For this experiment, for the base network, we choose to use the ResNet-18 architecture [10] to stay consistent with [26]. Additionally, we also make an ablation study for deeper base network architectures: ResNet-50 and ResNet-101.

Hyperparameters selection: The architecture of the plugin has been chosen based on the validation scores from Figs. 5 and 6. As indicated before, we first train the base network on the given task. Then, we freeze the base network's weights and add a number of plugins. We train each plugin for 36 epochs using the Adam [14] optimizer with the Xavier initialization [7] with a starting learning rate of $1e{-3}$, which decreases with the number of epochs to $1e{-4}$.

	no plugin	FC, RL4, RL3	FC, RL4	**FC**
ResNet18	23.00	27.56	27.61	**27.97**
ResNet50	25.84	29.85	29.65	**29.93**
ResNet101	26.56	29.49	29.79	**30.13**

Table 5: Scores obtained when attaching plugins at different places to different versions of ResNet. We consider attachments to the last FC layer, end of the third residual layer (RL3), and fourth residual layer (RL4). Plugins always consist of 500 and 2048 neurons. For this task, we always achieve the highest score when applying a single plugin to the last layer of the base network.

	mAP	Inference time [s]
Base model [5]	23.00	25.64
F. Prop (LF) [26]	25.26	93.36
F. Prop (RF) [26]	25.70	103.27
Plugins (Ours)	**27.97**	**25.72**

Table 6: Results on the COCO'14. The baseline is ResNet-18 trained for the multi-label experiment. Our method not only achieves the state-of-the-art in means of the mAP but also is the fastest during inference, being barely slower than the base network.

During the hyperparameters selection, we verified all combinations of attaching a plugin to the conv13, conv17, and FC layers of the ResNet-18. We report that a single attachment to the last FC layers results in the highest mAP. Using any earlier layer still improves the baseline, but such plugin overfits much easier. Using a combination of the FC layer and any other convolutional layer leads to a lower performance comparing to a single attachment to the FC layer; see Fig. 5. These results are aligned with the conclusions from Sec. 4.1.

In the next experiment, we search for the best architecture of a single plugin. We consider different number of layers and neurons in the Plugin Networks. See Fig. 6 for details. The highest score is achieved by using the two-layered architectures with 500 and 2048 neurons, respectively. The two and three-layered networks get to the plateau after around 20 epochs, while the four-layered networks start to overfit.

Comparison with the state-of-the-art: We compare ourselves to the Layer-wise Feedback-prop (LF) and Residual Feedback-prop (RF) Inference proposed by [26]. Results presented in this work are based on the open-sourced online implementation[2] provided by the authors of RF and LF. Due to the random choice of the known and unknown labels, the baseline may differ, but the overall gain stays similar. We show that in terms of the mAP, we achieve the state-of-the-art with a significant margin. Furthermore, as expected, the inference phase is much faster compared to the Feedback-

[2]github.com/uvavision/feedbackprop

Model	# of plugins	mean-IoU
Baseline	0	65.5
conv1-3	3	65.7
conv1-5	5	70.5
deconv1-3	3	71.1
deconv1-5	5	71.2
conv1-5, deconv1-5	10	**72.2**

Table 7: Scores obtained when attaching plugins at different layers of the FCN architecture for the task of semantic segmentation.

prop methods. Please refer to Tab. 6 for results.

Finally, we verify the usage of the Plugin Networks for deeper architectures, such as ResNet50 and ResNet101; see Tab. 5. We notice an improvement for each base network. As for the ResNet-18, we achieve the highest score when only one plugin is used at the last FC layer.

4.3. Scene semantic segmentation

In this section, we evaluate the Plugin Networks on multi-cue object class segmentation task. We take fully convolutional network (FCN) [25] as a starting point. Next, we experiment by adding several plugins to its architecture. For the baseline we take the pre-trained FCN-8s model[3] trained on the SBD dataset [9]. We use the same dataset when training the Plugin Networks. We validate our models on the Pascal VOC 2011 segmentation challenge dataset [4]. We follow [25] and take *Pascal VOC 2011 segval*[4] validation split in order to avoid overlapping images between these two datasets. Thus, our training and validation datasets consist of 8498 and 736 images, respectively. Objects in the image are assigned to one of 21 classes.

The base model (FCN-8s without plugins) results in IoU score of 65.5%. For this scenario, we assume that we know classes present in the image at the inference time, which constitutes partial evidence. Therefore, our partial evidence is a vector of 21 elements. The goal of the Plugin Network is to improve the output segmentation masks of the base network. We experiment with attaching several plugins to the FCN-8s model. We report the results in Tab. 7. In contrary to findings from previous experiments, the Plugin Networks provide the highest increase in the IoU, when multiple of them are used. We achieve the highest gain of 72.2% of IoU when attaching five plugins into all of the convolutional layers and all of the transposed convolutional layers. We also outperform the previously proposed method [24], which achieved 69.2%.

Using partial evidence through the Plugin Networks, we are able to soften the wrong feature maps and strengthen the expected ones. It results in major improvement of the base network. When multiple objects are present in the scene, the base model may have a problem to assign a proper label to a particular object consistently. As expected, when the baseline network makes a mistake by assigning a wrong label of a class that is not present in the image, plugins correct these with a correct class. The examples are shown in Fig. 7. For instance, in the last row, the pixels belonging to a bottle are inconsistently assigned to different classes by the base model. Using the Plugin Networks fixes the problem.

[3]github.com/wkentaro/pytorch-fcn

[4]github.com/shelhamer/fcn.berkeleyvision.org

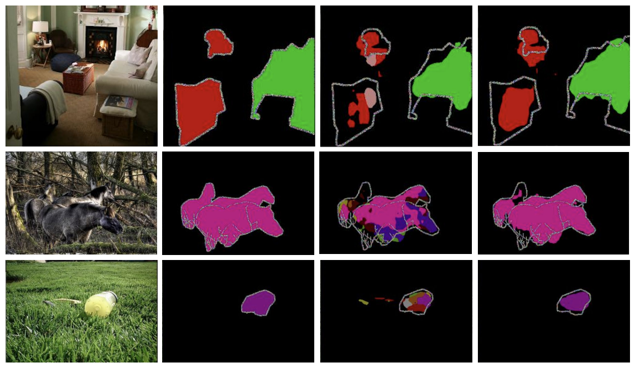

Figure 7: Examples of semantic segmentation masks predicted by the base model with and without Plugin Networks. The images are taken from the Pascal VOC 2011 validation set. In rows, we show different examples, and in columns, from left to right: input image, ground-truth (all classes), base model, FCN-8s with ten plugins. Our method shows a clear improvement in the output segmentation masks.

5. Conclusions

In this work, we introduced the Plugin Networks – a simple, yet effective method to exploit the availability of partial evidence in the context of visual recognition tasks. Plugin Networks are integrated directly with the intermediate layers of pre-trained convolutional neural networks, and thanks to their lightweight design can be trained efficiently with low computational cost and limited amount of data. Results presented on three challenging tasks and various datasets show the superior performance of the proposed method with respect to the state-of-the-art approaches.

We believe that this work can open novel research directions related to solving visual recognition tasks with partial evidence, as our Plugin Networks are agnostic to the input signal and can accommodate arbitrary modality of the input data, including audio or textual cues. Therefore, their multimodal nature can allow richer contextual cues to be taken into account in the inference procedure, leading to more effective and efficient visual recognition models.

Acknowledgements

This research was partially supported by the Polish National Science Centre grant no. UMO-2016/21/D/ST6/01946.

References

[1] J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam. Large-scale object classification using label relation graphs. In *ECCV*, 2014. 2

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 1

[3] S. K. Divvala, D. Hoiem, J. Hays, A. A. Efros, and M. Hebert. An empirical study of context in object detection. In *CVPR*, 2009. 2

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop. 1, 2, 4, 8

[5] H. Fang, S. Gupta, F. N. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, C. L. Zitnick, and G. Zweig. From captions to visual concepts and back. In *CVPR*, 2015. 1, 6, 7

[6] C. Galleguillos, A. Rabinovich, and S. J. Belongie. Object categorization using co-occurrence, location and appearance. In *CVPR*, 2008. 2

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010. 5, 7

[8] K. Grauman, F. Sha, and S. J. Hwang. Learning a tree of metrics with disjoint visual features. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *NIPS*, 2011. 3

[9] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *ICCV*, 2011. 8

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 7

[11] H. Hu, G.-T. Zhou, Z. Deng, Z. Liao, and G. Mori. Learning structured inference neural networks with label relations. In *CVPR*, 2016. 1, 2, 3, 4, 5, 6

[12] S. J. Hwang, K. Grauman, and F. Sha. Semantic kernel forests from multiple taxonomies. In *NIPS*, 2012. 3

[13] J. Johnson, L. Ballan, and L. Fei-Fei. Love thy neighbors: Image annotation by exploiting image metadata. *ICCV*, 2015. 2, 3

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5, 7

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NIPS*, 2012. 4

[16] S. Lee, J. Kim, J. Ha, and B. Zhang. Overcoming catastrophic forgetting by incremental moment matching. *CoRR*, abs/1703.08475, 2017. 3

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *ECCV*, 2014. 1, 2, 4, 6

[18] J. McAuley and J. Leskovec. Image labeling on a network: Using social-network metadata for image classification. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *ECCV*, 2012. 3

[19] A. Miech, I. Laptev, and J. Sivic. Learnable pooling with context gating for video classification. *CoRR*, abs/1706.06905, 2017. 2

[20] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. In *CVPR*, 2014. 2

[21] V. Ordonez, J. Deng, Y. Choi, A. C. Berg, and T. L. Berg. From large scale image categorization to entry-level categories. In *ICCV*, 2013. 3

[22] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*, 2018. 2

[23] S. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *NIPS*, 2017. 3

[24] A. Rosenfeld, M. Biparva, and J. K. Tsotsos. Priming neural networks. In *CVPR Workshops*, June 2018. 2, 4, 8

[25] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *PAMI*, 39(4):640–651, 2017. 8

[26] T. Wang, K. Yamaguchi, and V. Ordonez. Feedback-prop: Convolutional neural network inference under partial evidence. In *CVPR*, 2018. 1, 2, 3, 4, 5, 6, 7

[27] J. Xiao, K. A. Ehinger, J. Hays, A. Torralba, and A. Oliva. Sun database: Exploring a large collection of scene categories. *IJCV*, 119:3–22, 2014. 4

[28] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 1, 2, 4

[29] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *PAMI*, 2017. 5