Meta Module Network for Compositional Visual Reasoning

Wenhua Chen, Zhe Gan, Linjie Li, Yu Cheng, William Wang, Jingjing Liu
1Microsoft Dynamics 365 AI Research, Bellevue, WA, USA
2University of California, Santa Barbara, CA, USA
{wenhuchen,william}@cs.ucsb.edu
{zhe.gan,lindsey.li,yu.cheng,jingjl}@microsoft.com

Abstract

Neural Module Network (NMN) exhibits strong interpretability and compositionality thanks to its handcrafted neural modules with explicit multi-hop reasoning capability. However, most NMNs suffer from two critical drawbacks: 1) scalability: customized module for specific function renders it impractical when scaling up to a larger set of functions in complex tasks; 2) generalizability: rigid predefined module inventory makes it difficult to generalize to unseen functions in new tasks/domains. To design a more powerful NMN architecture for practical use, we propose Meta Module Network (MMN) centered on a novel meta module, which can take in function recipes and morph into diverse instance modules dynamically. The instance modules are then woven into an execution graph for complex visual reasoning, inheriting the strong explainability and compositionality of NMN. With such a flexible instantiation mechanism, the parameters of instance modules are inherited from the central meta module, retaining the same model complexity as the function set grows, which promises better scalability. Meanwhile, as functions are encoded into the embedding space, unseen functions can be readily represented based on its structural similarity with previously observed ones, which ensures better generalizability. Experiments on GQA and CLEVR datasets validate the superiority of MMN over state-of-the-art NMN designs. Synthetic experiments on held-out unseen functions from GQA dataset also demonstrate the strong generalizability of MMN. Our code and model are released in Github1.

1. Introduction

Visual reasoning requires a model to learn strong compositionality and generalization abilities, i.e., understanding and answering compositional questions without having seen similar semantic compositions before. Such compositional visual reasoning is a hallmark for human intelligence that endows people with strong problem-solving skills given limited prior knowledge. Neural module networks (NMNs) [2, 3, 15, 21, 14, 28] have been proposed to perform such complex reasoning tasks. NMN requires a set of pre-defined functions and explicitly encodes each function into unique shallow neural networks called modules, which are composed dynamically to build an instance-specific network for each input question. This approach has high compositionality and interpretability, as each module is designed to accomplish a specific sub-task, and multiple modules can be combined to perform an unseen combination of functions during inference.

1https://github.com/wenhuchen/Meta-Module-Network
However, NMN suffers from two major limitations. 1) **Scalability**: When the complexity of the task increases, the set of functional semantics scales up, so does the number of neural modules. For example, in the recent GQA dataset [17], a larger set of functions (48 vs 25, see Appendix for details) with varied arity is involved, compared to previous CLEVR dataset [20]. To solve this task with standard NMN framework [2, 15, 28], an increased amount of modules are required to implement for these functions, leading to higher model complexity. 2) **Generalizability**: Since the model is tied to a pre-defined set of functionalities when a new question with unseen functional semantics appears, no existing module can be readily applied to the new semantics, limiting the model’s ability to generalize.

In order to enhance NMN for more practical use, we propose Meta Module Network (MMN). As depicted in Figure 1, MMN is based on a meta module (a general-purpose neural network), which can take a function recipe (key-value pairs) as input to embed it into continuous vector space and feed it as a side input to instantiate different instance modules. Depending on the specification provided in function recipe, different instance modules are created to accomplish different sub-tasks. As different instance modules inherit the same parameters from the meta module, model complexity remains the same as the function set enlarges. For example, if the recipe has \( K \) slots, and each slot takes \( N \) values, a compact vector of the recipe can represent up to \( N^K \) different functions. This effectively solves the scalability issue of NMN. When creating instance modules for specified sub-tasks, the input recipes are encoded into the embedding space for function instantiation. Thus, when an unseen recipe appears, it can be encoded into the embedding space to instantiate a novel instance module based on embedding similarity with previously observed recipes. This metamorphous design effectively overcomes NMN’s limitation on generalizability.

MMN draws inspiration from Meta Learning [35, 32, 9] as a learning-to-learn approach - instead of learning independent functions to solve different sub-tasks, MMN learns a meta-function that can generate a function to solve specific sub-task. The learning algorithm of MMN is based on a teacher-student framework to provide module supervision: an accurate “symbolic teacher” first traverses a given scene graph to generate the intermediate outputs for the given functions from specific recipe; the intermediate outputs are then used as guidelines to teach each “student” instance module to accomplish its designated sub-task in the function recipe. The module supervision together with the original question answering supervision are used jointly to train the model.

The model architecture of MMN is illustrated in Figure 2: (1) the coarse-to-fine semantic parser converts an input question into its corresponding program (i.e., a sequence of functions); (2) the meta module is instantiated into different instance modules based on the function recipes of the predicted program, which is composed into an execution graph; (3) the visual encoder encodes the image features that are fed to the instance modules; (4) during training, we provide intermediate module supervision and end-step answer supervision to jointly train all the components.

Our main contributions are summarized as follows. (i) We propose Meta Module Network that effectively extends the scalability and generalizability of NMN for more practical use, allowing it to handle tasks with unseen compositional function from new domain. With a metamorphous meta module learned through teacher-student supervision, MMN provides great flexibility on model design and model training that cleverly overcomes the rigid hand-crafting of NMN. (ii) Experiments conducted on CLEVR and GQA benchmarks demonstrate the scalability of MMN to accommodate larger set of functions. (iii) Qualitative visualization on the inferential chain of MMN also demonstrates its superb interpretability and strong transferability.

2. Related Work

**Neural Module Networks** By parsing a question into a program and executing the program through dynamically composed neural modules, NMN excels in interpretability and compositionality by design [2, 3, 15, 21, 14, 44, 28, 40]. For example, IEP [15] and N2NMN [21] aims to make the whole model end-to-end trainable via the use of reinforcement learning. Stack-NMN [14] proposes to make soft layout selection so that the whole model is fully differentiable, and Neural-Symbolic VQA [44] proposes to perform completely symbolic reasoning by encoding images into scene graphs. However, its success is mostly restricted to simple datasets with a limited set of functions, whose performance can be surpassed by simpler methods such as relational network [34] and FiLM [31]. Our MMN is a module network in concept, thus possessing high interpretability and compositionality. However, different from traditional NMN, to enhance its scalability and generalizability, MMN uses only a general-purpose meta module for program execution recurrently, which makes MMN inherently a monolithic network, ensuring its strong empirical performance without sacrificing model interpretability.

**Monolithic Network** Another line of research on visual reasoning is focused on designing monolithic network architecture, such as MFB [46], BAN [23], DCN [29], and MCAN [45]. These black-box models have achieved strong performance on challenging datasets, such as VQA [4, 11] and GQA [18], surpassing the NMN approach. More re-
cently, multimodal pre-training algorithms [38, 27, 36, 7, 10] have been proposed that further lift state of the art on diverse tasks such as VQA [11], NL VR^2 [37], and VCR [47]. They use a unified neural network to learn general-purpose reasoning skills [17], which is more flexible and scalable than NMN. Most monolithic networks for visual reasoning resort to attention mechanism for multimodal fusion [49, 50, 48, 45, 46, 24, 23, 22, 26, 16]. To realize multi-hop reasoning on complex questions, SAN [43], MAC [17] and MuRel [5] models have been proposed. As the monolithic network is not tied to any pre-defined functionality, it has better generalizability to unseen questions. However, since the reasoning procedure is conducted in the feature space, such models usually lack interpretability, or the ability to capture the compositional language.

GQA Models GQA was introduced in [18] for real-world visual reasoning. Simple monolithic networks [41], MAC network [17], and language-conditioned graph neural networks [16, 12] have been developed for this task. LXMERT [38], a large-scale pre-trained encoder, has also been tested on this dataset. Recently, Neural State Machine (NSM) [19] proposed to first predict a probabilistic scene graph, then perform multi-hop reasoning over the graph for answer prediction. The scene graph serves as a strong prior to the model. Our model is designed to leverage dense visual features extracted from object detection models, thus orthogonal to NSM and can be enhanced with their scene graph generator once it is publicly available. Different from the aforementioned approaches, MMN also performs explicit multi-hop reasoning based on predicted programs to demonstrate inferred reasoning chain.

3. Proposed Approach

The visual reasoning task [18] is formulated as follows: given a question $Q$ grounded in an image $I$, where $Q = \{q_1, \ldots, q_M\}$ with $q_i$ representing the $i$-th word, the goal is to select an answer $a \in A$ from a set of possible answers. During training, we are provided with an additional scene graph $G$ for each image $I$, and a functional program $P$ for each question $Q$. During inference, scene graphs and programs are not provided.

Figure 2 provides an overview of Meta Module Network (MMN), which consists of three components: (i) Program Generator (Sec. 3.1), which generates a functional program from the input question; (ii) Visual Encoder (Sec. 3.2), which consists of self-attention and cross-attention layers on top of an object detection model, transforming an input image into object-level feature vectors; (iii) Meta Module (Sec. 3.3), which can be instantiated to different instance modules to execute the program for answer prediction.

3.1. Program Generator

Similar to other programming languages, we define a set of syntax rules for building valid programs and a set of semantics to determine the functionality of each program. Specifically, we define a set of functions $\mathcal{F}$ with their fixed arity $n_f \in \{1, 2, 3, 4\}$ based on the “semantic string” provided in GQA dataset [18]. The definitions for all the functions are provided in the Appendix. The defined functions can be divided into 10 different function types (e.g., “relate”, “verify”, “filter”, “choose”), and each abstract function type is further implemented with different realizations based on fine-grained functionality (e.g., “verify”: “ver-
ify_attribute”, “verify_geometric”, “verify_relation”, “verify_color”), which take different arguments as inputs.

In total, there are 48 different functions defined in GQA environment, which poses great challenges to the scalability in visual reasoning. The returned values of these functions are List of Objects, Boolean, or String (Object refers to the detected bounding box, and String refers to object name, attributes, relations, etc.) A program $P$ is viewed as a sequence of function calls $f_1, \ldots, f_L$. For example, in Figure 3, $f_2$ is $\text{Relate}([1], \text{beside}, \text{boy})$, the functionality of which is to find a boy who is beside the objects returned by $f_1: \text{Select}($ball$)$. Formally, we call $\text{Relate}$ the “function name”, [1] the “dependency” (previous execution results), and beside, boy the “arguments”. By exploiting the dependency relationship between functions, we build an execution graph, where each node represents a function and each edge denotes an input-output dependency relationship between connected nodes.

In order to generate syntactically plausible programs, we follow [8] and adopt a coarse-to-fine two-stage generation paradigm, as illustrated in Figure 3. We first encode the question as a context vector, and then decode a sketch step by step (the sketch only contains the function name without arguments). Once the sketch is decoded, the arity and types of the decoded functions are determined. For example, after generating “Relate”, there are three arguments following this function with the first argument as the dependency. The sketch is thus expanded as “Relate (#1, #2, #3)”, where “#i” denotes the $i$-th unfilled slot. We then apply a fine-grained generator to fill in the slots of dependencies and arguments for the sketch as a concrete program $P$. During the slot-filling phase, we mask the infeasible tokens at each time step to greatly reduce the search space.

Such a two-stage generation process helps guarantee the plausibility and grammaticality of synthesized programs. For example, if function $\text{Filter}$ is sketched, we know there are two tokens required to complete the function. The first token should be selected from the dependency set ([1], [2], ...), while the second token should be selected from the attribute set (e.g., color, size). With these syntactic constraints to shrink the search space, our program synthesizer can achieve a 98.8% execution accuracy (i.e., returning the same result as the ground truth after execution) compared to execution accuracy of 93% of a standard sequence generation model.

### 3.2. Visual Encoder

The visual encoder is based on a pre-trained object detection model [33, 1] that extracts from image $I$ a set of regional features $R = \{r_i\}_{i=1}^N$, where $r_i \in \mathbb{R}^{D_v}$, $N$ denotes the number of regions of interest, and $D_v$ denotes the feature dimension. Similar to a Transformer block [39], we first use two self-attention networks, $\text{SA}_{q}$ and $\text{SA}_{r}$, to encode the question and the visual features as $\hat{Q} = \text{SA}_{q}(Q, Q; \phi_q)$ and $\hat{R} = \text{SA}_{r}(R, R; \phi_r)$, respectively. $\hat{Q} \in \mathbb{R}^{N \times D}, \hat{R} \in \mathbb{R}^{N \times D}$, and $D$ is the network’s hidden dimension. Based on this, a cross-attention network $\text{CA}$ is applied to use the question as guidance to refine the visual features into $V = \text{CA}(\hat{R}, \hat{Q}; \phi_c) \in \mathbb{R}^{N \times D}$, where $\hat{Q}$ is used as the query vector, and $\phi = \{\phi_q, \phi_r, \phi_c\}$ denotes all the parameters in the visual encoder. The attended visual features $V$ will then be fed as the visual input to the meta module, which is detailed in Sec. 3.3.

### 3.3. Meta Module

As opposed to having a full inventory of task-specific parameterized modules for different functions as in NNMN [3], we design an abstract meta module that can be instantiated into instance modules based on an input function recipe, which is a set of pre-defined key-value pairs specifying the properties of the function. As exemplified in Figure 4, when taking recipe $\text{Function:relate; Geometric:to the left as the input}$, the Recipe Embedder produces a recipe vector to instantiate the abstract meta module into a “geometric relation” module, which specifically searches for target objects that the current object is to the left of. When taking recipe $\text{Function:filter; Type:color; Attribute:pink as input}$, the Embedder will instantiate the meta module into a “filter pink”
module, which specifically looks for the objects with pink color in the input objects.

Two-layered Attention Figure 4 demonstrates the computation flow in meta module, which is built upon two-level multi-head attention network [39]. A Recipe Embedder encodes a function recipe into a real-valued vector \( r_f \in \mathbb{R}^D \). In the first attention layer, \( r_f \) is fed into an attention network \( g_d \) as the query vector to incorporate the output \((o_d:K)\) of dependent modules. The intermediate output \((o_d)\) from this attention layer is further fed into a second attention network \( g_v \) to incorporate the visual representation \( V \) of the image. The final output is denoted as \( g(r_f, o_{1:K}, V) = g_v(g_d(r_f, o_{1:K}), V) \).

Instantiation & Execution The instantiation is accomplished by feeding a function \( f \) to the meta module \( g \), which results in a wrapper function \( g_f(o_{1:K}, V; \psi) \) known as an instance module \( \psi \) denotes the parameters of the module). Each module \( g_f \) outputs \( o(f) \in \mathbb{R}^D \), which acts as the message passed to its neighbor modules. For brevity, we use \( o(f_i) \) to denote the MMN’s output at the \( i \)-th function \( f_i \). The final output \( o(f_L) \) of function \( f_L \) will be fed into a softmax-based classifier for answer prediction. During training, we optimize the parameters \( \psi \) (in meta module) and \( \phi \) (in visual encoder) to maximize the likelihood \( p_{\phi,\psi}(a|P,Q,R) \) on the training data, where \( a \) is the answer, and \( P,Q,R \) are programs, questions and visual features.

3.4. Learning

In order to train the meta module to learn the instantiation process from given function recipes \( i.e., \) how to generate functions), we propose a Teacher-Student framework depicted in Figure 5. First, we define a Symbolic Executor as the “Teacher”, which can take the input function \( f \) and traverse the provided training scene graph to obtain intermediate results \( i.e., \) distribution over the objects on the ground-truth scene graph). The “Teacher” exhibits it as a guideline \( \gamma \) for the “Student” instance module \( g_f \) to follow.

Symbolic Teacher We first execute the program \( P = f_1, \cdots, f_L \) on the ground-truth scene graph \( G \) provided in the training data to obtain all the intermediate execution results. According to the function definition (see Appendix for details), the intermediate results are either of type List of Objects or Boolean. The strategy of representing the results follows: (i) Non-empty List of Objects: use the first element’s vertexes \([x_1, y_1, x_2, y_2] \); (ii) Empty List of Objects: use dummy vertexes \([0, 0, 0, 0] \); (iii) “True” from Boolean: use the vertexes from last step; (iv) “False” from Boolean: use dummy vertexes as in (iii). Therefore, the intermediate results can be unified in the form of quadruples denoted as \( b_i \). To align these quadruple \( b_i \) regions with the regions \( R \) proposed by the object detector from the visual encoder, we compute its overlap against all the regions \( r_j \in R \) as \( a_{i,j} = \frac{\text{Intersect}(b_i, r_j)}{\text{Union}(b_i, r_j)} \). Based on whether there exist any overlaps, we handle the following two cases differently:

1. If \( \sum_j a_{i,j} > 0 \), which means that there exists detected bounding boxes overlapping with the \( b_i \), we normalize \( a_{i,j} \) over \( R \) to obtain a distribution \( \gamma_{i,j} = \frac{a_{i,j}}{\sum_j a_{i,j}} \) and append an extra 0 in the end to obtain \( \gamma_i \in \mathbb{R}^{N+1} \).

2. If \( \sum_j a_{i,j} = 0 \), which means no detected bounding box has overlap with \( b_i \) (or \( b_i = [0, 0, 0, 0] \)), we use the one-hot distribution \( \gamma_i = [0, \cdots, 0, 1] \in \mathbb{R}^{N+1} \) as the distribution. The last bit represents “No Match”.

We call distributions \( \gamma_{i,j} \) the guideline from symbolic teacher. Please refer to the rightmost part of Figure 5 to better understand the computation.

Student Module We propose to demonstrate the guideline distributions \( \gamma_{i,j} \) from the symbolic teacher for student instance modules \( g_f \) to imitate. Formally, we let each instance module \( g_f \) predict its guideline distribution based on its output representation \( o(f) \), denoted as \( \hat{\gamma}_i = \text{softmax}(\text{MLP}(o(f))) \). During training, we enforce the instance module’s prediction \( \hat{\gamma}_i \) to align the guideline distribution \( \gamma_i \) by minimizing their KL divergence \( KL(\hat{\gamma}_i | \gamma_i) \). This side task is aimed to help the meta module learn the mapping from recipe embedding space \( r_f \in \mathbb{R}^D \) to function space \( f \in \mathbb{R} \) like a function factory rather than directly learning independent functions \( f \) itself. Such a “learning to learn” or meta-learning paradigm gives our model the capability to generalize to unseen sub-tasks encoded in the
4. Experiments

We first evaluate the proposed Meta Module Network on CLEVR dataset [20] to preliminarily validate its effectiveness on the synthetic environment. Then we evaluate on the GQA v1.1 dataset [18] and compare it with state-of-the-art methods. As GQA is a more realistic testbed to demonstrate scalability and generalizability of our model, we will focus on it throughout our experiments. To provide visualization of the inferential chains and perform fine-grained error analysis based on that. We design synthesized experiments to quantitatively measure our model’s generalization ability towards unseen functional semantics.

4.1. Preliminary Experiments on CLEVR

The CLEVR dataset [20] consists of rendered images featuring 3D-objects of various shapes, materials, colors, and sizes, coupled with machine-generated compositional multi-step questions that measure performance on an array of challenging reasoning skills. Each question is also associated with a tree-structured functional program that was used to generate it, specifying the reasoning operations that should be performed to compute the answer. We use the standard training set containing 700K questions-program pairs to train our model and parser along with the provided scene graphs. We define a set of function \( F \) with arity of \( n_f \in \{1, 2\} \) provided in the dataset. The definitions of all functions are provided in Appendix, and there are 25 functions in total with similar return type as GQA. We follow NS-VQA [44] to train detectors based on MaskRCNN [13] and detect top 32 bounding boxes ranked by their confidence scores. We use a hidden dimension \( D = 256 \) for both the visual encoder and the meta module.

We report our experimental results on the standard test set in Table 1. We observe that MMN can outperform the standard MMN [15] with more compact representation with meta module and scene-graph-based intermediate supervision. Except for numerical operations like Count and Compare Number, MMN can achieve similar accuracy as the state-of-the-art NS-VQA [44] model. As CLEVR dataset is not the focus of this paper due to its synthetic nature and limited set of semantic functions, we use it only as the preliminary study. Note that we have also attempted to reimplement the NS-VQA approach on GQA, and observe that the accuracy is very low (≈ 30% on test set). This is due to that NS-VQA performs pure symbolic reasoning, thus requiring the scene graph to be accurately generated; while generating scene graphs is challenging in GQA with open-domain real images.

4.2. GQA Experimental Setup

GQA Dataset [18] contains 22M questions over 140K images. This full “all-split” dataset has unbalanced answer
distributions, thus, is further re-sampled into a “balanced-split” with a more balanced answer distribution. The new split consists of 1M questions. Compared with the VQA v2.0 dataset [11], the questions in GQA are designed to require multi-hop reasoning to test the reasoning skills of developed models. Compared with the CLEVR dataset [20], GQA greatly increases the complexity of the semantic structure of questions, leading to a more diverse function set. The real-world images in GQA also bring in a bigger challenge in visual understanding. In GQA, around 94% of questions need multi-hop reasoning, and 51% questions are about the relationships between objects. Following [18], the evaluation metric used in our experiments is accuracy (including binary and open-ended).

The dimensionality of input image features $D_v$ is 2048, extracted from the bottom-up-attention model [1]. For each image, we keep the top 48 bounding boxes ranked by confidence score with the positional information of each bounding box in the form of $[\text{top-left-x, top-left-y, bottom-right-x, bottom-right-y}]$, normalized by the image width and height. Both the meta module and the visual encoder have a hidden dimension $D$ of 512 with 8 heads. GloVe embeddings [30] are used to encode both questions and function keywords with 300 dimensions. The total vocabulary size is 3761, including all the functions, objects, and attributes. For training, we first use the 22M unbalanced “all-split” to bootstrap our model with a batch size of 2048 for 5 epochs, then fine-tune on the “balanced-split” with a batch size of 256. The testdev-balanced split is used for model selection.

### 4.3. GQA Experimental Results

We report our experimental results on the test2019 split (from the public GQA leaderboard) in Table 2. First, we observe significant performance gain from MMN over NMN [3], which demonstrates the effectiveness of the proposed meta module mechanism. Further, we observe that our model outperforms the VQA state-of-the-art monolithic model MCAN [45] by a large margin, which demonstrates the strong compositional property of our module-based approach. Overall, our single model achieves competitive performance (top 2) among published approaches. Notably, we achieve higher performance than LXMERT [38], which is pre-trained on large-scale out-of-domain datasets. The performance gap with NSM [19] is debatable since our model is standalone without relying on well-tuned external scene graph generation model [42, 43, 6].

To verify the contribution of each component in MMN, we perform several ablation studies. (1) w/o Module Supervision vs. w/ Module Supervision. We investigate the influence of module supervision by changing the hyper-parameter $\eta$ from 0 to 2.0 to see how much influence the module supervision has on the model performance. (2) w/o Bootstrap vs. w/ Bootstrap. We investigate the effectiveness of bootstrapping in training to validate whether we could use the large-scale unbalanced split to benefit on the model’s performance.

We further report the ablation results for the validation split in Table 3. From Ablation (1), we observe that without module supervision, our MMN already achieves decent improvement over 6-layered MCAN [45]. Since all the modules have shared parameters, our model has similar parameter size as 1-layered MCAN. The result demonstrates the efficiency of the parameterization in our MMN. By increasing $\eta$ from 0.1 to 0.5, accuracy steadily improves, which reflects the effectiveness of module supervision. Further increasing the value of $\eta$ did not improve the performance empirically. From Ablation (2), we observe that bootstrapping is a critical step for MMN, as it explores more data to better regularize functionalities of reasoning modules. Bootstrap for 4 epochs can yield better performance in our experiments.

### 4.4. Generalization Experimental Results

Similar to Meta Learning [9], we also evaluate whether our meta module has learned the ability to adapt to unseen sub-tasks. To evaluate such generalization ability, we perform additional experiments, where we held out all the training instances containing verify_shape,
Table 4. Analysis of MMN’s generalizability to unseen functions. In NMN, since the unseen function does not have pre-defined module, the performance is close to randomness. In MMN, 0% means without any training instances, 100% means fully-supervision.

<table>
<thead>
<tr>
<th>Function</th>
<th>relate.shape</th>
<th>relate.name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>NMN</td>
<td>0% (MMN)</td>
</tr>
<tr>
<td>Accuracy</td>
<td>50%</td>
<td>61%</td>
</tr>
<tr>
<td></td>
<td>NMN</td>
<td>0% (MMN)</td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td>23%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>filter.location</th>
<th>choose.name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>NMN</td>
<td>0% (MMN)</td>
</tr>
<tr>
<td>Accuracy</td>
<td>50%</td>
<td>77%</td>
</tr>
<tr>
<td></td>
<td>NMN</td>
<td>0% (MMN)</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>62%</td>
</tr>
</tbody>
</table>

To better interpret the model’s behavior, we also perform quantitative analysis to diagnose the errors in the inferential chain. Here, we held out a small validation set to analyze the execution accuracy of different functions. Our model obtains Recall@1 of 59% and Recall@2 of 73%, which indicates that the object selected by the symbolic teacher has 59% chance of being top-1, and 73% chance as the top-2 by the student model, significantly higher than random-guess Recall@1 of 2%, demonstrating the effectiveness of module supervision.

Furthermore, we conduct a detailed analysis of function-wise execution accuracy to understand the limitation of MMN. We found that most erroneous functions are relate and query, having 44% and 60% execution accuracy respectively. These errors are mainly related to scene understanding, which suggests that the scene graph model is critical to surpassing NSM [19] on performance. However, this is out of the scope of this paper and we plan to leave it for future study.

5. Conclusion

In this paper, we propose Meta Module Network that resolves the known challenges of NMN. Our model is built upon a meta module, which can be instantiated into an instance module to perform designated functionalities dynamically. Our approach can significantly outperform baseline methods and achieve comparable performance to state of the art while maintaining strong explainability.


