Curriculum Learning for Compositional Visual Reasoning

Wafa Aissa[1,2], Marin Ferecatu[1] and Michel Crucianu[1]
[1]*Cedric Laboratory, Conservatoire National des Arts et Métiers, Paris, France*
[2]*XXII Group, Paris, France*

Keywords: Compositional Visual Reasoning, Visual Question Answering, Neural Module Networks, Curriculum Learning.

Abstract: Visual Question Answering (VQA) is a complex task requiring large datasets and expensive training. Neural Module Networks (NMN) first translate the question to a reasoning path, then follow that path to analyze the image and provide an answer. We propose an NMN method that relies on predefined cross-modal embeddings to "warm start" learning on the GQA dataset, then focus on Curriculum Learning (CL) as a way to improve training and make a better use of the data. Several difficulty criteria are employed for defining CL methods. We show that by an appropriate selection of the CL method the cost of training and the amount of training data can be greatly reduced, with a limited impact on the final VQA accuracy. Furthermore, we introduce intermediate losses during training and find that this allows to simplify the CL strategy.

1 INTRODUCTION

Visual Question Answering (VQA) consists in answering potentially complex questions regarding the content of images. Datasets like VQA2.0 (Goyal et al., 2017) and GQA (Hudson and Manning, 2019) were put forward in support of this task. These datasets are very large, leading to expensive training. However, since they are built from collected real images with (possibly assisted) human labeling, they inevitably contain many biases. Integrated approaches like (Xiong et al., 2022; Wang et al., 2021) have the highest overall accuracies on these databases but are prone to taking bias-promoted "shortcuts", as shown *e.g.* by their lower performance on out-of-distribution data (Kervadec et al., 2021). Furthermore, integrated approaches lack transparency in the reasoning process, even though some limited explanations can be obtained by following the flow of attention.

Alternatively, Neural Module Networks (NMN) were introduced (Hu et al., 2017) with the aim to make the reasoning explicit. They were quite successful for visual reasoning on synthetic image datasets like CLEVR, where word grounding is comparatively easy and there is significant control over scene composition. NMNs are nevertheless hard to train on real images where grounding is difficult, attributes are more diverse and data biases are hard to control. To make learning more effective and less expensive, we rely on NMN but employ predefined cross-modal embeddings to "warm start" the training process on GQA, then explore Curriculum Learning to improve learning of such a complex task and reduce both the cost and the amount of data required.

Curriculum Learning (CL) (Elman, 1993; Soviany et al., 2022; Wang et al., 2022) consists in learning the easier parts of the task first, rather than the entire task at once. However, adequate difficulty criteria are not easy to define. We show that by an appropriate selection of these criteria for VQA, the cost of training can be significantly reduced and less training data is required to reach a comparable level of accuracy.

To summarize, the contributions of our work are three fold:

- First, we employ text and image object embeddings produced by a cross-modal transformer, with the goal of aligning multi-modal features to reinforce joint data patterns and thus help the learning process to achieve results faster.
- Second, we propose several Curriculum Learning strategies to reduce both the training cost and the amount of data required for learning complex reasoning tasks.
- Third, we define and employ intermediate module losses (one per module) during training, using ground-truth labels generated from image graphs. The aim is to stabilize the learning and help the

Aissa, W., Ferecatu, M. and Crucianu, M.
Curriculum Learning for Compositional Visual Reasoning.
DOI: 10.5220/0011895400003417
In *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - Volume 5: VISAPP*, pages 888-897
ISBN: 978-989-758-634-7; ISSN: 2184-4321

modules converge to the expected behavior defined by the modules' ground-truth and controlled by the local loss.

The paper is organized as follows: the next section situates our proposals in the context of existing work on VQA and Curriculum learning, Sec. 3 describes the modular framework we employ and our use of cross-modal features. Then in Sec. 4 we define and motivate the CL strategies we propose. Evaluation results are presented and discussed in Sec. 5.

2 RELATED WORK

We first review some recent work on visual reasoning for VQA. We then turn to the use of Curriculum learning for complex tasks, more specifically for VQA.

2.1 Visual Question Answering

VQA is usually addressed with either integrated cross-modal frameworks or compositional neural module networks.

Cross-Modality Transformers. Transformer networks (Vaswani et al., 2017) have been widely applied to multiple language and vision tasks, and they have been recently adapted for reasoning problems such as VQA. Models like ViLBERT (Lu et al., 2019), VisualBERT (Li et al., 2019b) and LXMERT (Tan and Bansal, 2019) showcased good performances on the VQA datasets VQA2.0 (Goyal et al., 2017) and GQA (Hudson and Manning, 2019). These frameworks start by extracting the text and image features: word embeddings are obtained via a pretrained BERT (Devlin et al., 2019) model, while Faster R-CNN (Ren et al., 2015) produces image region bounding boxes and corresponding visual features. Then, a cross-attention mechanism allows to align word embeddings and image features after training on a wide range of multi-modal tasks. One downside of integrated visual reasoning models is their lack of interpretability. Another drawback is their tendency to make "shortcuts" in reasoning, by learning the bias in the data as evidenced by their limited performance on the out-of-distribution data in GQA-OOD (Kervadec et al., 2021). However, an effective cross-modal feature encoder can be obtained by discarding the final classification component from an integrated model. We employ here input features generated by an off-the-shelf large-scale cross-modal transformer encoder.

Neural Module Networks (NMN). To make the reasoning process more transparent and human-like, compositional NMNs (Hu et al., 2017; Li et al., 2019a) perform multi-hop reasoning by decomposing a complex reasoning task into several easier subtasks. An NMN consists of a generator and an executor. The generator maps a question to a sequence of reasoning instructions (called a *program*). The executor assigns each sub-task from this program to a neural module and passes the results to the next modules. In (Chen et al., 2021) a meta-learning approach is employed in the NMN framework to improve the scalability and generalization of the resulting model. The generator decodes the question into a program whose sub-tasks are used to instantiate a meta module. The image features are extracted by a visual encoder implemented as a transformer network and a cross-attention layer mixes word embeddings and image features. While the combination of a generator and an executor in NMNs appears more complex than an integrated model, the "hardwired" reasoning process of an NMN is inherently transparent and has the potential to avoid part of the reasoning "shortcuts" caused by data bias. Interestingly, it was shown in (Kervadec et al., 2021) that by using the programs resulting from questions as additional supervision for the LXMERT integrated model allows to reduce sample complexity and improve performance on GQA-OOD. In our work, we aim to take advantage of both the transparency of NMN architectures and the quality of transformer-encoded representations by implementing a composable NMN over multimodal transformer vision and language features.

2.2 Curriculum Learning

Curriculum learning was introduced in (Elman, 1993) where the author shows that successful learning may depend on "starting small" by first learning a simple grammar with a recurrent network and then gradually learning more complex tasks such as relative clauses, number agreement, etc. CL was later applied to various machine learning tasks and recently adapted to textual question answering (QA) in (Liu et al., 2018). The authors use a sampling function that gives higher selection weights to simple QA pairs and then, as the training advances, it selects more complex QA pairs. A *term frequency* selector and a *grammar* selector assess the difficulty of the training examples. In (Sachan and Xing, 2016) CL is reframed as a self-paced learning (SPL) algorithm and the question loss is taken as the measure of difficulty. The authors implement several heuristics reminding of active learning in order to improve SPL performance.

Curriculum Learning for VQA. The definition of relevant difficulty criteria for VQA is challenging and this may explain why there is little work on the use of CL for VQA. The recent work in (Askarian et al., 2021) applies CL in a modular VQA context to the synthetic CLEVR dataset (Johnson et al., 2017a). The base model is from (Johnson et al., 2017b), with an LSTM generator and generic residual blocks for the executor modules. The experiments were conducted on the executor alone, using as input the ground-truth programs directly. Several difficulty criteria were evaluated, including program length, answer hierarchy, and question loss. The results demonstrated that CL with a question loss difficulty criterion has a positive impact in a low data setting. However, the study in (Askarian et al., 2021) was focused on the CLEVR dataset (Johnson et al., 2017a) consisting of synthetic images of simple 3D objects, with a limited number of classes or attributes and reliable object detection. In our work, we employ the GQA dataset that is based on real-world images with many classes and several names for some of them, as well as more complex relations and more challenging object detection. We thus have to completely redefine the candidate CL strategies.

3 MODULAR VQA FRAMEWORK

Our model takes as input a triplet composed of an image, a question and a program, and predicts an answer. We start by extracting aligned language and vision features for both the image and the question using a state-of-the-art cross-modal transformer. Then the program, which is a sequence of modules, is used to build the neural modules network that is executed on the image to answer the question (see Fig. 1). In the next subsections, we present the feature extraction process and describe the program executor.

Cross-Modal Features. Compositional visual reasoning aims to perform logical and/or geometrical inferences involving several related objects in a complex scene. To achieve this, the reasoning modules are conditioned by text arguments. The visual and textual representations are vital to the reasoning process, therefore having good bounding box features and question text embeddings is crucial. To extract cross-modal language and vision representations we rely on LXMERT (Tan and Bansal, 2019), a transformer model pretrained on multiple multi-modal tasks such as Masked Cross-Modality language models, masked object predictions, Cross-Modality Matching, and VQA. LXMERT showcases high accuracy on the training tasks, so we employ it as a feature extractor. It is worth mentioning that we only use the cross-modality encoder representations and discard the answer classification component. More precisely, we freeze LXMERT weights and we pass the image I through the object-relationship encoder and the question Q through the language encoder. Then, the Cross-Modality Encoder aligns the representations to finally output the object bounding box features v_j of each object o_j in the image I and the embedding h_i of each word q_i in the question Q.

Neural Modules. Our compositional reasoning model allows to perform complex reasoning tasks by decomposing them into easier sub-tasks. These sub-tasks are inspired by the human generic reasoning skills such as object detection, attribute identification, object relation recognition, object comparison, etc. We designed a library of modules where each module is responsible for performing a reasoning sub-task. The modules were designed to be intuitive and easily interpretable, each of them being implemented by a series of basic algorithmic operations such as dot products and MLPs. Modules can be categorized into three different groups based on their output type: attention, boolean and answer modules. For instance, an attention module such as `Select` is responsible of detecting an object bounding box by rendering an attention vector over the object bounding boxes contained in the image. Boolean modules such as `And` or `Or` make logical inferences and answer modules such as `QueryName` give a probability distribution over the answer vocabulary. Table 1 shows an example from each module category. An exhaustive module list with definitions is provided in the appendix.

Modular Network Instantiation. A program consists of a sequence of modules implemented as neural networks, as illustrated in Table 1. Each program is instantiated as a larger NMN following the sequence of program modules, where each module has dependencies d_m to get information from the previous one, and arguments a_m to condition its behavior. For example, the `FilterAttribute` module depends on the output of the `Select` module: it shifts the attention on the selected objects corresponding to the input text argument. The program executor is responsible of managing module dependencies by using a memory buffer to save the outputs that serve as inputs for the next modules. The design insures that a module can have at most two dependencies.

The generic modules require a textual argument $a_{m,i}$ to determine which facet of the module to use, for example, the `FilterAttribute` module can be

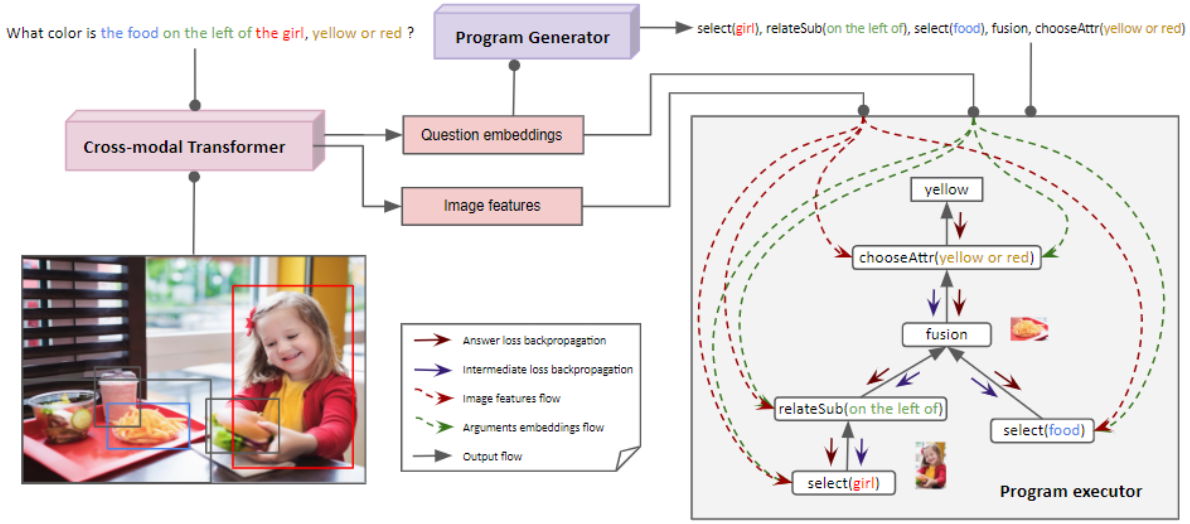

Figure 1: The proposed modular VQA framework: The (question, image) pair is used by a transformer model to generate aligned cross-modal embeddings for words and objects. These are used by a Program Generator module to produce a program (represented as a sequence of sub-task modules), which will be then applied by the Program Executor module to the image to answer the question. The proposed work focuses on improving the Program Executor by using several curriculum learning (CL) strategies.

Table 1: Sample module definitions. S: softmax, σ: sigmoid, r: RELU, $\mathbf{W}_i$: weight matrix, $\mathbf{a}$: attention vector (36×1), $\mathbf{V}$: visual features (768×36), $\mathbf{t}$: text features (768×1), $\odot$: Hadamard product.

Name	Dependencies	Output	Definition
Select	–	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t})$ $\mathbf{Y} = r(\mathbf{W}\mathbf{V})$ $\mathbf{o} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x}))$
RelateSub	[**a**]	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t})$ $\mathbf{Y} = r(\mathbf{W}\mathbf{V})$ $\mathbf{z} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x}))$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x}\odot\mathbf{y}\odot\mathbf{z}))$
VerifyAttr	[**a**]	boolean	$\mathbf{x} = r(\mathbf{W}\mathbf{t})$ $\mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a})$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x}\odot\mathbf{y}))$
And	[$\mathbf{b}_1$,$\mathbf{b}_2$]	boolean	$\mathbf{o} = \mathbf{b}_1 \times \mathbf{b}_2$
ChooseAttr	[**a**]	answer	$\mathbf{x} = r(\mathbf{W}\mathbf{t})$ $\mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a})$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x}\odot\mathbf{y}))$
QueryName	[**a**]	answer	$\mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}))$ $\mathbf{o} = S(\mathbf{W}\mathbf{y})$

called for multiple attribute categories such as color and size. For instance, to filter the red objects we use the argument word "red" and input its cross-modal embedding to the module.

Module Execution Supervision. The executor network takes as input an image in the form of a list of object bounding boxes and their LXMERT embeddings. Every executor network ends with an answer module, the output of which is compared to the ground-truth to compute the output loss. To help modules converge faster to their expected behavior we also add the loss of each module to the output loss. The ground truth for each module is extracted from the image graphs given by the GQA dataset. Each image graph has k assigned bounding boxes $bbox^*_{1...k}$ with their names, coordinates, attributes, and relations. Since we have two different intermediate module types, we define an intermediate loss for the attention modules and another for the boolean modules.

4 CURRICULUM LEARNING FOR VQA

Our aim is to study CL for VQA and find a CL method that allows to significantly reduce training cost while making a better use of the data.

A CL method is usually defined by a difficulty criterion, a scheduler and a sampling function. The difficulty criterion allows to characterize the samples: training starts with the easiest samples, then progressively moves toward more difficult samples. Question loss was employed with some success as a difficulty criterion in (Sachan and Xing, 2016) for QA and in (Askarian et al., 2021) for VQA. However, computing question loss requires a first training iteration over all the training data. To make our own difficulty criteria, we assume that reasoning about a single object and its properties is simpler than examining the relations between several objects and comparing their attributes. The number of different objects in

the question should then be a good indication of the complexity of reasoning and thus a relevant *a priori* difficulty criterion for CL. Program length is another potentially relevant criterion that takes into account the flow of gradient in module networks corresponding to longer programs and is related to the previous criterion (more objects in the question require longer programs). Note that several criteria can be combined to define the increasing difficulty of the training samples in CL. When employing the number of objects as a primary criterion, we also evaluate its refinement based on program length: for each number of objects in the question, we start with the short programs, then continue with the medium length ones and end with the long programs.

The scheduler in CL decides when the curriculum should be updated. A simple solution is to employ a fixed sample size for each difficulty level. The evolution of the loss can be employed to adjust this size.

The sampling function allows to modulate the selection of training samples within each difficulty level and works by assigning weights to all the examples. A relevant choice is to balance the occurrence probabilities of the different types of answer modules. Another criterion, used in boosting, is to privilege programs that lead to higher errors.

When the curriculum is updated, the new training sample has a higher level of difficulty than the previous ones. This does not mean that it subsumes the past samples and this is particularly true for the complex task of VQA. To avoid catastrophic forgetting, we add to the current sample (corresponding to the current difficulty level) a random selection from the past samples.

5 EXPERIMENTS

5.1 Experimental Setup

GQA Dataset. GQA (Hudson and Manning, 2019) features over 18M compositional questions and 113K real-world images. The reasoning steps of the questions are represented by functional programs. The questions and programs are generated by a question engine from the corresponding image graph. The image graphs are composed from ground-truth object bounding boxes together with their names, attributes, and relations. GQA is based on the Visual Genome dataset.

The GQA dataset has two versions: a balanced version (with a uniform distribution over the answers) and an unbalanced one. For the unbalanced version, the `train-all` split has over 14M examples and the `testdev-all` has 172,174 examples, while the balanced version has over 943,000 examples for `train`, 132,062 for `val` and a `testdev` of 12,578 examples. To have a larger number of examples available for CL, we use the unbalanced GQA dataset. The experiments trained on the balanced dataset use the union of `train` and `val` as done in (Tan and Bansal, 2019), this combination also allows us to get over 1M examples when training on the balanced set. We follow the recommendations of the dataset authors by evaluating the performance on the `test-dev` split instead of the `val` split when using the object-based features because they were trained on some images from the `val` set (Anderson et al., 2018).

In the GQA dataset, each question/image pair in the `train`, `val`, and `testdev` sets is associated with a functional program. The programs use 124 distinct modules, some of which only correspond to very few questions. We consider that modules should only differ when they correspond to operations that are different. We thus group specific modules into more general ones. For example, modules like `ChooseHealthier` and `ChooseOlder` are grouped in a `ChooseAttribute` module. This results in only 32 modules, the list of which is presented in the appendix.

Metrics. Several metrics are employed to compare CL and standard learning. Since our focus is on reducing the cost of training, we measure the total number of example presentations during training (Comp. cost). We also mention the maximal number of different examples seen during training (# examples), as different methods can make use of larger or smaller portions of the training set. While we focus on the cost of training, we nevertheless want to reach an accuracy that is close to the one obtained by standard learning, so we also report the accuracy of the predicted answers.

5.2 Evaluated Methods

When describing the different performed experiments, we use the following notations, which correspond to different algorithmic choices:

– **Unbalanced:** We train on all the examples from the unbalanced GQA train split, we use the traditional random batch training strategy and the model sees all the data examples in every epoch.

– **Balanced:** We train on the balanced version of the GQA dataset. At every epoch, the model is trained on all the balanced dataset examples.

– **Random:** Instead of training on all the dataset examples, only 1M random examples from the unbalanced dataset are presented to the model at every iteration.

– **CL:** The model is trained using Curriculum Learning and the filtering is driven by the number of objects in the programs. At every CL iteration the model sees 1M examples filtered from the unbalanced dataset by the curriculum sampler. The training needs 4 CL iterations to be complete, where each iteration has an increased difficulty given by the number of objects of its programs (ranging from 1 to 4).

– **Length (L):** The curriculum sampler filters the programs by their lengths for every number of objects, a CL-iteration is defined by a number of objects and a program length (short or medium or long).

– **Weights (W):** We use several sampling weights for the filtered programs: 'uniform' indicates that the sampling is uniform, so the sampling with replacement results in a sample that is uniformly distributed over all the dataset. To make the answer modules distribution of the resulting sample more uniform we use the 'answer module' weighting (denoted by W.a); this balances the answer modules appearances in the result sample so that the model equally sees all the defined answer modules. The 'modules loss' weighting (denoted by W.b) indicates that the example's weight is proportional to the sum of the average losses of the modules composing its program, to focus the model on hard examples.

– **Pretrain (P):** The model's parameters are initialized from a model trained using the **Random** variant described above.

– **Repeat (R):** We repeat the same CL-iteration twice.

5.3 Implementation Details

We perform the experiments using the pre-processed GQA dataset programs and we focus on investigating the effect of several CL policies on the Program Executor module. Although our system also uses a transformer model as a generator to translate the question to its corresponding program, its task is relatively easy compared to the executor training and, similar to previous works (Li et al., 2019a; Chen et al., 2021), we achieve near perfect translation results on the `testdev-all` set.

We employ LXMERT as a feature extractor and freeze its weights. LXMERT image inputs are the object bounding boxes provided by a Faster R-CNN object detection model (Anderson et al., 2018), where the number of bounding boxes per image is fixed to 36. We feed the questions and their corresponding object bounding boxes to LXMERT; the encoding yields 36 object features and the question word embeddings for each question/image pair. The extracted features and embeddings have the size of the LXMERT hidden size, *i.e.* 768.

During CL, we fix the sample size to 1M examples per CL iteration. Starting from the second CL iteration, 20% of the training sample is sampled from the examples seen in the previous iteration. Concerning the number of training examples, it is worth mentioning that we sample the training examples with replacement. Therefore, the number of *distinct* examples seen by the executor is lower than the sample size. To reduce the complexity of our proposed model, we allow weight sharing between compatible modules. For example, the `relateObj` and `relateSub` modules have similar structures and functionalities, both have visual and textual layers to project bounding box features and word embeddings in a multi-modal reasoning space (they also have an output layer to classify the answer). These two modules also have similar functionalities, *i.e.* they both relate to an object given an anchor object and a relation but they have a different relation direction. Therefore, we share the weights between the visual layers and the textual layers of both modules respectively, but without sharing the output layers to guarantee transitional direction differences. For training all the models we use SGD with a learning rate of 0.1 and a batch size of 1024.

5.4 Evaluation Results

This section presents an analysis of the performance and the cost of our modular VQA framework with multiple CL training strategies, followed by a comparison with models not using CL to show the effectiveness of our proposed training approach.

Comparison of CL Methods. We start by a comparative analysis of the proposed CL strategies as described in Sec. 4. Table 2 reports the performance of our model based on the different CL configurations detailed in Sec. 5.2. The goal of CL is to make the training more effective and to achieve the highest accuracy while training for fewer iterations. Therefore, for each model are shown the number of iterations and training examples required to reach the highest accuracy.

From the results, it is clear that the 'answer'

Table 2: Results on `testdev-all` for several CL strategies.

Model	CL configuration			Iterations	Number of examples ($\leq$)	Accuracy
	weighting	pretraining	iterations/level			
CL+W.a	answer	–	1	4	4 M	0.642
CL+W.b	losses	–	1	4	4 M	0.635
CL+W.a+P	answer	2 iterations	1	$[2]+3$	5 M	0.670
CL+W.a+P+R	answer	2 iterations	2	$[2]+5$	7 M	**0.681**

weighting is the most effective weighting function. One can see this as a balancing of the answer modules presence over the training sample. The CL+W.a model (using the 'answer' weighting) achieves higher accuracy results than the CL+W.b model (with the 'losses' weighting), both reaching their top respective accuracies after 4 training iterations only. The 'answer' weighting also yields better accuracy than the 'uniform' weighting after the same number of training iterations. This is shown by comparing CL+L and CL+L+W.a in Table 3. Moreover, the accuracy of CL+L+W.a continues to increase after the 11th iteration to achieve its top at iteration 12. The superior performance of the 'answer' weighting function in two different comparable settings makes us select this weighting for the rest of the experiments.

The refinement of the CL difficulty (or hardness) measure using the number of question objects (Length-CL difficulty measure) increases the CL+W.a top accuracy by 1%, see the CL+L+W.a line in Table 3. However, this improvement has a significant cost, as CL+L+W.a requires 12 training iterations (12M examples) unlike CL+W.a which only needs 4 iterations (4M examples). This reinforces the idea that with a more refined difficulty measure the model has more time to adjust to difficult examples, and its accuracy gradually increases to achieve a better top accuracy in a CL setting. But training on 12M examples is expensive since the overall dataset size is 14M examples. We thus decided to explore different options to obtain comparable results at a lower cost.

A promising finding was that pretraining the models for a few iterations with a randomly sampled 1M examples each leads to an accuracy increase of over 1.5%, as shown by the CL+W.a+P model which was pretrained for only 2 iterations. This "warms up" the model to the modular aspect of our VQA framework, allowing it to be more general and effective before starting the CL. An interesting finding was that the model reached peak accuracy before iterating over the full CL configuration. The accuracy drop resulting after the 4th iteration may be explained by model overfitting on the questions with 4 objects. Indeed, in the GQA dataset these questions have a substantially unbalanced answer distribution.

Table 3: Results on `testdev-all` with program length as a refinement for the CL difficulty measure. Computation cost is the number of seen examples per iteration times the number of iterations.

Model	Comp. cost	# examples	Accuracy
CL+L	11	11 M	0.650
CL+L+W.a	12	12 M	0.655

A further finding is that repeating the same CL-iteration twice (as in CL+W.a+P+R) improves the top accuracy results by 1.1%, while only moderately increasing the number of iterations. This can be explained by the fact that doubling the number of training iterations helps the model better understand the structure of training data without augmenting the training data size. As detailed in Sec. 5.3, when sampling with replacement we obtain a number of distinct examples that is slightly lower than the sample size, therefore the reported number of examples (# examples) is an upper bound of the # examples actually employed.

As a general conclusion, we consider the CL+W.a+P+R model as the best modular VQA model that scores the best accuracy of 68.1% after 7 training iterations using less than 7M distinct examples, *i.e.* less than half of the training data.

Impact of CL. We perform several experiments to assess the impact of the CL on our compositional visual reasoning framework. We do this by training our model without CL (Unbalanced, Balanced, and Random), then comparing the accuracy performance and the experiment cost in terms of computation cost and training data examples. In Table 4 we report the accuracy and cost results of the conducted experiments and compare them to the performance of our best CL model **CL+W.a+P+R**.

The Unbalanced model (trained on the entire unbalanced training set of 14M) achieves the highest accuracy value of 70.2%. This model also has the highest training cost among the evaluated models.

The Balanced model, trained on the balanced dataset for a large number of epochs, achieves lower results than the Unbalanced model. This is partly due to the fact that the balancing reduces not only the

number of questions in the dataset, but also the diversity of the programs. Also, to the use of the unbalanced `testdev-all` for evaluation.

By comparing our best CL model (**CL+W.a+P+R**) to the models trained without CL (no-CL), we find very significant gains in terms of computational cost, *e.g.* an 18-fold reduction compared to the top contender, the model trained on the Unbalanced dataset. The price to pay—a drop of only 2% in accuracy—appears reasonable. The Random model, trained on randomly sampled 12M examples, performs almost as well as the Unbalanced model, an expected result since both models use a similar amount of distinct training examples (12M vs 14M). The Unbalanced model requires an almost 9 times more expensive training than Random, but the improvement in accuracy (70.2 % vs. 69.4%) hardly justifies it. However, the proposed CL model has an almost 2 times lower computational cost than Random, confirming the superiority of curriculum learning in this type of application.

Table 4: Comparaison of our CL model (CL+W.a+P+R) with no-CL models (Unbalanced, Balanced, and Random) on the `testdev-all` set.

Model	Comp. cost	# examples	Accuracy
Unbalanced	9×14 M	14 M	**0.702**
Balanced	50×1.4 M	1.4 M	0.678
Random	12×1 M	≤ 12 M	0.694
CL+W.a+P+R	$\mathbf{7} \times \mathbf{1}$ M	$< \mathbf{7}$ M	0.681

6 CONCLUSION

In this work we present several Curriculum Learning (CL) strategies within a Neural Module Network (NMN) framework for Visual Question Answering (VQA). Our visual reasoning approach leverages a cross-modal Transformer encoder to extract aligned question/image features along with question programs to perform multi-step reasoning over the image and predict an answer. Our model employs an NMN architecture composed of multiple neural modules, each capable of performing a reasoning subtask. We compare several CL strategies for VQA. Our model is evaluated on the GQA dataset and shows very interesting results in terms of computational cost reduction. To drive the CL strategy, we introduce a difficulty measure based on the number of objects in the question and we achieve close accuracy results by training on a judiciously sampled 50% of the training data, compared to an NMN model trained without CL on the entire training set.

ACKNOWLEDGEMENTS

We thank Souheil Hanoune for his insightful comments. This work was partly supported by the French Cifre fellowship 2018/1601 granted by ANRT, and by XXII Group.

REFERENCES

Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., and Zhang, L. (2018). Bottom-up and top-down attention for image captioning and visual question answering. In *CVPR*.

Askarian, N., Abbasnejad, E., Zukerman, I., Buntine, W., and Haffari, G. (2021). Curriculum learning effectively improves low data VQA. In Rahimi, A., Lane, W., and Zuccon, G., editors, *Australasian Language Technology Association Workshop (ALTA) 2021*, pages 22–33. ACL.

Chen, W., Gan, Z., Li, L., Cheng, Y., Wang, W. Y., and Liu, J. (2021). Meta module network for compositional visual reasoning. In *WACV*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. 2019 Conf. ACL*, pages 4171–4186.

Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71–99.

Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D. (2017). Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *CVPR*.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. (2017). Learning to reason: End-to-end module networks for visual question answering. In *ICCV*.

Hudson, D. A. and Manning, C. D. (2019). GQA: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. B. (2017a). CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*.

Johnson, J., Hariharan, B., Van Der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., and Girshick, R. (2017b). Inferring and executing programs for visual reasoning. In *ICCV*.

Kervadec, C., Wolf, C., Antipov, G., Baccouche, M., and Nadri, M. (2021). Supervising the transfer of reasoning patterns in VQA. In *NeurIPS*.

Li, G., Wang, X., and Zhu, W. (2019a). Perceptual visual reasoning with knowledge propagation. In *ACM MM*, MM '19, page 530–538, New York, NY, USA. ACM.

Li, L. H., Yatskar, M., Yin, D., Hsieh, C., and Chang, K. (2019b). VisualBERT: A simple and performant baseline for vision and language. *CoRR*, abs/1908.03557.

Liu, C., He, S., Liu, K., and Zhao, J. (2018). Curriculum learning for natural answer generation. In *IJCAI*. AAAI Press.

Lu, J., Batra, D., Parikh, D., and Lee, S. (2019). ViLBERT: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *NeurIPS*.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*.

Sachan, M. and Xing, E. (2016). Easy questions first? A case study on curriculum learning for question answering. In *Proc. 54th Annual Meeting of the ACL*, pages 453–463, Berlin, Germany. ACL.

Soviany, P., Ionescu, R. T., Rota, P., and Sebe, N. (2022). Curriculum learning: A survey. *Int. J. Comput. Vis.*, 130(6):1526–1565.

Tan, H. and Bansal, M. (2019). LXMERT: Learning cross-modality encoder representations from transformers. In Inui, K., Jiang, J., Ng, V., and Wan, X., editors, *EMNLP/IJCNLP (1)*, pages 5099–5110. ACL.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NeurIPS*.

Wang, J., Ji, Y., Sun, J., Yang, Y., and Sakai, T. (2021). MIRTT: learning multimodal interaction representations from trilinear transformers for visual question answering. In Moens, M., Huang, X., Specia, L., and Yih, S. W., editors, *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 2280–2292. Association for Computational Linguistics.

Wang, X., Chen, Y., and Zhu, W. (2022). A survey on curriculum learning. *TPAMI*, 44(9):4555–4576.

Xiong, P., Shen, Y., and Jin, H. (2022). MGA-VQA: multi-granularity alignment for visual question answering. *CoRR*, abs/2201.10656.

APPENDIX

In this appendix, for reference purposes, we present the exhaustive list of modules together with their dependencies, types, and definitions (see Table 5). The column 'output' represents the module type: Attention, Boolean, or Answer.

Attention modules produce an attention vector **a** where each element represents the relevance of the attended image object. Boolean modules make logical inferences and output a scalar representing the probability of the outcome. Answer modules give a probability distribution over the answer classes.

As mentioned in Section 5.3, we used a weight sharing technique to reduce the number of the model parameters; this also allows the shared layers to have a better defined behavior and to be updated based on a larger number of training examples. The overall principle is that a transfer is made only between some of the textual and visual layers, each module having a distinct output layer to guarantee its fine-tuning to the module's sub-task.

To decide what layers from which modules will share parameters, we assess the similarity between the modules by analyzing their functional and architectural properties. The former is derived from the module reasoning sub-task and the latter is derived from the module layer architectures. We cannot exhaustively describe here all the performed analysis, but in the following, we exemplify our strategy by comparing some of the modules and explaining their inherent similarities and differences. The `Select` module detects a relevant bounding box (given the name of an object) and the `FilterAttr` detects a relevant bounding box given an attribute. Functionally, they both solve a detection problem but have different textual argument semantics. Architecturally, they both have the same layer structure: a textual layer, a visual layer, and an output layer. We decide to share the visual layer between these two modules but use different textual layers to respect the semantic differences between the textual arguments.

However, `FilterAttr` can share its textual layer with other modules having an attribute as a textual argument (`VerifyAttr`, `FilterNot`).

The `Same` and `Different` boolean modules assess whether or not two selected objects share the same characteristic (provided by the textual argument). The probability p of two objects being similar is the opposite of them being different. Therefore, they share the same layers including the output layer and we use the relation $p(\text{Different}) = 1 - p(\text{Same})$ to differentiate them.

The object relations modules such as `RelateSub` and `RelateObj` have similar functionalities and neural structures. They share their visual layers to get a common scene representation and they share the textual layer due to the semantic similarity of their arguments (a relation).

Table 5: Exhaustive module definitions. S: softmax, σ: sigmoid, r: RELU, $\mathbf{W}$: weight matrix, $\mathbf{a}$: attention vector (36×1), $\mathbf{b}$: boolean scalar, $\mathbf{V}$: visual features (768×36), $\mathbf{t}$: text features (768×1), $\odot$: Hadamard product, $[a\|\mathrm{b}]$: concatenation, *min*: element-wise minimum.

Name	Dependencies	Output	Definition
Select	–	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{Y} = r(\mathbf{W}\mathbf{V}),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x}))$
FilterAttr	[**a**]	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{Y} = r(\mathbf{W}\mathbf{V}), \mathbf{z} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x})),$ $\mathbf{o} = \min(\mathbf{a}, \mathbf{z})$
FilterNot	[**a**]	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{Y} = r(\mathbf{W}\mathbf{V}), \mathbf{z} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x})),$ $\mathbf{o} = \min(\mathbf{a}, 1 - \mathbf{z})$
FilterPos	[**a**]	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{Y} = r(\mathbf{W}\mathbf{V}), \mathbf{z} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x})),$ $\mathbf{o} = \min(\mathbf{a}, \mathbf{z})$
RelateSub	[**a**]	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{Y} = r(\mathbf{W}\mathbf{V}), \mathbf{z} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
RelateObj	[**a**]	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{Y} = r(\mathbf{W}\mathbf{V}), \mathbf{z} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
RelateAttr	[**a**]	attention	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{Y} = r(\mathbf{W}\mathbf{V}), \mathbf{z} = S(\mathbf{W}(\mathbf{Y}^T\mathbf{x})),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
Fusion	[$\mathbf{a}_1$,$\mathbf{a}_2$]	attention	$\mathbf{o} = \min(\mathbf{a}_1, \mathbf{a}_2)$
And	[$\mathbf{b}_1$,$\mathbf{b}_2$]	boolean	$\mathbf{o} = \mathbf{b}_1 \times \mathbf{b}_2$
Or	[$\mathbf{b}_1$,$\mathbf{b}_2$]	boolean	$\mathbf{o} = \mathbf{b}_1 + \mathbf{b}_2 - \mathbf{b}_1 \times \mathbf{b}_2$
Same	[$\mathbf{a}_1$,$\mathbf{a}_2$]	boolean	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_1)), \mathbf{z} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_2)),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
SameAll	[**a**]	boolean	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a})),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
Different	[$\mathbf{a}_1$,$\mathbf{a}_2$]	boolean	$\mathbf{o} = 1 - same(\mathbf{a}_1, \mathbf{a}_2)$
DifferentAll	[**a**]	boolean	$\mathbf{o} = 1 - same(\mathbf{a})$
Exist	[**a**]	boolean	$\mathbf{o} = \sigma(\mathbf{W}([\mathbf{a} \,\|\| \max(\mathbf{a}) \,\|\| \min(\mathbf{a}) \,\|\| mean(\mathbf{a})]))$
VerifyRelSub	[$\mathbf{a}_1$,$\mathbf{a}_2$]	boolean	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_1)), \mathbf{z} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_2)),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
VerifyRelObj	[$\mathbf{a}_1$,$\mathbf{a}_2$]	boolean	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_1)), \mathbf{z} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_2)),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
VerifyAttr	[**a**]	boolean	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
VerifyPos	[**a**]	boolean	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}),$ $\mathbf{o} = \sigma(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
ChooseName	[**a**]	answer	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
ChooseAttr	[**a**]	answer	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
Compare	[$\mathbf{a}_1$,$\mathbf{a}_2$]	answer	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_1), \mathbf{z} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_2),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
ChoosePos	[**a**]	answer	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
ChooseRel	[$\mathbf{a}_1$,$\mathbf{a}_2$]	answer	$\mathbf{x} = r(\mathbf{W}\mathbf{t}), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_1), \mathbf{z} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_2),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y} \odot \mathbf{z}))$
Common	[$\mathbf{a}_1$,$\mathbf{a}_2$]	answer	$\mathbf{x} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_1), \mathbf{y} = r(\mathbf{W}(\mathbf{V}\mathbf{a}_2),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x} \odot \mathbf{y}))$
QueryName	[**a**]	answer	$\mathbf{x} = r(\mathbf{W}(\mathbf{V}\mathbf{a}), \mathbf{o} = S(\mathbf{W}(\mathbf{x}))$
QueryAttr	[**a**]	answer	$\mathbf{x} = r(\mathbf{W}(\mathbf{V}\mathbf{a}),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x}))$
QueryPos	[**a**]	answer	$\mathbf{x} = r(\mathbf{W}(\mathbf{V}\mathbf{a}),$ $\mathbf{o} = S(\mathbf{W}(\mathbf{x}))$
AnswerLogic	[**b**]	answer	$\mathbf{o}_{\mathbf{yes}} = \mathbf{b}, \mathbf{o}_{\mathbf{no}} = \mathbf{1} - \mathbf{b}$