Plan today

1. Routing Networks (ICLR 2018)
    1. Slides 2 through 16
2. Inferring and Executing Programs for Visual Reasoning (ICCV 2017)
    1. Slides 17 through 25
3. Report on progress for current work (Aiming for CVPR 2020)
    1. Slides 26 through 33

Routing Networks: Adaptive Selection of Non-linear Functions for Multi-Task Learning (ICLR 2018)

Clemens Rosenbaum, Tim Klinger, Matthew Riemer

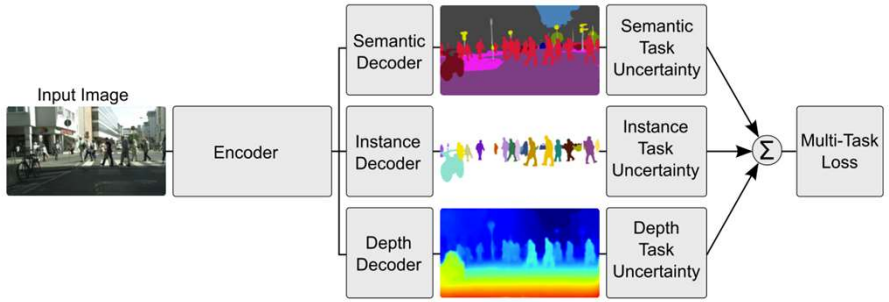
Oct 12, 2019 – Presented by Andrew Luo

Motivation

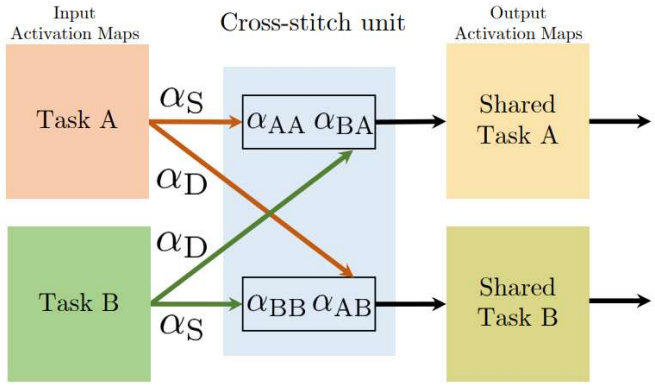
- Multi-task learning
  - Train a single network on multiple tasks
  - Some tasks may have improved performance under joint training
  - Other tasks may degrade when trained jointly
- Many ways to fix this
  - Confusion matrix for multi-task, hand tune architecture or losses
  - Reweight losses…
  - Reweight features… etc. etc.

Examples of this kind of work

- Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics (2017)
  - Re-weights losses by per-task variance



- Cross-stitch Networks for Multi-task Learning (2016)
  - “Cross-stich” units for linear combinations of per-task features



Their proposal

- Have a pool of functional units, have a single routing unit
- Input is passed to a routing unit
- Routing unit passes input to a functional unit that it chooses
- Passed back to router
- Router decides to do nothing, or feed to another functional unit

Many ways to structure routers

- They mention many ways to design the router
  - Can be conditioned on just previous input or just task selection or both
  - Can be conditioned on past decisions
  - Can be conditioned on current recursive depth etc.

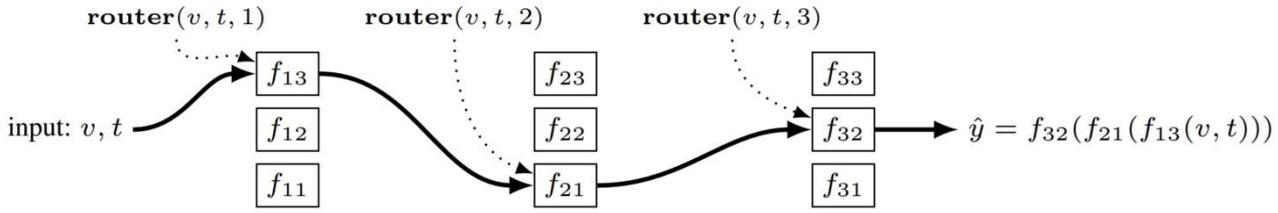
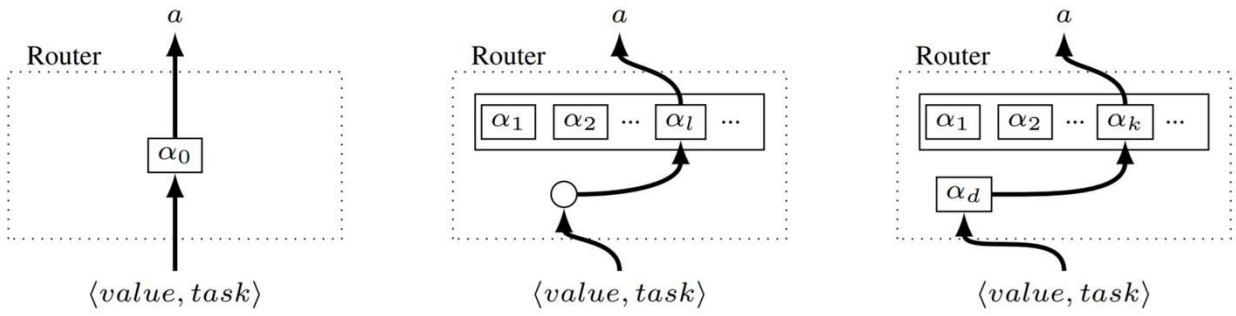


Figure 1: Routing (forward) Example

Many ways to structure routers

- Can have one router for all tasks
- Can have one router per task
- Can have router for per-task routers
- Given above, can be per recursive depth, or all depths shared



Many ways to design the functional blocks

- Also many ways to design the functional blocks
  - Set of blocks at time ***t*** can be the same set as ***t-1***
  - Can be a completely new set
  - Can be different types of layers (as long as input dimensions are the same)

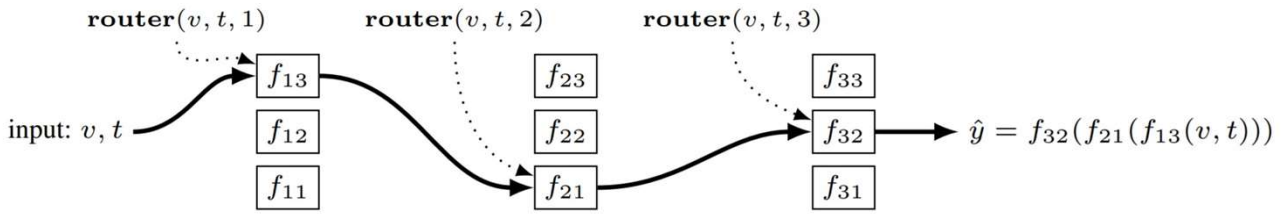


Figure 1: Routing (forward) Example

Inference (after it is trained)

Algorithm 1: Routing Algorithm

input : x, t, n:
$x \in \mathbb{R}^d$, d the representation dim;
t integer task id;
n max depth

output: v - the vector result of applying the composition of the selected functions to the input x

$v \leftarrow x$
for i *in* $1...n$ **do**
  $a \leftarrow \textbf{router}(x, t, i)$
  if $a \neq$ *PASS* **then**
    $x \leftarrow \textbf{function\_block}_a(x)$
return v

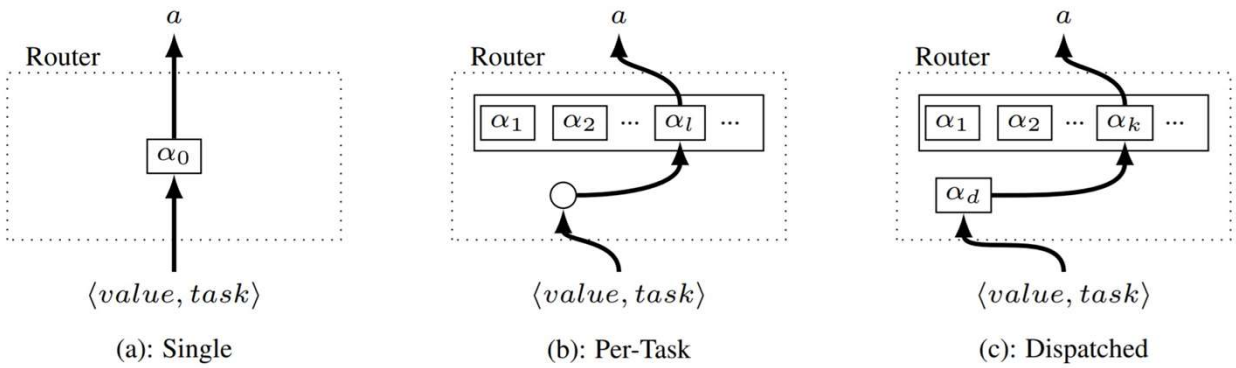
1. Initialize the input
2. Initialize the task id
3. Initialize the maximum recursive depth
4. Feed input to router
5. Router selects function block, and feeds input to block
6. Block passes output back to router
7. Router can terminate, or go to **STEP 5**

How to train

- Training via reinforcement learning
  - Loss landscape is changing, because we are also optimizing the parameters of each block
  - Loss landscape is changing, because we have multiple tasks that could be competing against each other

How to train

- For (a), we can use REINFORCE or Q-learning
- For (b) and (c), they use an algorithm for multi-agent learning called Weighted Policy Learner *(WPL)*



Actual design of their experiments

- 4 conv layer backbone that is NOT learned via routing
- Tasks are multi-class MNIST, CIFAR-100-coarse_superclass, Mini-ImageNet
- 3 fc layers that are learned via routing
- Experiment 1:
  - 1 router per layer
  - Agent-per-task or single-agent-all-tasks
  - WPL or REINFORCE or Q-Learning
  - Agent is either neural network (takes in previous input) **OR**
  - Agent is just a table, just takes in task-ID and ‘recursion’ depth
  - Set of layers for depth=1 is different from depth=2, different from depth=3

Results for experiment 1

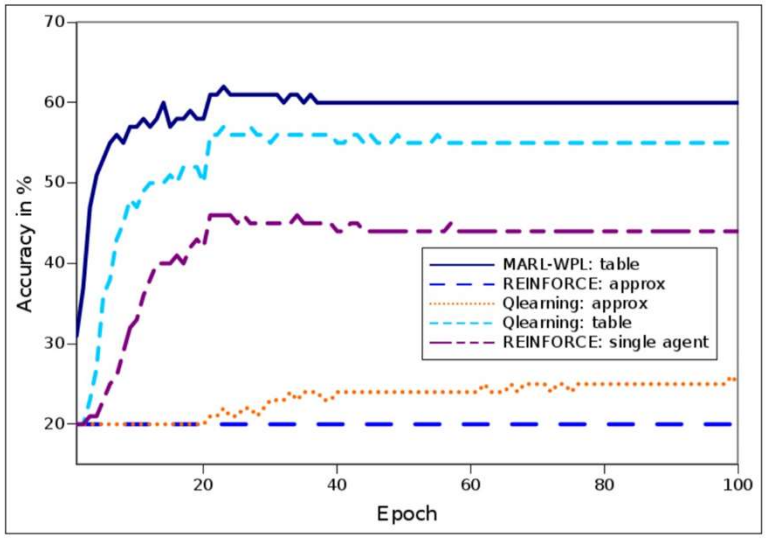
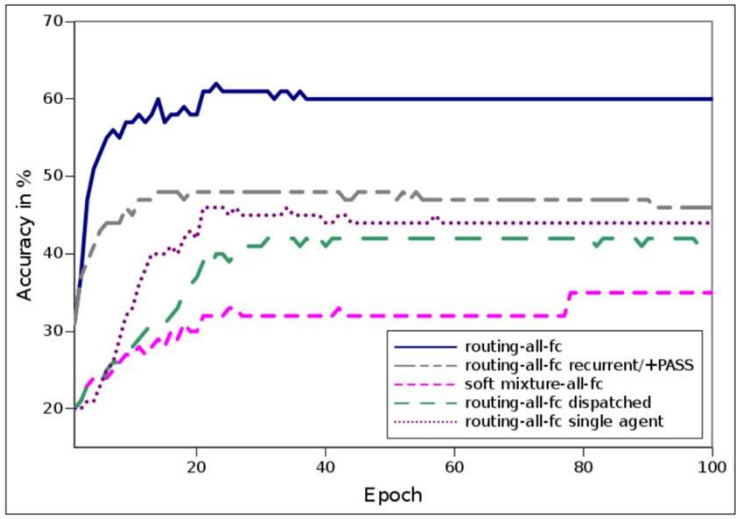


Figure 4: Influence of the RL algorithm on CIFAR-MTL. Detailed descriptions of the implementation each approach can be found in the Appendix in Section 7.3.

Actual design of their experiments

- Experiment 2:
  - 1 router per layer
  - Agent-per-task or single-agent-all-tasks
  - WPL or REINFORCE
  - Agent is just a table, just takes in task-ID and ‘recursion’ depth
  - Set of layers for depth=1 is different from depth=2, different from depth=3
  - OR set of layers is shared (see recurrent label in the next slide)

Results for experiment 2



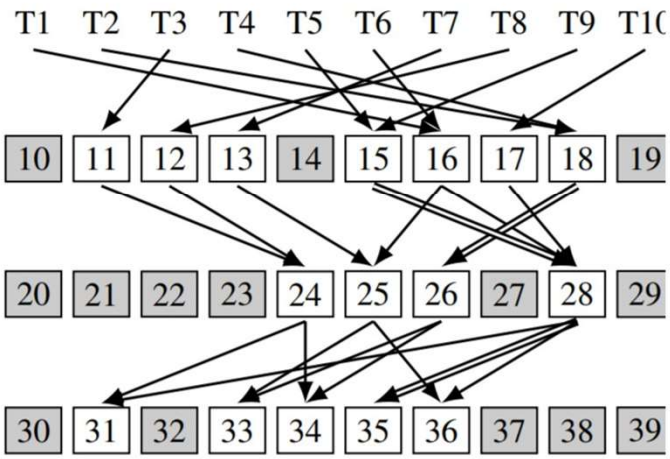
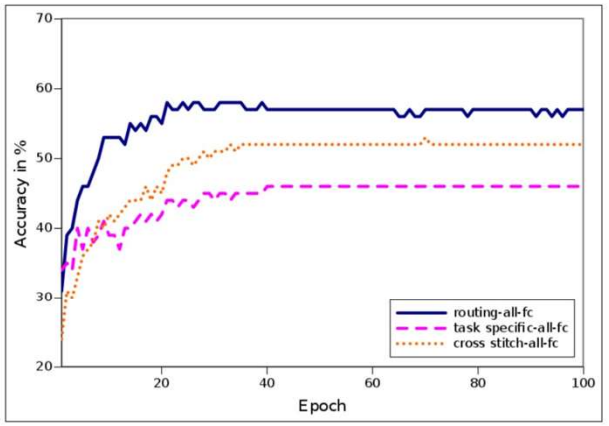
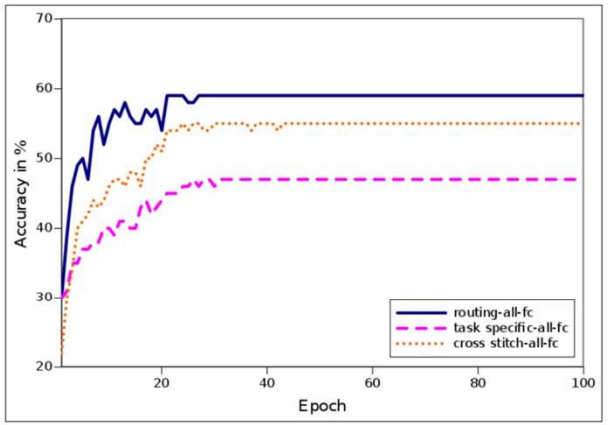


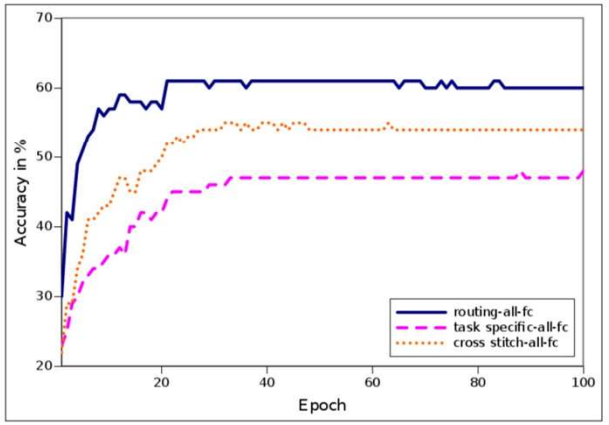
Figure 11: An actual routing ma
MNIST-MTL.



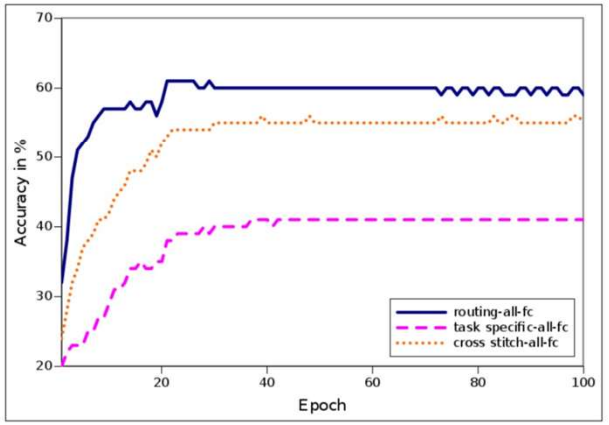
(a) first 2 tasks



(b) first 3 tasks



(c) first 5 tasks



(d) first 10 tasks

Inferring and Executing Programs for Visual Reasoning (ICCV 2017)

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, Ross Girshick

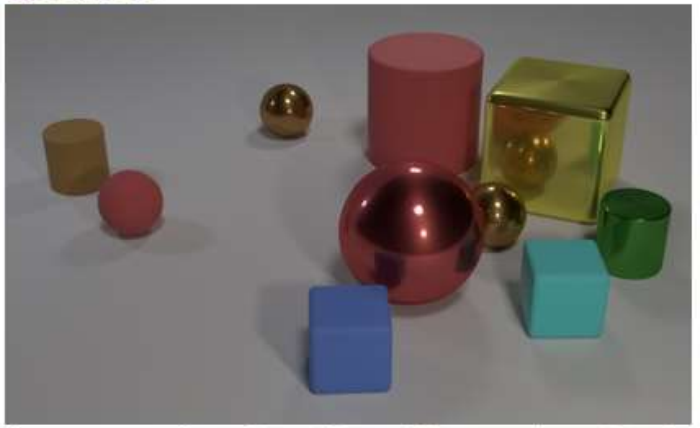
Problem definition

- Visual Question and Answering (VQA)
  - Look at picture, look at question
  - Answer question

VQA
Q: How many bikes are there?
A: 2
Q: What number is the bus?
A: 48

VQA
Q: How many pickles are on the plate?
A: 1
Q: What is the shape of the plate?
A: round

Questions in CLEVR test various aspects of visual reasoning including **attribute identification**, **counting**, **comparison**, **spatial relationships**, and **logical operations**.



Q: Are there an **equal number** of **large things** and **metal spheres**?
Q: **What size** is the **cylinder** **that is left of** the **brown metal** thing **that is left of** the **big sphere**?
Q: There is a **sphere** with the **same size as** the **metal cube**; is it **made of the same material as** the **small red sphere**?
Q: **How many** objects are **either** **small cylinders** or **red** things?

General approaches

RNN + CNN based approach:

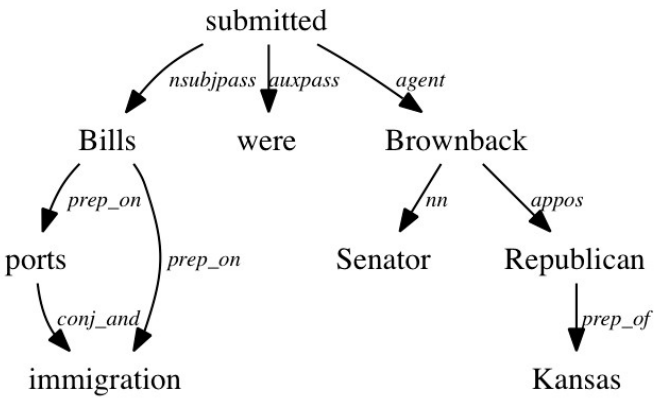
1. Use RNN or attention based language model to produce question embedding, use CNN to produce image embedding
2. Fuse the embedding somehow, maybe add some sort of attention mechanism
3. Use MLP to produce the answer from the embedding

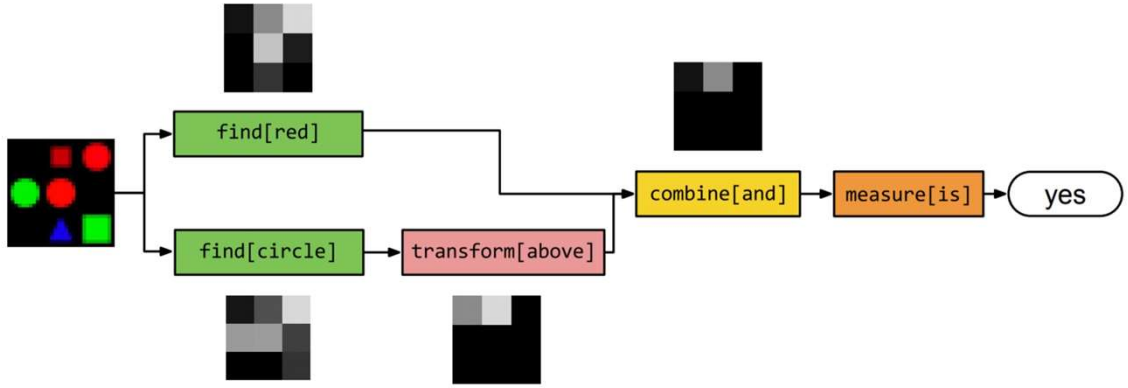
Module based approach (2017 onwards):

1. Use CNN to produce image embedding (Also see using Mask-RCNN work from Kexin Yi & Jiajun Wu)
2. Construct hierarchy of modules, using embedding
3. Train modules to produce answer

Previous work

- Neural Module Networks (CVPR 2016)
  - Uses Stanford Parser to produce a dependency parse
  - Construct Modules from the parse

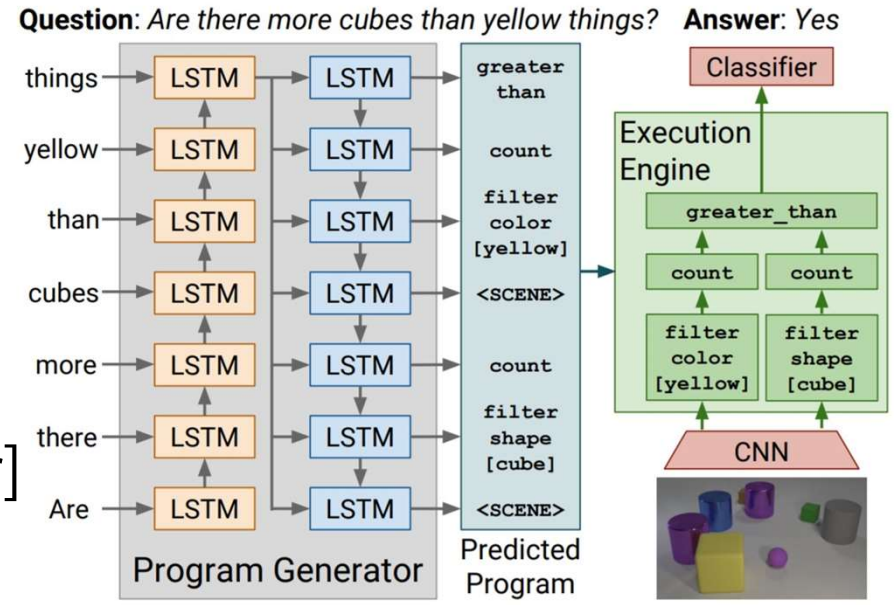




(b) NMN for answering the question *Is there a red shape above a circle?* The

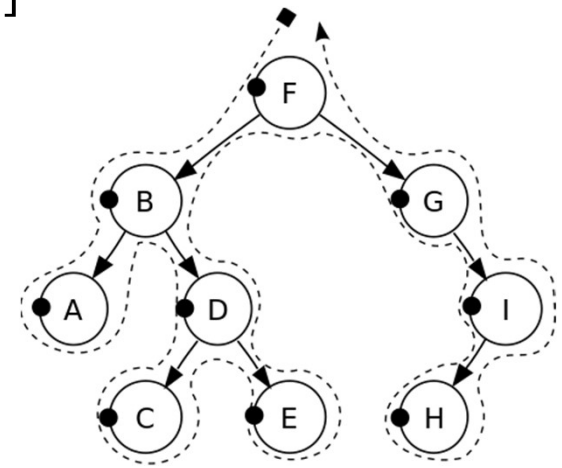
Their approach

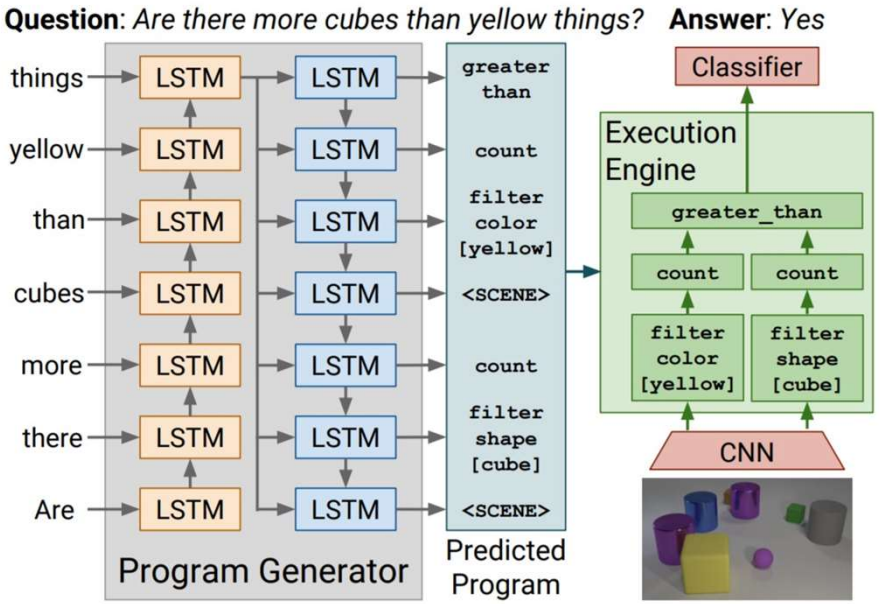
- They have a program generator
- And a separate program executor
- First trained using full supervision [Image, Question, Program, Answer]
- Fined tuned with reinforcement learning [Image, Question, Answer]



Program Generator

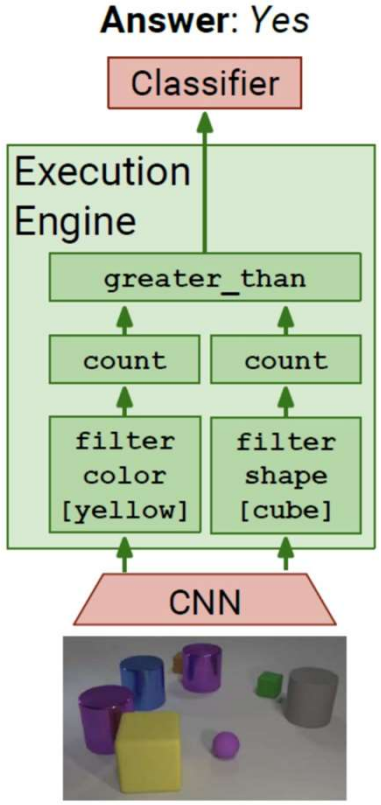
- LSTM Seq2Seq to model
- When converting to tree, use prefix traversal (Preorder)
- Incase input is not sufficient, pad with [SCENE]





Function Modules

- Fixed set of function
- Functions are Res Blocks
- Function can be specific to given attribute (a specific color, size etc.) – double check in code
- 1 input or 2 input modules
- In 2 input case, reduce to 1 input case using 1x1 conv
- Uses ResNet conv4 as feature extractor



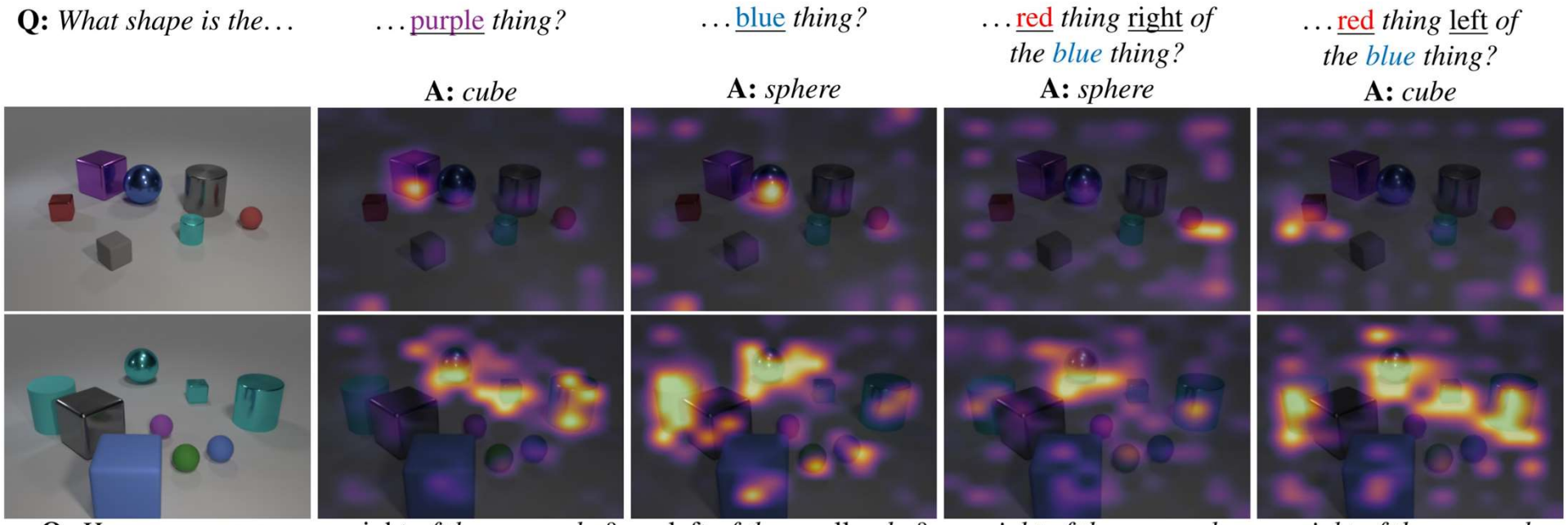


Figure 3. Visualizations of the norm of the gradient of the sum of the predicted answer scores with respect to the final feature map. From

Method	Exist	Count	Compare Integer			Query				Compare				Overall
			Equal	Less	More	Size	Color	Mat.	Shape	Size	Color	Mat.	Shape	
Q-type mode	50.2	34.6	51.4	51.6	50.5	50.1	13.4	50.8	33.5	50.3	52.5	50.2	51.8	42.1
LSTM	61.8	42.5	63.0	73.2	71.7	49.9	12.2	50.8	33.2	50.5	52.5	49.7	51.8	47.0
CNN+LSTM	68.2	47.8	60.8	74.3	72.5	62.5	22.4	59.9	50.9	56.5	53.0	53.8	55.5	54.3
CNN+LSTM+SA [46]	68.4	57.5	56.8	74.9	68.2	90.1	83.3	89.8	87.6	52.1	55.5	49.7	50.9	69.8
CNN+LSTM+SA+MLP	77.9	59.7	60.3	83.7	76.7	85.4	73.1	84.5	80.7	72.3	71.2	70.1	69.7	73.2
Human† [19]	96.6	86.7	79.0	87.0	91.0	97.0	95.0	94.0	94.0	94.0	98.0	96.0	96.0	92.6
Ours-strong (700K prog.)	**97.1**	**92.7**	**98.0**	**99.0**	**98.9**	**98.8**	**98.4**	**98.1**	**97.3**	**99.8**	**98.5**	**98.9**	**98.4**	**96.9**
Ours-semi (18K prog.)	95.3	90.1	93.9	97.1	97.6	98.1	97.1	97.7	96.6	99.0	97.6	98.0	97.3	95.4
Ours-semi (9K prog.)	89.7	79.7	85.2	76.1	77.9	94.8	93.3	93.1	89.2	97.8	94.5	96.6	95.1	88.6