Lecture 11:

Imposing Task-Specific Structure on DNNs

Visual Computing Systems
Stanford CS348V, Winter 2018
Today

- Examples of DNN authors imposing structure on networks to better perform a desired task

- For each example, consider:
  - What knowledge does the human inject?
  - What does the computer learn?

- Tasks:
  - Image/video compression networks  [Toderici 16, Tsai AAAI 18]
  - Visual Question Answering via Neural Module Networks  [Johnson 17, 17]
Image compression using DNNs
Review: JPG image compression

- Lossy compression designed to retain information that is most important to human perception
- Human-designed compact representation

\[
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\
0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\
0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Coefficient reordering

RLE compression of zeros
Entropy compression of non-zeros
Compressed bits
Deep learning learns useful representations

- Can we apply deep learning techniques to obtain compact image representations for efficient storage and transmission?

- Discussion: why should we even consider this? (isn’t JPG a pretty darn good representation?)
Example: autoencoder
Compressing 32x32 8-bit RGB thumbnails (24 bpp)

Auto-encoder: learn to compress (encode) and reconstruct (decode) the input signal
- Jointly train D, B, and E using supervision from Loss(x, x')
Progressive encoding: chain copies of autoencoder
(each iteration contributes additional bits to stored output)

Input: 32x32x3 residual \( r_{t-1} \)

\[
F_t(r_{t-1}) = D_t(B(E_t(r_{t-1})))
\]

\( r_0 \) = input image to compress

Version 1:
each iteration predicts the residual

\[
r_t = F_t(r_{t-1}) - r_{t-1}
\]

\[
x' = \sum_{t=1}^{N} F_t(r_{t-1})
\]

Version 2: (stateful E() and D() units)
each iteration predicts input image

\[
r_t = F_t(r_{t-1}) - r_0
\]

\[
x' = F_N(r_{N-1})
\]

In both cases, loss given by \( \| r_t \|_2^2 \) for all \( t \)

[Toderici ICLR 16]
Binarization

- **Step 1:** output of encoder passes through fully-connected layer with \( m \) outputs (to “squeeze” to desired number of outputs)
- **Step 2:** quantize each output to a bit

\[
B(x) = f(tanh(Wx + b))
\]

\[
f(x) = \begin{cases} 
-1 & x < 0 \\
+1 & x \geq 0 
\end{cases}
\]

Add random perturbation during training (regularization):

\[
f(x) = x + \epsilon
\]

\[
\epsilon \sim \begin{cases} 
1 - x & \text{with probability } \frac{1+x}{2}, \\
-x - 1 & \text{with probability } \frac{1-x}{2},
\end{cases}
\]
Version 1 autoencoder

Fully-connected version:
Input is 8x8 block
Each fully connected layer has 512 outputs and tanh non-linearity
Each iteration through auto encoder yields 4 bits (two iterations shown)
Version 1 autoencoder (convolutional)

Convolutional version:
2 bits per spatial location of output per iteration
32x32 input → 8x8 spatial outputs (128 bits per iteration)

Figure 1: The fully-connected residual autoencoder. We depict a two-iteration architecture, with the goal of the first iteration being to encode the original input patch and the goal of the second iteration being to encode the residual from the first level's reconstruction. In our 64-bit results, reported in Table 1, we have 16 iterations giving 4 bits each. The blocks marked with 512 are fully-connected neural network layers with 512 units and \( \text{tanh} \) nonlinearities. The loss applied to the residuals in training is a simple L2 measure.

Figure 2: The fully-connected LSTM residual encoder. The 512 LSTM blocks represent LSTM layers with 512 units. This figure shows an unrolling of the LSTM, needed for training, to two time steps. The actual architecture would have only the first row of blocks, with the functionality of the second row (and subsequent recursions) being realized by feeding the residual from the previous pass back into the first LSTM block. For the results reported in Table 1, this repeated feeding back was done 16 times, to generate 64 bit representations. The vertical connections between the LSTM stages in the unrolling shows the effect of the persistent memory instead each LSTM. The loss is applied to the residuals in training is a simple L2 measure. Note that in contrast to Figure 1, in which the network after the first step is used to predict the previous step's residual error, in this LSTM architecture, each step predicts the actual output.

Figure 3: The convolutional / deconvolutional residual encoder. The convolutional layers are depicted as sharp rectangles, while the deconvolutional layers are depicted as rounded rectangles. The loss is applied to the residuals.
Version 2 autoencoder (LSTM-based)

(Convolutional form of the LSTM auto encoder also exists)

LSTM version: predicts source image each iteration (not a residual)

LSTM units:
- Recurrent: output from iteration t-1 fed into unit in iteration t
- Stateful: each unit maintains its own hidden state

Figure 1: The fully-connected residual autoencoder. We depict a two-iteration architecture, with the goal of the first iteration being to encode the original input patch and the goal of the second iteration being to encode the residual from the first level's reconstruction. In our 64-bit results, reported in Table 1, we have 16 iterations giving 4 bits each. The blocks marked with 512 are fully-connected neural network layers with 512 units and \( \tanh \) nonlinearities. The loss applied to the residuals in training is a simple L2 measure.

Figure 2: The fully-connected LSTM residual encoder. The 512 LSTM blocks represent LSTM layers with 512 units. This figure shows an unrolling of the LSTM, needed for training, to two time steps. The actual architecture would have only the first row of blocks, with the functionality of the second row (and subsequent recursions) being realized by feeding the residual from the previous pass back into the first LSTM block. For the results reported in Table 1, this repeated feeding back was done 16 times, to generate 64 bit representations. The vertical connections between the LSTM stages in the unrolling shows the effect of the persistent memory instead each LSTM. The loss is applied to the residuals in training is a simple L2 measure. Note that in contrast to Figure 1, in which the network after the first step is used to predict the previous step's residual error, in this LSTM architecture, each step predicts the actual output.

Figure 3: The convolutional / deconvolutional residual encoder. The convolutional layers are depicted as sharp rectangles, while the deconvolutional layers are depicted as rounded rectangles. The loss is applied to the residuals.

[Toderici ICLR 16]
We describe various methods for variable-length encoding of image patches using neural networks, with JPEG, while the convolutional/deconvolutional LSTM model is able to significantly outperform JPEG on the SSIM perceptual metric. And demonstrate that for the given benchmark, the fully-connected LSTM model can perform on par against the (de)convolutional LSTM encoder at these targets. In both cases, the LSTM model presented in Section 3.5) with a given bit budget of either 64 or 128 bytes, and compared its SSIM. In terms of coding efficiency, we took an autoencoder architecture (one iteration of the model.

Compressed images with conv/deconv LSTM architecture

Compressed images with LSTM architecture

Compressed images with conv/deconv LSTM architecture

**Average bpp:**

<table>
<thead>
<tr>
<th></th>
<th>From left to right</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG</td>
<td>0.641 0.875 1.117 1.375</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.625 0.875 1.125 1.375</td>
</tr>
<tr>
<td>(De)Convolutional LSTM</td>
<td>0.625 0.875 1.125 1.375</td>
</tr>
</tbody>
</table>
## Compression results

Table 1: Comparison between the proposed methods for a given compression target size (in bytes) on the 32x32 image benchmark.

<table>
<thead>
<tr>
<th>Method</th>
<th>Patch Size</th>
<th>SSIM / 64B Target (Header-less Size)</th>
<th>SSIM / 128B Target (Header-less Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header-less JPEG</td>
<td>8×8</td>
<td>0.70 (72.5 bytes avg.)</td>
<td>0.80 (133 bytes avg.)</td>
</tr>
<tr>
<td>Header-less JPEG 2000</td>
<td></td>
<td>0.66 (73 bytes avg.)</td>
<td>0.77 (156 bytes avg.)</td>
</tr>
<tr>
<td>Header-less WebP</td>
<td></td>
<td>0.62 (80.7 bytes avg.)</td>
<td>0.73 (128.2 bytes avg.)</td>
</tr>
<tr>
<td>Fully Connected Residual Encoder (Shared Weights)</td>
<td>8×8</td>
<td>0.46</td>
<td>0.48</td>
</tr>
<tr>
<td>Fully Connected Residual Encoder (Distinct Weights)</td>
<td>8×8</td>
<td>0.65</td>
<td>0.75</td>
</tr>
<tr>
<td><strong>LSTM Compressor</strong></td>
<td><strong>8×8</strong></td>
<td><strong>0.69</strong></td>
<td><strong>0.81</strong></td>
</tr>
<tr>
<td>Conv/Deconv Residual Encoder (Shared Weights)</td>
<td>32×32</td>
<td>0.45</td>
<td>0.46</td>
</tr>
<tr>
<td>Conv/Deconv Residual Encoder (Distinct Weights)</td>
<td>32×32</td>
<td>0.65</td>
<td>0.75</td>
</tr>
<tr>
<td>Convolutional/Deconvolutional Autoencoder</td>
<td>32×32</td>
<td>0.76</td>
<td>0.86</td>
</tr>
<tr>
<td><strong>Conv/Deconv LSTM Compressor</strong></td>
<td><strong>32×32</strong></td>
<td><strong>0.77</strong></td>
<td><strong>0.87</strong></td>
</tr>
</tbody>
</table>

**SSIM**: Structural similarity index (attempt to quantify perceived similarity of images)

---

4.4 A **NALYSIS**

These 32×32 images contain considerable detail that is perceptually relevant. As can be seen in Figure 4, compressing these images without destroying salient visual information or hallucinating false details is challenging. At these very low bitrates and spatial resolution, JPEG block artifacts become extremely prominent, and WebP either introduces blocking or overly blurs the image depending on the strength of the internal filter. Color smearing artifacts due to the codecs' default (4:2:0) chroma subsampling are also clearly visible.

Compared to JPEG, the non-convolutional LSTM model slightly reduces inter-block boundaries on some images but can also lead to increased color bleeding (e.g., on mandrill as shown in Figure 4). Furthermore, the visual quality never exceeds JPEG on average as measured by SSIM and shown in Figure 5. This motivates the (de)convolutional LSTM model, which eliminates block artifacts while avoiding excessive smoothing. It strikes the best balance between preserving real detail and avoiding color smearing, false gradients, and hallucinated detail not present in the original image.

Note that the (de)convolutional LSTM model exhibits perceptual quality levels that are equal to or better than both JPEG and WebP at 4% – 12% lower average bitrate. We see this improvement despite the fact that, unlike JPEG and WebP, the LSTMs do not perform chroma subsampling as a preprocess. However, at the JPEG quality levels used in Figure 4, disabling subsampling (i.e., using 4:4:4 encoding) leads to a costly increase in JPEG's bitrate: 1.32 – 1.77 bpp instead of 1.05 – 1.406 bpp, or 26% greater. This means that if we desired to preserve chroma fidelity, we would need to drastically...

---

[Toderici ICLR 16]
Summary / thoughts

- **Idea:** learn how to compress thumbnail-sized images by trying to compress large database of tiny images
  - Loss is not perceptually motivated (if there was a differentiable perceptual loss metric, they would have used it instead of L2 on pixel residual)

- **Improvement on JPG for small images, future work extends to large images by exploiting global redundancy** [Toderici 2016]

- **Why use learning for this problem?**
  - Potential for higher quality encode (learn better representations than humans can manually craft)
  - General mechanism to specialize representations for task
    - [Toderici 2016]: specific to thumbnail images
    - What about camera-viewpoint specific compression?
    - Task-based definition of loss rather than pixels (compress subject to still being able to recognize objects)
Video compression

- Idea: use deep network to compress only the residual of H.264 compressed video stream (recall: residual = loss due to H.264 compression)
Takeaway:

- Can learn to compress residuals effectively when compressor tailored to specific type of video stream

Figure 3: PSNR comparisons on four datasets at different bandwidths. We compare our pipeline with H.264 and an artifact-removal method based on (Kim, Lee, and Lee 2016; Zhang et al. 2017).

(a) KITTI  (b) Assassins Creed

Figure 4: SSIM comparisons on four datasets at different bandwidths. We compare our pipeline with H.264 and an artifact-removal method based on (Kim, Lee, and Lee 2016; Zhang et al. 2017).

(c) Skyrim  (d) Borderlands

Figure 5: Example results on the KITTI and video game datasets. We compare our pipeline with H.264 and an artifact-removal method. The corresponding bit rate and PSNR are shown next to the images. Best viewed with enlarged images.
Camera-specific compression?

Security cameras (stationary)

Head mounted cameras
(person only sees so many things)

On-vehicle cameras
(specialization to neighborhood or city?)
Discussion:
Neural Module Networks
Visual question answering

Common approach at the time:
- DNN (LSTM) to compute embedding of the question
- Use the question to "attend" to the relevant part of the image
- Mix visual features in attended region and question embedding to answer question

Figure 3: Our architecture for VQA: Multimodal Compact Bilinear (MCB) with Attention. Conv implies convolutional layers and FC implies fully connected layers. For details see Sec. 3.2.

\[ \text{CNN} \text{(ResNet152)} \rightarrow 16k \times 14 \times 14 \rightarrow 2048 \times 14 \times 14 \rightarrow 2048 \times 14 \times 14 \]

WE, LSTM

\[ \text{Softmax} \]

Weighted Sum

1 x 14 x 14 \n512 x 14 x 14 \nCNN \text{ (ResNet152)} \ 16k \times 14x14 \ 2048x14x14 \ 2048x14x14 \ Conv, ReLU \ Conv \]
Example challenge: debugging failures

what is the color of the shirt on the man

- red (0.24)
- blue (0.19)
- orange (0.16)
- gray (0.14)
- white (0.09)

Took 0.049 sec

what is the color of the shirt on the person on the left

- red (0.39)
- orange (0.30)
- pink (0.15)
- blue (0.07)
- white (0.03)

Took 0.051 sec
Main idea: images and questions can be answered from contents of images alone (no external knowledge)

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 metal things?
Second major idea: neural module networks

- If a question is just a logical expression, why isn’t the program used to answer it dependent on the structure of a question?

**Question text:**
How many cylinders are in front of the small thing and on the left side of the green object?

**NLP semantic parser**
question to “program” (program = neural module network)

**DNN Topology specific to question:**
(Unique DNN per question)
Grammar of CLEVR questions

CLEVR function catalog

- value
- objects
- objects
- objects
- objects
- objects
- object
- value
- value
- value
- number
- number
- number
- object
- object
- object
- objects
- objects
- objects

Filter <attr> → objects
And → objects
Or → objects
Exist Count → yes/no
Query <attr> → value
Equal → yes/no
Equal → yes/no
Less / More → yes/no
Same <attr> → objects
Relate → objects
Unique → object

[Johnson 2017]
State-of-the-art at time of CLEVR

Accuracy of state-of-art VQA techniques on CLEVR dataset
Hand-engineering CLEVR

- **Simplification:**
  - Assume question text to program parse is perfect
  - Just answer question based on program+image

- **Learned modules (manually annotated intermediate results based on data in CLEVR scene graph description):**
  - Detection: YOLO trained on CLEVR objects
  - Color: SVM trained on VGG conv5 features
  - Material: SVM trained on VGG conv5 features
  - Size: SVM trained on bounding box coordinates
  - Depth: LR trained on bounding box coordinates

- **Not learned modules (implemented by hand):**
  - Counting
  - Integer comparison
  - Attribute comparison
  - Spatial relations (left, right, behind, in front)
  - Logical operators (and, or)
# Hand-engineered solution

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Detect</th>
<th>Shape</th>
<th>Color</th>
<th>Material</th>
<th>Size</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>96.39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94.57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><em>no qtype</em></td>
</tr>
<tr>
<td>90.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><em>yes qtype</em></td>
</tr>
</tbody>
</table>

[Slide credit: Karima Ma]
Testing on real world images*
(* Text question rewritten as expression manually. Focus is not NLP.)

Q. “What color is the bowl”? A. orange

Q. “What color is the bus?” A. green

[Slide credit: Karima Ma]
Testing on real world images

Q. “How many sheep are in the photo?”
A. 10

Q. “How many zebras are there?”
A. 5
Testing on real world images

Q. “What is on the bench?”
A. airplane

Q. “How many people are on the elephant?”
A. 3
Back to CLEVR…

- Challenge 1: convert ambiguous English text to valid program
- Challenge 2: learning behavior of individual modules
  - CLEVR provides module network definitions for a question, as well as the end-to-end answer to the question, but not a specification of how individual modules should behave!
Solving CLEVR

Challenge 1: convert ambiguous English text to valid program
- [Johnson 2017] trains program generator via REINFORCE <see paper>
- Or just assume ground truth program known for today’s discussion

Challenge 2: learning behavior of individual modules
- Module definition: WxHxC input —> WxHxC output
- One ResNet block with 3x3 convs
- Binary modules concat inputs WxHx2C
- Compositions of programs as valid DNNs

“How many cylinders are in front of the small thing and on the left side of the green object?”

DNN Topology specific to question: (Unique DNN per question)
Discussion: end-to-end learning solution

- Construct neural module network
- Evaluate network to compute answer
- Backpropagate loss, updating weights of modules used in neural module network
  - Classification loss since final layer is a classifier over all possible value answers

What has happened:
- Learning system only knows what module to use when a module of type T is needed
- Learning system has no definition for each of the module’s behavior/semantics
- By using the module X in many tasks whenever “FilterColor(x)” is needed, module X ultimately should learn to perform a filtering by color operation
- But there is no guarantee of what each module learns!
First, results...

<table>
<thead>
<tr>
<th>Method</th>
<th>Exist</th>
<th>Count</th>
<th>Compare Integer</th>
<th>Query</th>
<th>Compare Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-type mode</td>
<td>50.2</td>
<td>34.6</td>
<td>51.4 51.6 50.5</td>
<td>50.1</td>
<td>50.8 33.5</td>
</tr>
<tr>
<td>LSTM</td>
<td>61.8</td>
<td>42.5</td>
<td>63.0 73.2 71.7</td>
<td>49.9</td>
<td>50.8 33.2</td>
</tr>
<tr>
<td>CNN+LSTM</td>
<td>68.2</td>
<td>47.8</td>
<td>60.8 74.3 72.5</td>
<td>62.5</td>
<td>59.9 50.9</td>
</tr>
<tr>
<td>CNN+LSTM+SA [46]</td>
<td>68.4</td>
<td>57.5</td>
<td>56.8 74.9 68.2</td>
<td>90.1</td>
<td>89.8 87.6</td>
</tr>
<tr>
<td>CNN+LSTM+SA+MLP</td>
<td>77.9</td>
<td>59.7</td>
<td>60.3 83.7 76.7</td>
<td>85.4</td>
<td>84.5 80.7</td>
</tr>
<tr>
<td>Human† [19]</td>
<td>96.6</td>
<td>86.7</td>
<td>79.0 87.0 91.0</td>
<td>97.0</td>
<td>94.0 94.0</td>
</tr>
<tr>
<td>Ours-strong (700K prog.)</td>
<td>97.1</td>
<td>92.7</td>
<td>98.0 99.0 98.9</td>
<td>98.8</td>
<td>98.4 98.1 97.3</td>
</tr>
<tr>
<td>Ours-semi (18K prog.)</td>
<td>95.3</td>
<td>90.1</td>
<td>93.9 97.1 97.6</td>
<td>98.1</td>
<td>97.7 96.6</td>
</tr>
<tr>
<td>Ours-semi (9K prog.)</td>
<td>89.7</td>
<td>79.7</td>
<td>85.2 76.1 77.9</td>
<td>94.8</td>
<td>93.3 89.2</td>
</tr>
</tbody>
</table>

Better than human performance!

Better performance than manually designing modules to perform subtasks!
- Discussion: What does this imply?
Attempting to introspect network behavior

Q: What shape is the... purple thing? blue thing? red thing right of the blue thing? red thing left of the blue thing?
A: cube sphere sphere cube

Q: How many cyan things are... right of the gray cube? left of the small cube? right of the gray cube and left of the small cube? right of the gray cube or left of the small cube?
A: 3 2 1 4

Visualizing loss gradient w.r.t. last WxHxC feature map output by DNN

Location of most salient part of image is intuitive.
(this part of image most impacts loss)
Do modules generalize beyond CLEVR?

- **Experiment 1:** train only configuration A
  - Configuration A
    - all cubes are gray, brown, blue, yellow
    - all cylinders are red
  - Configuration B (flip shape/color association)
    - all cubes are red
    - all cylinders are gray, brown, blue, yellow
- **Experiment 2:** have humans write new questions
  - Underlined words are words not in CLEVR dataset
- **Fine tune program generator (end-to-end based on final answers)**

<table>
<thead>
<tr>
<th>Method</th>
<th>Train A</th>
<th>Train B</th>
<th>Finetune A</th>
<th>Finetune B</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>55.2</td>
<td>50.9</td>
<td>51.5</td>
<td>54.9</td>
</tr>
<tr>
<td>CNN+LSTM</td>
<td>63.7</td>
<td>57.0</td>
<td>58.3</td>
<td>61.1</td>
</tr>
<tr>
<td>CNN+LSTM+SA+MLP</td>
<td>80.3</td>
<td>68.7</td>
<td>75.7</td>
<td>75.8</td>
</tr>
<tr>
<td>Ours (18K prog.)</td>
<td>96.6</td>
<td>73.7</td>
<td>76.1</td>
<td>92.7</td>
</tr>
</tbody>
</table>

Q: Is there a blue box in the items? A: yes

Q: What shape object is farthest right? A: cylinder

Q: Are all the balls small? A: no

Predicted Program:
- `exist` filter_shape[cube]
- `filter_color[blue]` scene

Predicted Answer: ✓ yes

Predicted Program:
- `query.shape` unique
- `relate[right]`
- `unique`
- `filter_shape[cylinder]`
- `filter_color[blue]` scene

Predicted Answer: ✓ cylinder

Predicted Program:
- `equal.size`
- `query.size`
- `unique`
- `filter_shape[sphere]`
- `scene`
- `query.size`
- `unique`
- `filter_shape[sphere]`
- `filter.size[small]`
- `scene`

Predicted Answer: ✓ no
Discussion questions

- Would you agree that training modules for CLEVR is a form of multi-task learning? (each question is a unique task)

- What were the two major benefits of end-to-end training approach?

- What are your thoughts on when learning should be used?
  - Interesting question: would left_of() module work on real-world images?

- Given the results in the paper how would you describe the semantics of a module? (e.g., the module that is supposed to filter_color(red))