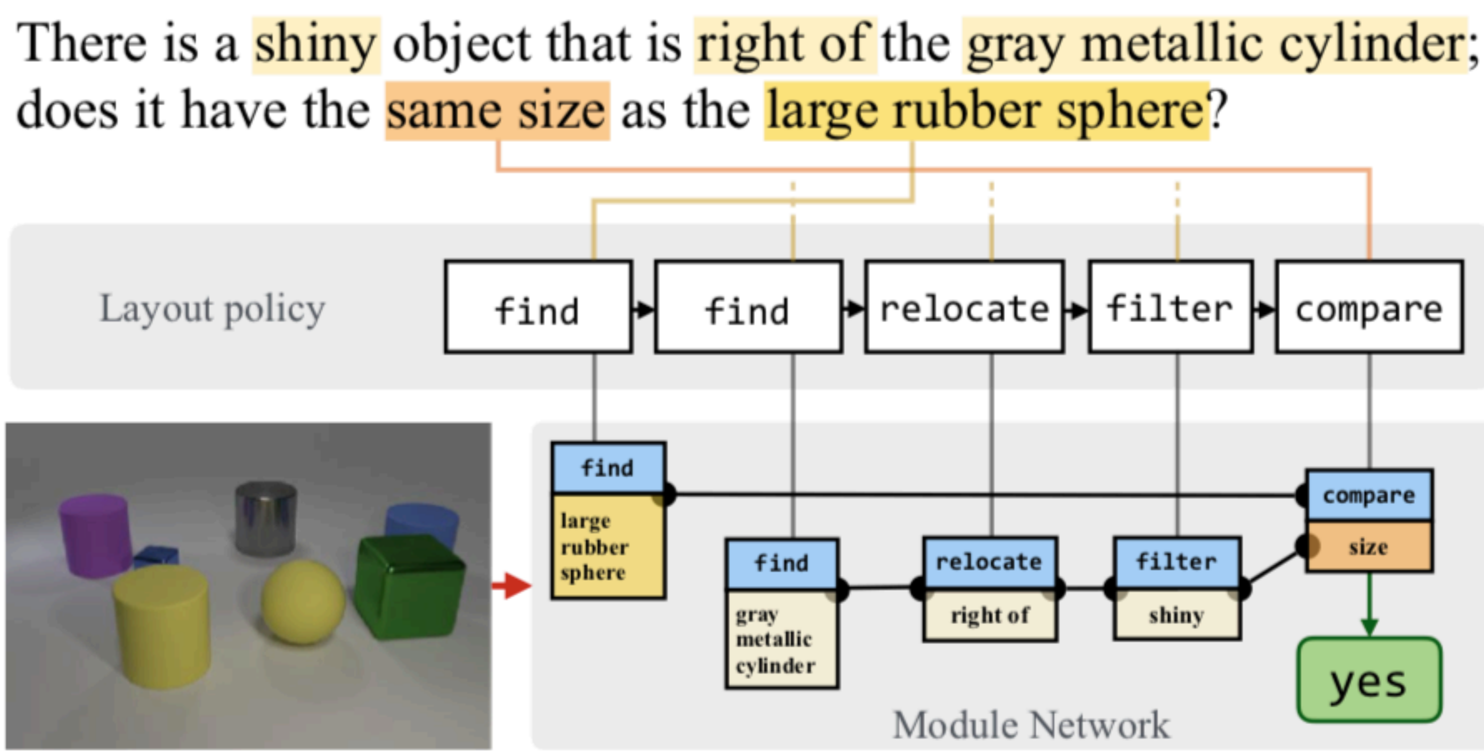
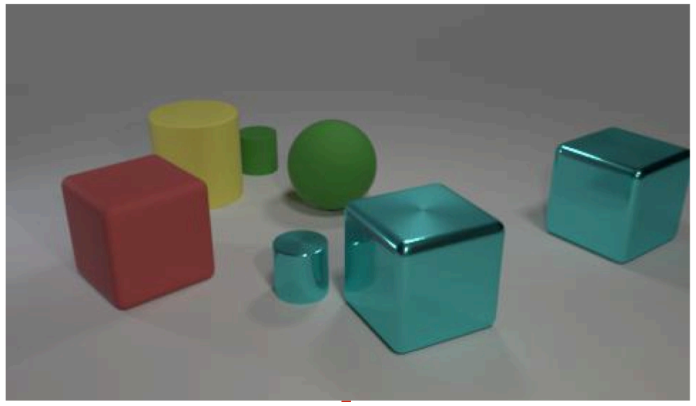
# Learning to Reason: End-to-End Module Networks for Visual Question Answering



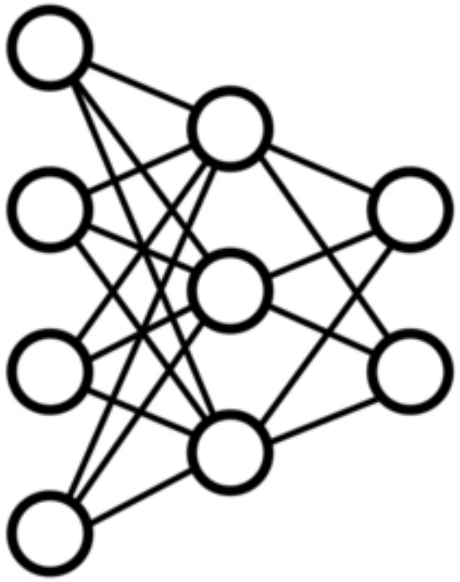

문 태 봉

# Motivation



How many other things are of the same size as the green matte ball?

Monolithic Neural Network

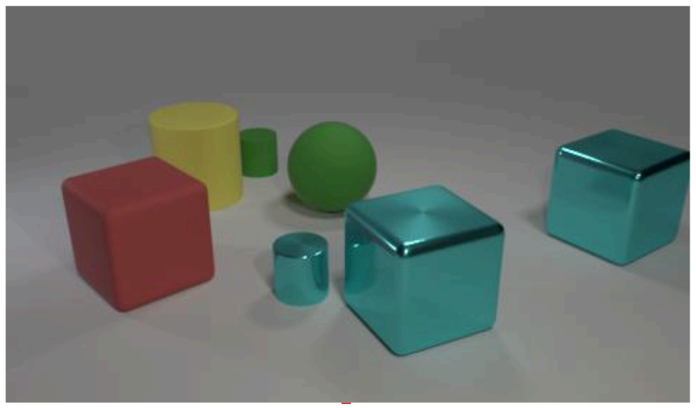


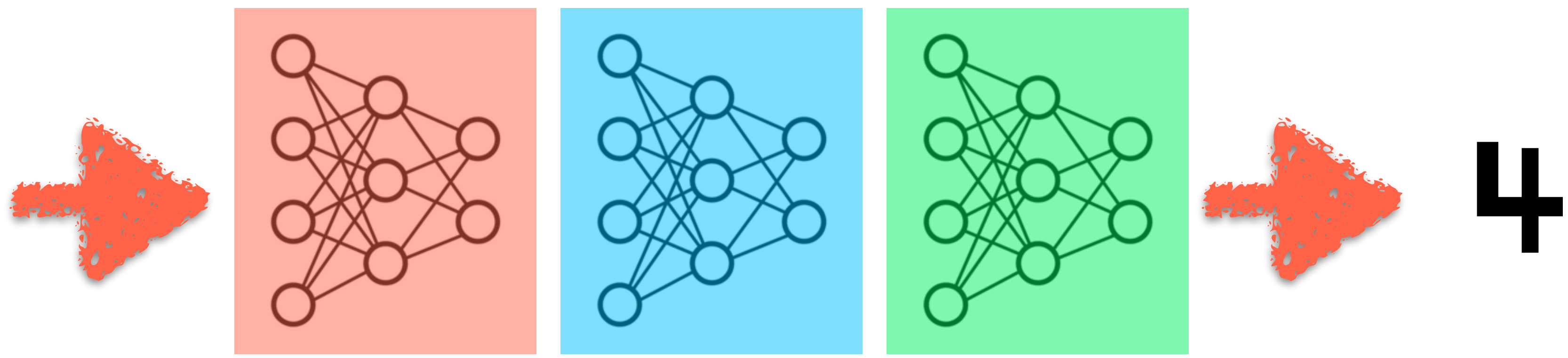
3

실제 답은 4

# Motivation

Neural Module Networks

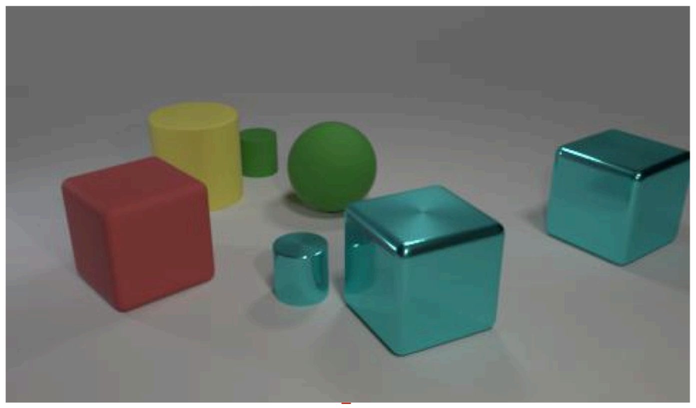


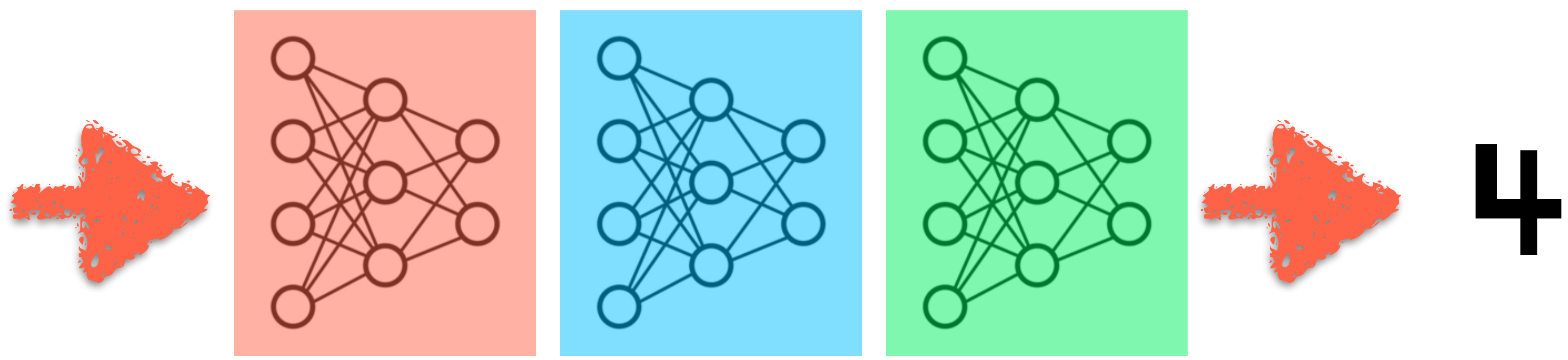



How many other things are of the same size as the green matte ball?

# Motivation

Neural Module Networks






How many other things are of the same size as the green matte ball?

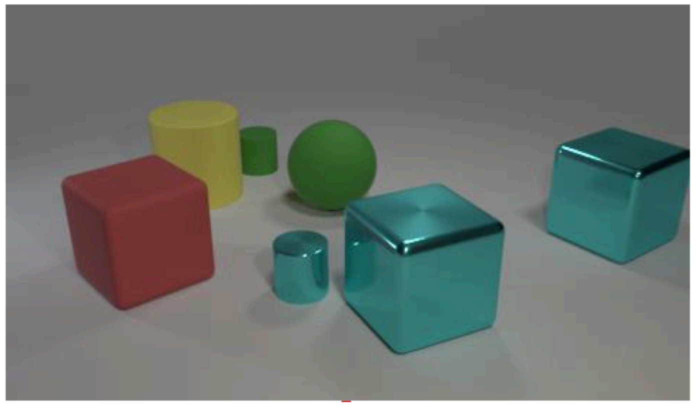
질문에 따라 Neural Module로 Network를 구성하자

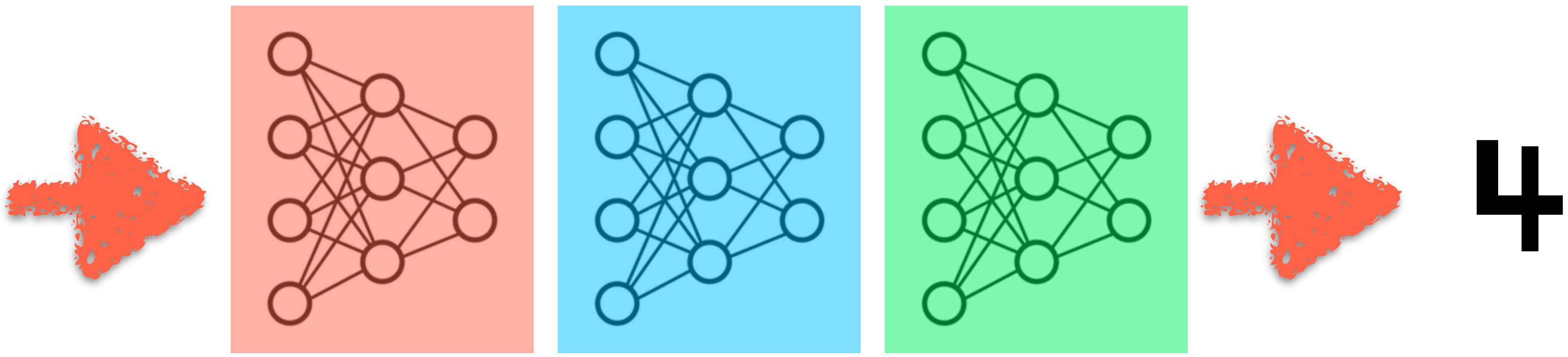
그러면 어떻게 구성할까?

policy: p( Layout | Question )로 Layout 예측

# Motivation

Neural Module Networks



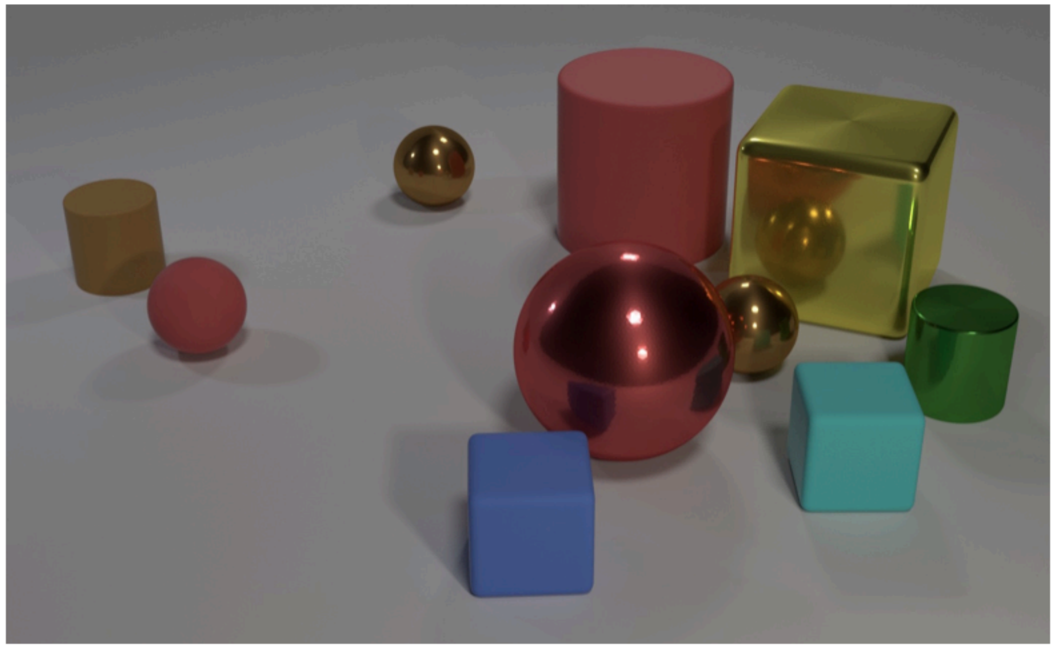



How many other things are of the same size as the green matte ball?

**End-to-end neural network: Layout Policy + NMN 함께 학습**

# Dataset: Clevr

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?

이미지: 100,000

질문: 1,000,000

Train(70%)/Val(15%)/Test(15%)

# Dataset: Clevr

jupyter answers_clevr.txt

File Edit View Language

```
0
1
10
2
3
4
5
6
7
8
9
blue
brown
cube
cyan
cylinder
gray
green
large
metal
no
purple
red
rubber
small
sphere
yellow
yes
```

jupyter vocabulary_clevr.txt

File Edit View Language

```
matte
metal
metallic
more
number
object
objects
of
on
or
other
purple
red
right
rubber
same
shape
shiny
side
size
small
sphere
spheres
than
that
the
there
thing
things
tiny
to
visible
what
yellow
```

jupyter vocabulary_layout.txt

File Edit View Language

```
_Scene
_Find
_Filter
_FindSameProperty
_Transform
_And
_Or
_Count
_Exist
_EqualNum
_MoreNum
_LessNum
_SameProperty
_Describe
<eos>
```

질문을 해봐요!

# Neural Modules

```python
class FindModule(nn.Module):
    '''
    Mapping image_feat_grid X text_param ->att.grid
    (N,D_image,H,W) X (N,1,D_text) --> [N,1,H,W]
    '''
    def __init__(self, image_dim, text_dim, map_dim):
        super(FindModule,self).__init__()
        self.map_dim = map_dim
        self.conv1 = nn.Conv2d(image_dim,map_dim,kernel_size=1)
        self.conv2 = nn.Conv2d(map_dim, 1, kernel_size=1)
        self.textfc = nn.Linear(text_dim,map_dim)

    def forward(self, input_image_feat, input_text, input_image_attention1=None, input_image_attention2=None):
        image_mapped = self.conv1(input_image_feat)  #(N, map_dim, H, W)
        text_mapped = self.textfc(input_text).view(-1, self.map_dim,1,1).expand_as(image_mapped)
        elmtwize_mult = image_mapped * text_mapped
        elmtwize_mult = F.normalize(elmtwize_mult, p=2, dim=1) #(N, map_dim, H, W)
        att_grid = self.conv2(elmtwize_mult) #(N, 1, H, W)
        return att_grid
```

| Module name | Att-inputs | Features | Output | Implementation details |
|---|---|---|---|---|
| find | (none) | $x_{vis}, x_{txt}$ | att | $a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W x_{txt}\right)$ |
| relocate | $a$ | $x_{vis}, x_{txt}$ | att | $a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W_1 \text{sum}(a \odot x_{vis}) \odot W_2 x_{txt}\right)$ |
| and | $a_1, a_2$ | (none) | att | $a_{out} = \text{minimum}(a_1, a_2)$ |
| or | $a_1, a_2$ | (none) | att | $a_{out} = \text{maximum}(a_1, a_2)$ |
| filter | $a$ | $x_{vis}, x_{txt}$ | att | $a_{out} = \texttt{and}(a, \texttt{find}[x_{vis}, x_{txt}]())$, *i.e.* reusing `find` and `and` |
| [exist, count] | $a$ | (none) | ans | $y = W^T \text{vec}(a)$ |
| describe | $a$ | $x_{vis}, x_{txt}$ | ans | $y = W_1^T \left(W_2 \text{sum}(a \odot x_{vis}) \odot W_3 x_{txt}\right)$ |
| [eq_count, more, less] | $a_1, a_2$ | (none) | ans | $y = W_1^T \text{vec}(a_1) + W_2^T \text{vec}(a_2)$ |
| compare | $a_1, a_2$ | $x_{vis}, x_{txt}$ | ans | $y = W_1^T \left(W_2 \text{sum}(a_1 \odot x_{vis}) \odot W_3 \text{sum}(a_2 \odot x_{vis}) \odot W_4 x_{txt}\right)$ |

# Neural Modules

```python
class AndModule(nn.Module):
    def __init__(self):
        super(AndModule,self).__init__()

    def forward(self, input_image_feat, input_text, input_image_attention1=None, input_image_attention2=None):
        return torch.max(input_image_attention1, input_image_attention2)


class OrModule(nn.Module):
    def __init__(self):
        super(OrModule,self).__init__()
    def forward(self, input_image_feat, input_text, input_image_attention1=None, input_image_attention2=None):
        return torch.min(input_image_attention1, input_image_attention2)
```

| Module name | Att-inputs | Features | Output | Implementation details |
|---|---|---|---|---|
| `find` | (none) | $x_{vis}, x_{txt}$ | att | $a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W x_{txt}\right)$ |
| `relocate` | $a$ | $x_{vis}, x_{txt}$ | att | $a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W_1 \text{sum}(a \odot x_{vis}) \odot W_2 x_{txt}\right)$ |
| `and` | $a_1, a_2$ | (none) | att | $a_{out} = \text{minimum}(a_1, a_2)$ |
| `or` | $a_1, a_2$ | (none) | att | $a_{out} = \text{maximum}(a_1, a_2)$ |
| `filter` | $a$ | $x_{vis}, x_{txt}$ | att | $a_{out} = \texttt{and}(a, \texttt{find}[x_{vis}, x_{txt}]())$, *i.e.* reusing `find` and `and` |
| `[exist, count]` | $a$ | (none) | ans | $y = W^T \text{vec}(a)$ |
| `describe` | $a$ | $x_{vis}, x_{txt}$ | ans | $y = W_1^T \left(W_2 \text{sum}(a \odot x_{vis}) \odot W_3 x_{txt}\right)$ |
| `[eq_count, more, less]` | $a_1, a_2$ | (none) | ans | $y = W_1^T \text{vec}(a_1) + W_2^T \text{vec}(a_2)$ |
| `compare` | $a_1, a_2$ | $x_{vis}, x_{txt}$ | ans | $y = W_1^T \left(W_2 \text{sum}(a_1 \odot x_{vis}) \odot W_3 \text{sum}(a_2 \odot x_{vis}) \odot W_4 x_{txt}\right)$ |

# Neural Modules

```python
class FilterModule(nn.Module):
    def __init__(self, findModule, andModule):
        super(FilterModule,self).__init__()
        self.andModule = andModule
        self.findModule = findModule

    def forward(self, input_image_feat, input_text, input_image_attention1=None, input_image_attention2=None):
        find_result = self.findModule(input_image_feat,input_text,input_image_attention1,input_image_attention2)
        att_grid = self.andModule(input_image_feat,input_text,input_image_attention1,find_result)
        return att_grid
```

| Module name | Att-inputs | Features | Output | Implementation details |
|---|---|---|---|---|
| `find` | (none) | $x_{vis}, x_{txt}$ | att | $a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W x_{txt}\right)$ |
| `relocate` | $a$ | $x_{vis}, x_{txt}$ | att | $a_{out} = \text{conv}_2\left(\text{conv}_1(x_{vis}) \odot W_1 \text{sum}(a \odot x_{vis}) \odot W_2 x_{txt}\right)$ |
| `and` | $a_1, a_2$ | (none) | att | $a_{out} = \text{minimum}(a_1, a_2)$ |
| `or` | $a_1, a_2$ | (none) | att | $a_{out} = \text{maximum}(a_1, a_2)$ |
| `filter` | $a$ | $x_{vis}, x_{txt}$ | att | $a_{out} = \text{and}(a, \texttt{find}[x_{vis}, x_{txt}]())$, *i.e.* reusing `find` and `and` |
| `[exist, count]` | $a$ | (none) | ans | $y = W^T \text{vec}(a)$ |
| `describe` | $a$ | $x_{vis}, x_{txt}$ | ans | $y = W_1^T \left(W_2 \text{sum}(a \odot x_{vis}) \odot W_3 x_{txt}\right)$ |
| `[eq_count, more, less]` | $a_1, a_2$ | (none) | ans | $y = W_1^T \text{vec}(a_1) + W_2^T \text{vec}(a_2)$ |
| `compare` | $a_1, a_2$ | $x_{vis}, x_{txt}$ | ans | $y = W_1^T \left(W_2 \text{sum}(a_1 \odot x_{vis}) \odot W_3 \text{sum}(a_2 \odot x_{vis}) \odot W_4 x_{txt}\right)$ |

# End-to-End Neural Module Network (N2NMN)

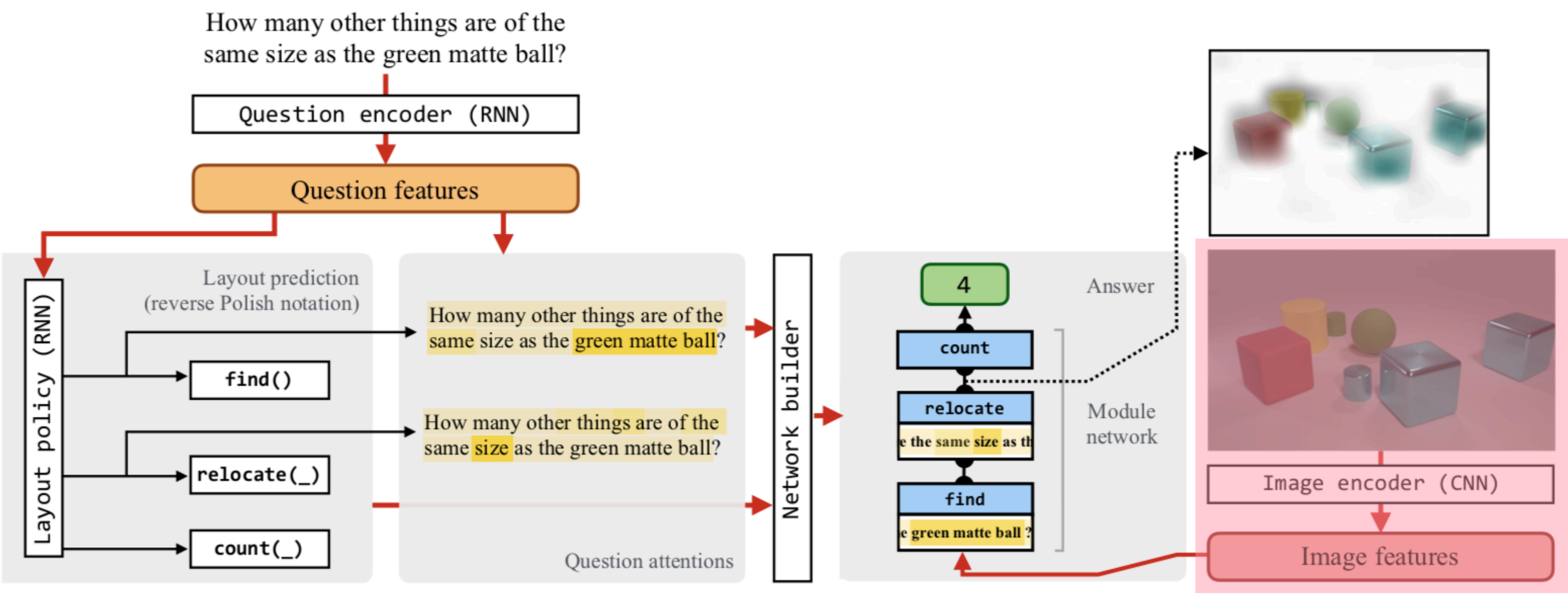
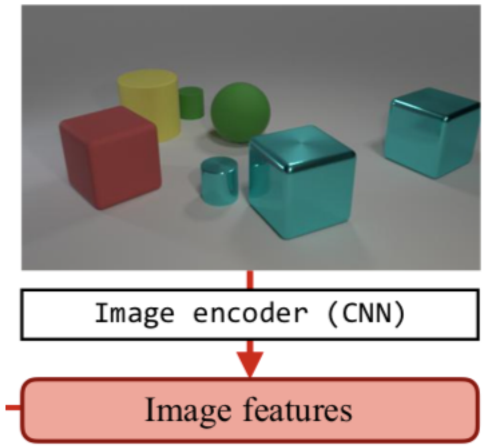



image feature extraction

질문을 해봐요!

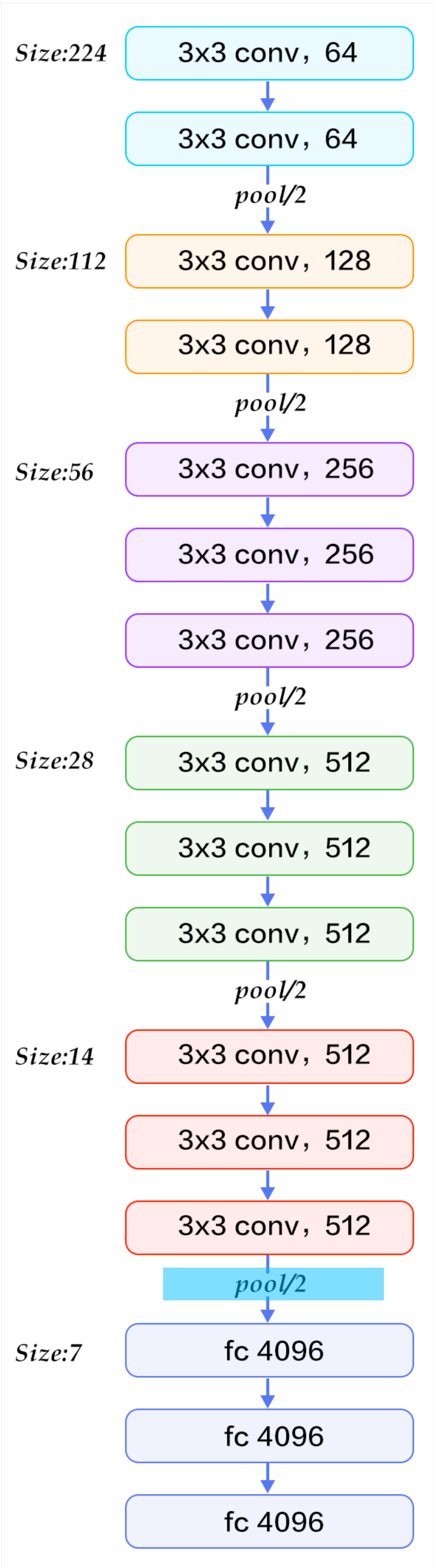
# Image Feature Extraction




## Image feature VGG-16

## TF code

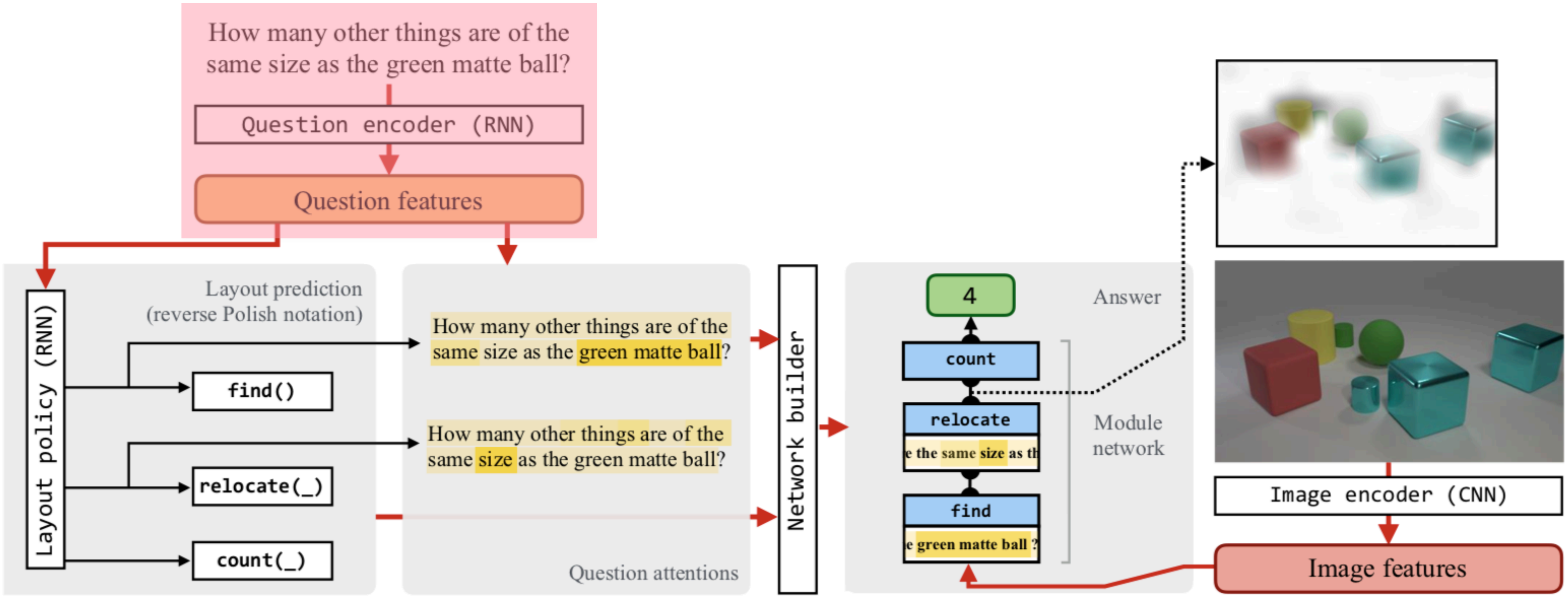
```
# layer 2
conv2_1 = conv_relu('conv2_1', pool1,
                    kernel_size=3, stride=1, output_dim=128)
conv2_2 = conv_relu('conv2_2', conv2_1,
                    kernel_size=3, stride=1, output_dim=128)
pool2 = pool('pool2', conv2_2, kernel_size=2, stride=2)
# layer 3
conv3_1 = conv_relu('conv3_1', pool2,
                    kernel_size=3, stride=1, output_dim=256)
conv3_2 = conv_relu('conv3_2', conv3_1,
                    kernel_size=3, stride=1, output_dim=256)
conv3_3 = conv_relu('conv3_3', conv3_2,
                    kernel_size=3, stride=1, output_dim=256)
pool3 = pool('pool3', conv3_3, kernel_size=2, stride=2)
# layer 4
conv4_1 = conv_relu('conv4_1', pool3,
                    kernel_size=3, stride=1, output_dim=512)
conv4_2 = conv_relu('conv4_2', conv4_1,
                    kernel_size=3, stride=1, output_dim=512)
conv4_3 = conv_relu('conv4_3', conv4_2,
                    kernel_size=3, stride=1, output_dim=512)
pool4 = pool('pool4', conv4_3, kernel_size=2, stride=2)
# layer 5
conv5_1 = conv_relu('conv5_1', pool4,
                    kernel_size=3, stride=1, output_dim=512)
conv5_2 = conv_relu('conv5_2', conv5_1,
                    kernel_size=3, stride=1, output_dim=512)
conv5_3 = conv_relu('conv5_3', conv5_2,
                    kernel_size=3, stride=1, output_dim=512)
pool5 = pool('pool5', conv5_3, kernel_size=2, stride=2)
return pool5
```



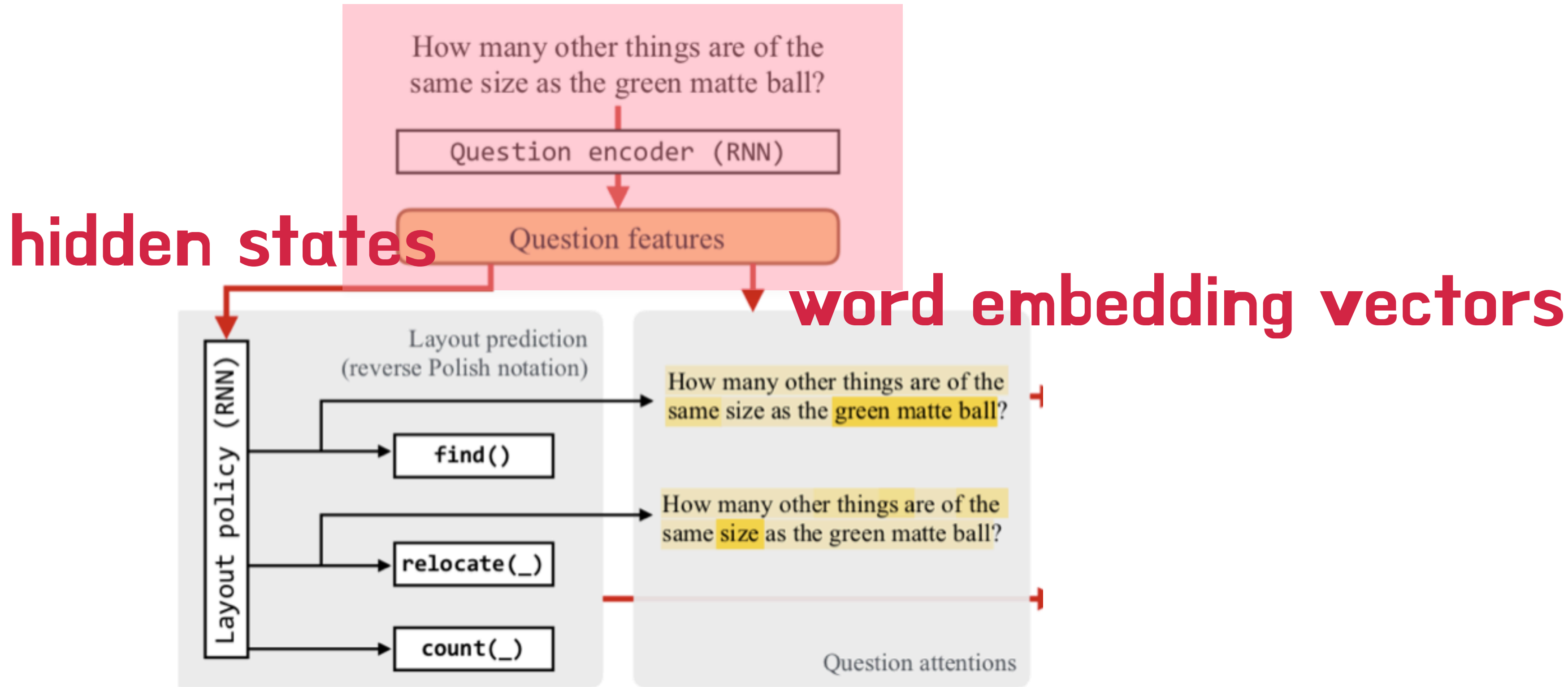

질문을 해봐요!

# End-to-End Neural Module Network (N2NMN)

## question feature extraction (encoding)



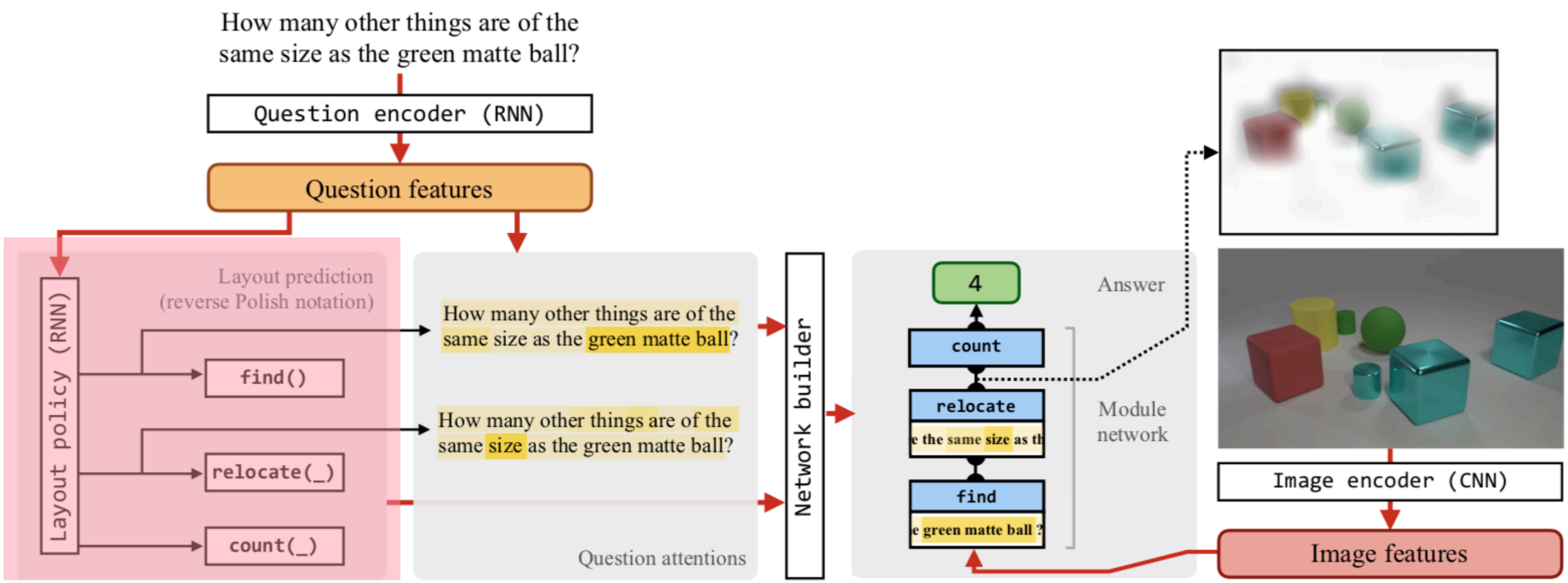
# Question Feature Extraction




```python
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size, input_encoding_size, num_layers=2):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.embedding = nn.Embedding(input_size, input_encoding_size)
        self.lstm = nn.LSTM(input_encoding_size, hidden_size)

    def forward(self, input_seqs, input_seq_lens, hidden):
        embedded = self.embedding(input_seqs)
        outputs, hidden = self.lstm(embedded)
        return outputs, hidden, embedded

    def initHidden(self, batch_size):
        result = torch.zeros(self.num_layers, batch_size, self.hidden_size)
        return result
```

**cf) LSTM 코드 구조**
- **outputs: 모든 time step에서 hidden states**
- **hidden: 마지막 time step에서 hidden state**

# End-to-End Neural Module Network (N2NMN)




layout prediction (decoding)

# Layout expression

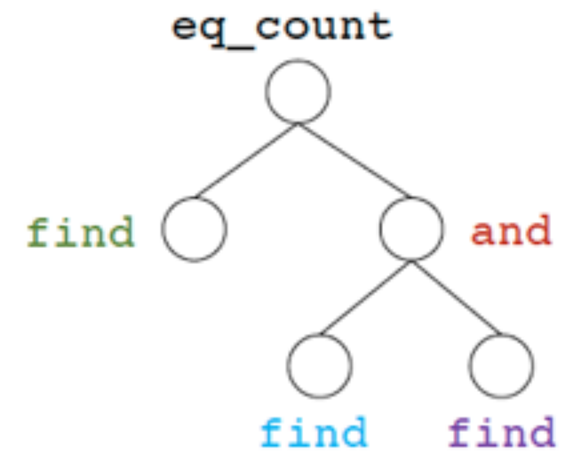
How many other things are of the same size as the green matte ball?

functional expression

**count(relocate(find))**

layout expression `eq_count(find(), and(find(), find()))`

syntax tree




Reverse Polish Notation `[find, find, find, and, eq_count]`

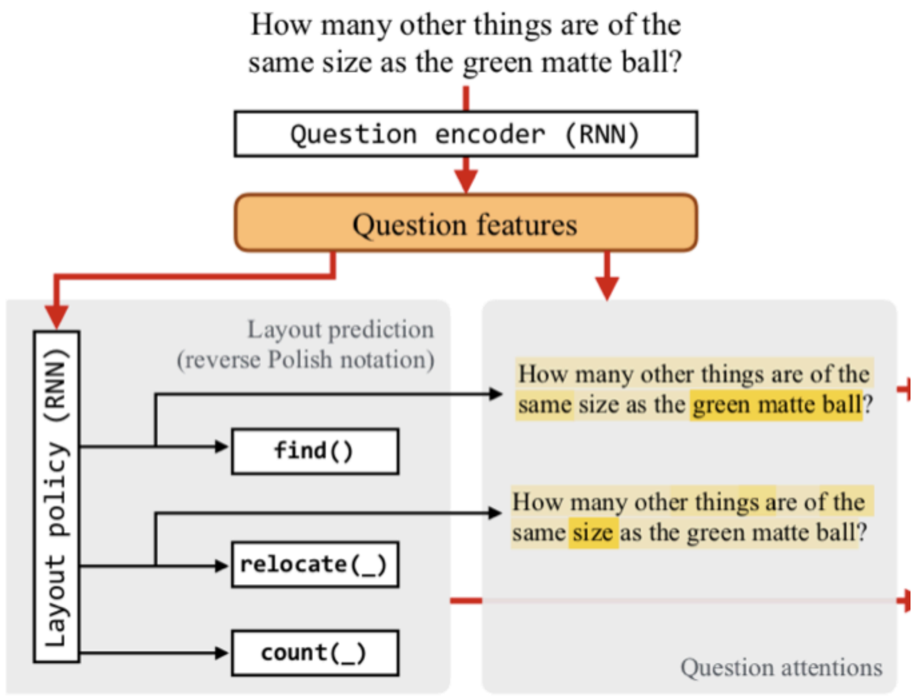
**NM Layout 표기법**

**NM을 network로 assemble하는 순서**

**따라서, Seq-to-seq network로 <eos>까지 NM token을 예측하여 assemble하자!**

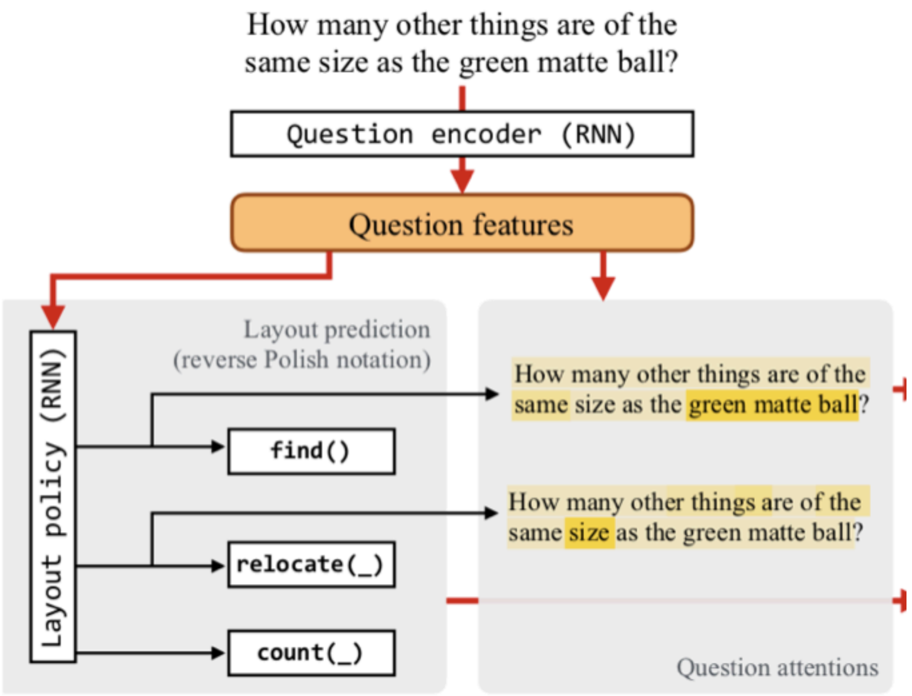
| Translation | | NM token 예측 |
|---|---|---|
| English token -> French token | ↔ | English token -> NM token |

# End-to-End Neural Module Network (N2NMN)



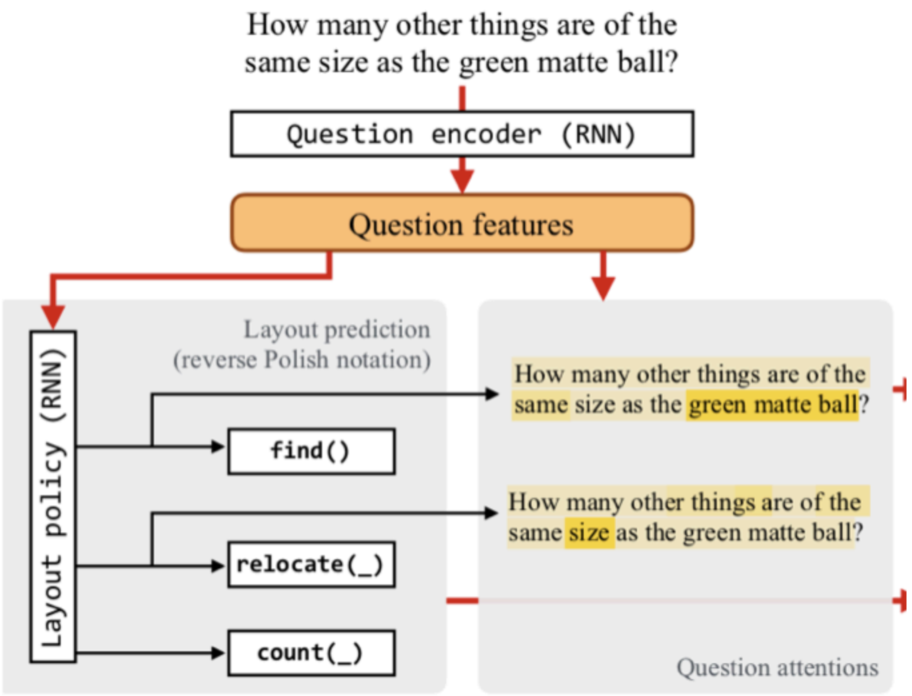

**Seq-to-seq net with attention 구조는 코드와 함께 보며 칠판에^^**

# End-to-End Neural Module Network (N2NMN)



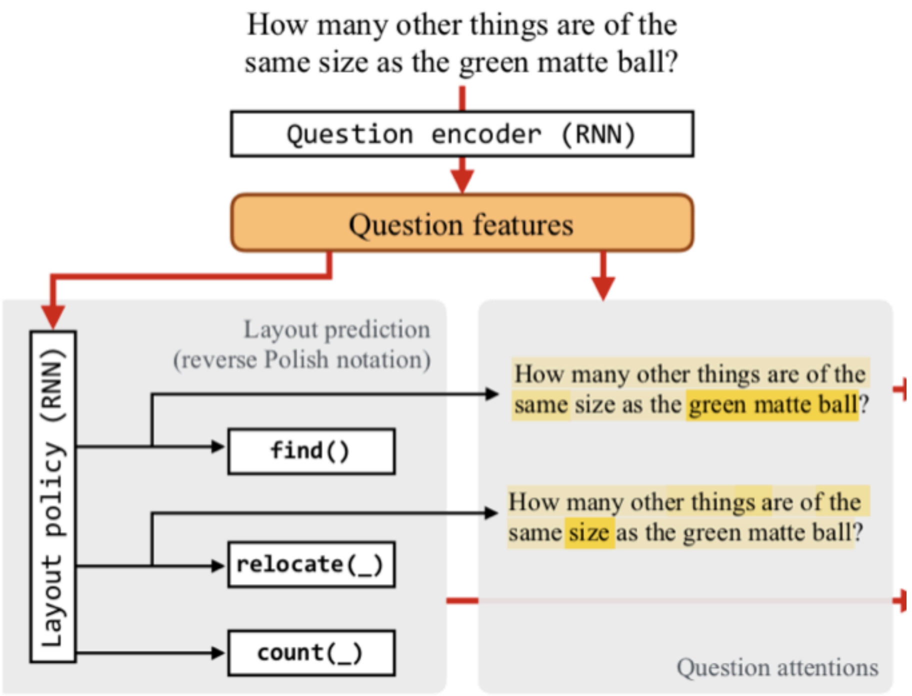

```
class LuongAttnDecoderRNN(nn.Module):
    def __init__(self, attn_model, embedding, hidden_size, output_size, n_layers=1,
dropout=0.1):
        super(LuongAttnDecoderRNN, self).__init__()

        # Keep for reference
        self.attn_model = attn_model
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout = dropout

        # Define layers
        self.embedding = embedding
        self.embedding_dropout = nn.Dropout(dropout)
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers, dropout=(0 if n_layers == 1
else dropout))
        self.concat = nn.Linear(hidden_size * 2, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

        self.attn = Attn(attn_model, hidden_size)
```

# End-to-End Neural Module Network (N2NMN)




```python
    def forward(self, input_step, last_hidden, encoder_outputs):
        # Note: we run this one step (word) at a time
        # Get embedding of current input word
        embedded = self.embedding(input_step)
        embedded = self.embedding_dropout(embedded)
        # Forward through unidirectional GRU
        rnn_output, hidden = self.gru(embedded, last_hidden)
        # Calculate attention weights from the current GRU output
        attn_weights = self.attn(rnn_output, encoder_outputs)
        # Multiply attention weights to encoder outputs to get new "weighted sum" context
vector
        context = attn_weights.bmm(encoder_outputs.transpose(0, 1))
        # Concatenate weighted context vector and GRU output using Luong eq. 5
        rnn_output = rnn_output.squeeze(0)
        context = context.squeeze(1)
        concat_input = torch.cat((rnn_output, context), 1)
        concat_output = torch.tanh(self.concat(concat_input))
        # Predict next word using Luong eq. 6
        output = self.out(concat_output)
        output = F.softmax(output, dim=1)
        # Return output and final hidden state
        return output, hidden
```

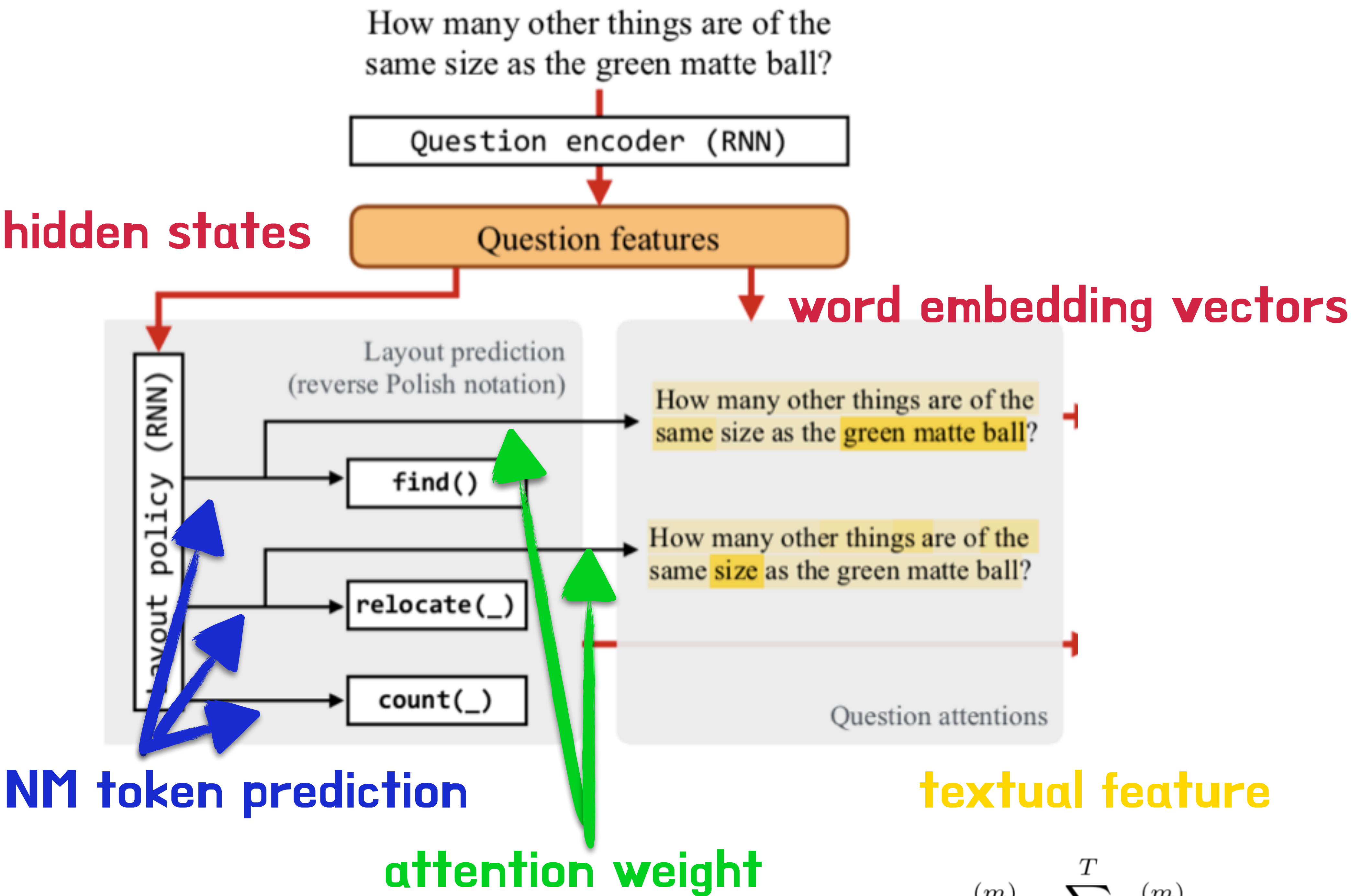
# End-to-End Neural Module Network (N2NMN)




```python
# Luong attention layer
class Attn(torch.nn.Module):
    def __init__(self, method, hidden_size):
        super(Attn, self).__init__()
        self.method = method
        if self.method not in ['dot', 'general', 'concat']:
            raise ValueError(self.method, "is not an appropriate attention method.")
        self.hidden_size = hidden_size
        if self.method == 'general':
            self.attn = torch.nn.Linear(self.hidden_size, hidden_size)
        elif self.method == 'concat':
            self.attn = torch.nn.Linear(self.hidden_size * 2, hidden_size)
            self.v = torch.nn.Parameter(torch.FloatTensor(hidden_size))

    def dot_score(self, hidden, encoder_output):
        return torch.sum(hidden * encoder_output, dim=2)

    def general_score(self, hidden, encoder_output):
        energy = self.attn(encoder_output)
        return torch.sum(hidden * energy, dim=2)

    def concat_score(self, hidden, encoder_output):
        energy = self.attn(torch.cat((hidden.expand(encoder_output.size(0), -1, -1), 
encoder_output), 2)).tanh()
        return torch.sum(self.v * energy, dim=2)

    def forward(self, hidden, encoder_outputs):
        # Calculate the attention weights (energies) based on the given method
        if self.method == 'general':
            attn_energies = self.general_score(hidden, encoder_outputs)
        elif self.method == 'concat':
            attn_energies = self.concat_score(hidden, encoder_outputs)
        elif self.method == 'dot':
            attn_energies = self.dot_score(hidden, encoder_outputs)

        # Transpose max_length and batch_size dimensions
        attn_energies = attn_energies.t()

        # Return the softmax normalized probability scores (with added dimension)
        return F.softmax(attn_energies, dim=1).unsqueeze(1)
```

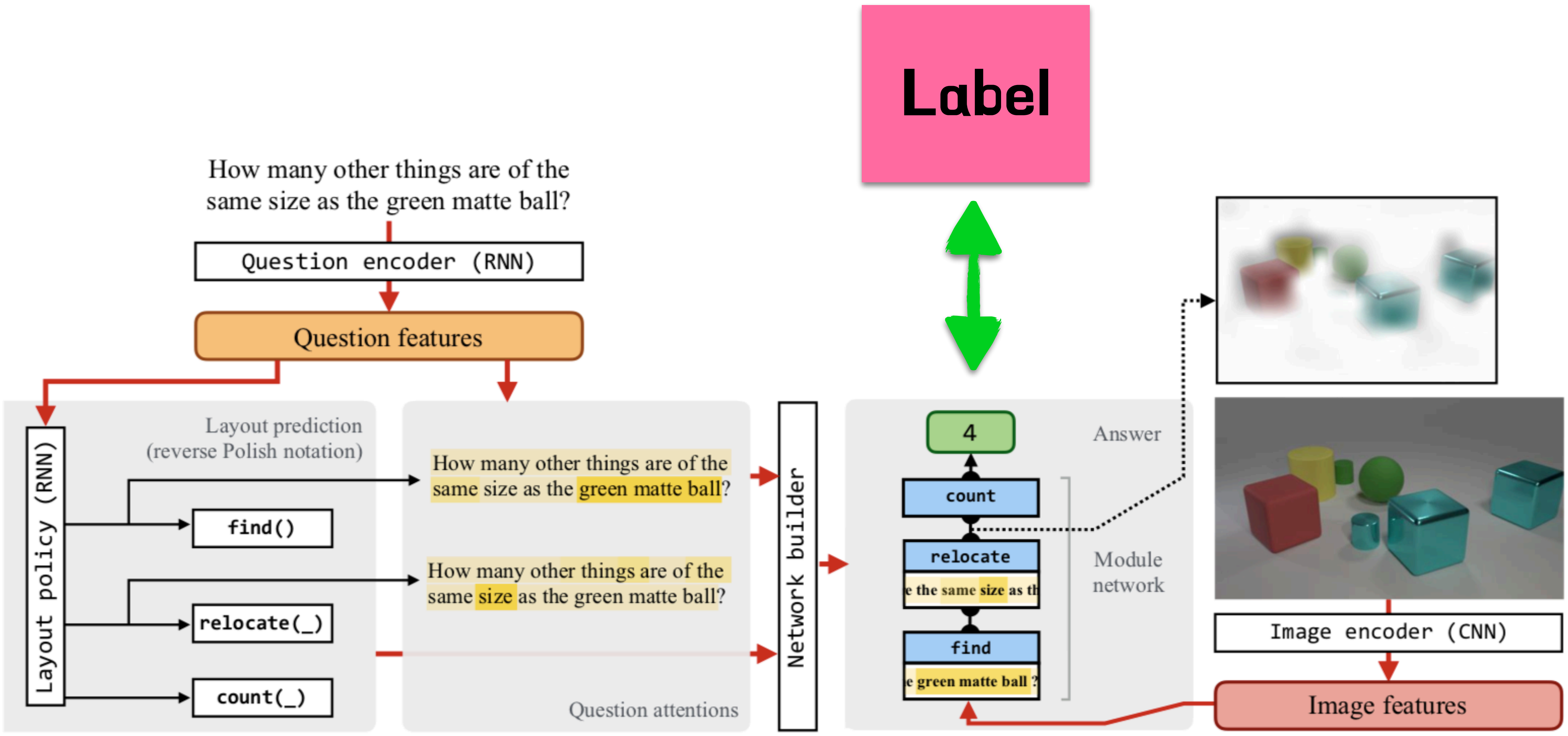
질문을 해봐요!

# NM token prediction + textual feature extraction




$$x_{txt}^{(m)} = \sum_{i=1}^{T} \alpha_i^{(m)} w_i$$

# End-to-end training




$$L(\theta) = E_{l \sim p(l|q;\theta)}[\tilde{L}(\theta, l; q, I)]$$

$$\nabla_\theta L \approx \frac{1}{M} \sum_{m=1}^{M} \left( \tilde{L}(\theta, l_m) \nabla_\theta \log p(l_m | q; \theta) + \nabla_\theta \tilde{L}(\theta, l_m) \right)$$

# 끝!!

$$\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$