Combined Convolutional and Recurrent Neural Networks for Hierarchical Classification of Images

Jaehoon Koo, Diego Klabjan, and Jean Utke

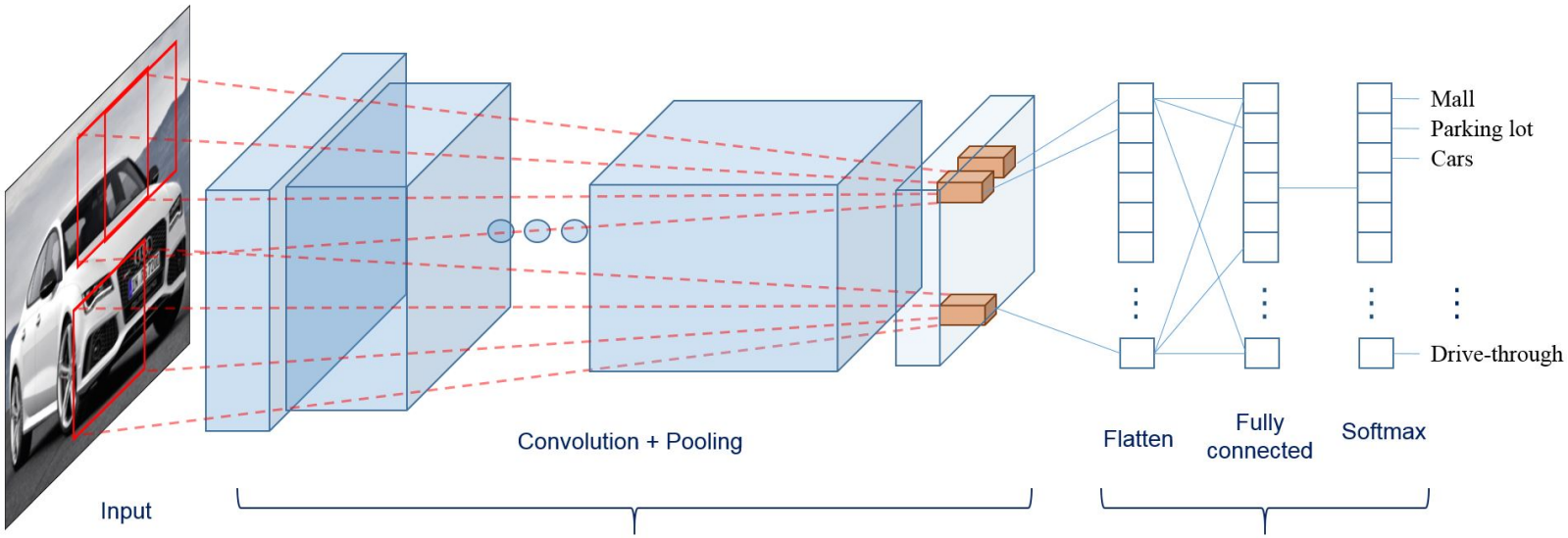
IEEE Bigdata 2020

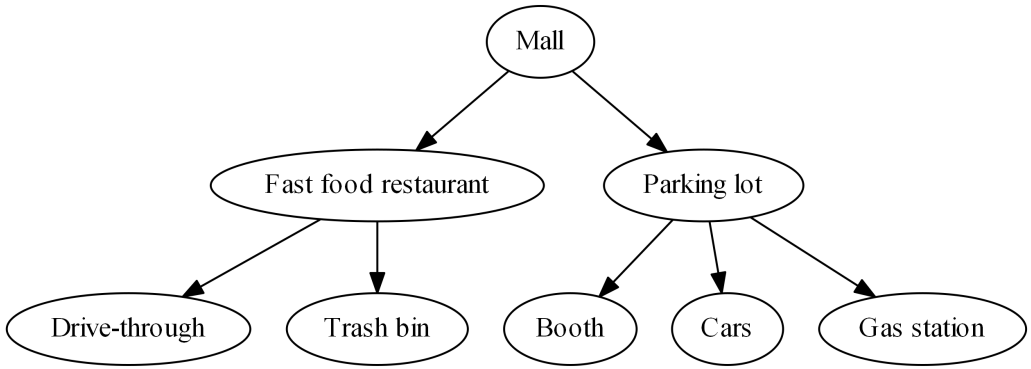
December 10-13, 2020

NORTHWESTERN
UNIVERSITY

Motivation

- Existing CNNs are trained as flat classifiers [Yan et al., 2015; Hu et al., 2016]
- Hierarchical relationships among object categories

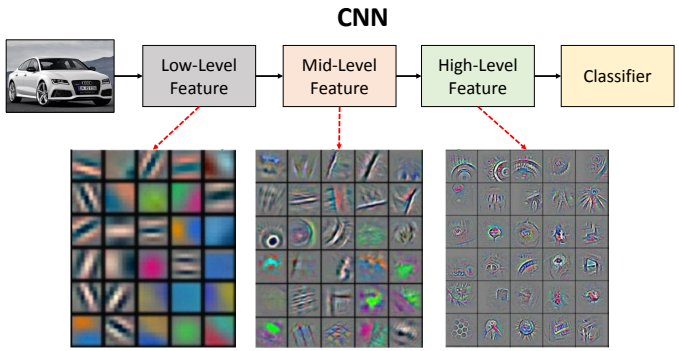




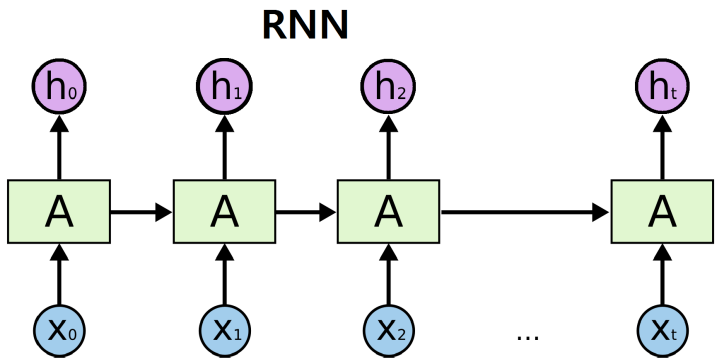
Deep Hierarchical Neural Network (DHNN)

A combined model for hierarchical classification of images

- CNN: feature learning
  - extract hierarchical representations of images

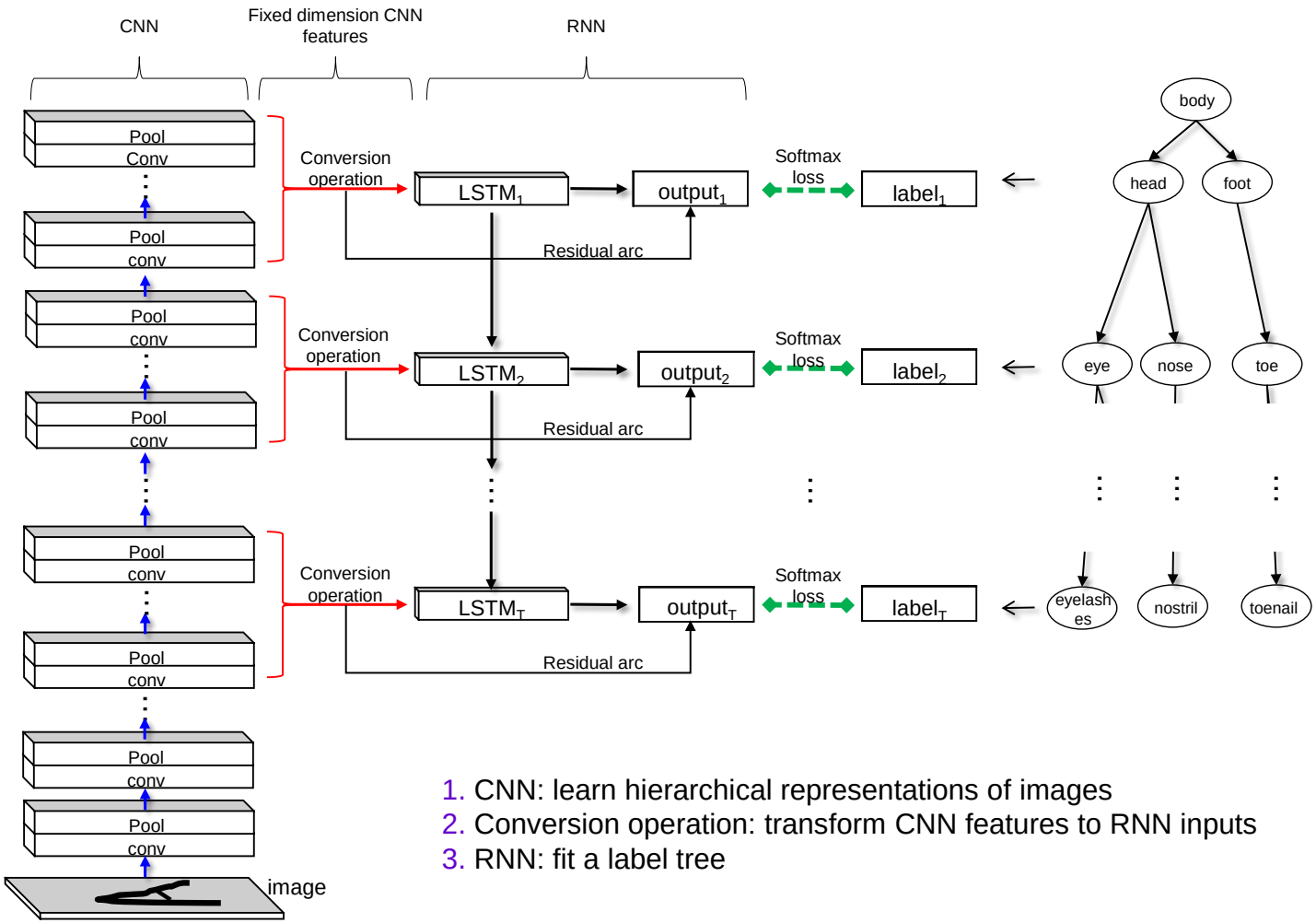


- RNN: output sequence learning
  - fit a hierarchical tree of label paths



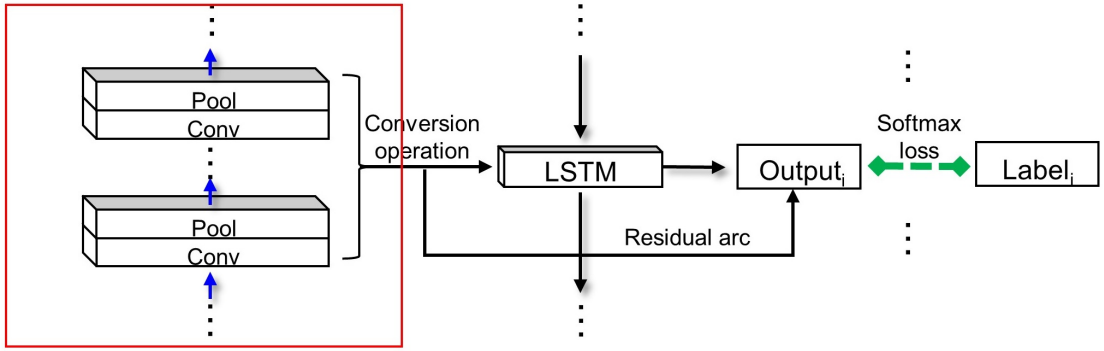
[Zeiler & Fergus, 2013] Intro-CNN Intro-RNN

DHNN Structure



1. CNN: learn hierarchical representations of images
2. Conversion operation: transform CNN features to RNN inputs
3. RNN: fit a label tree

DHNN - CNN

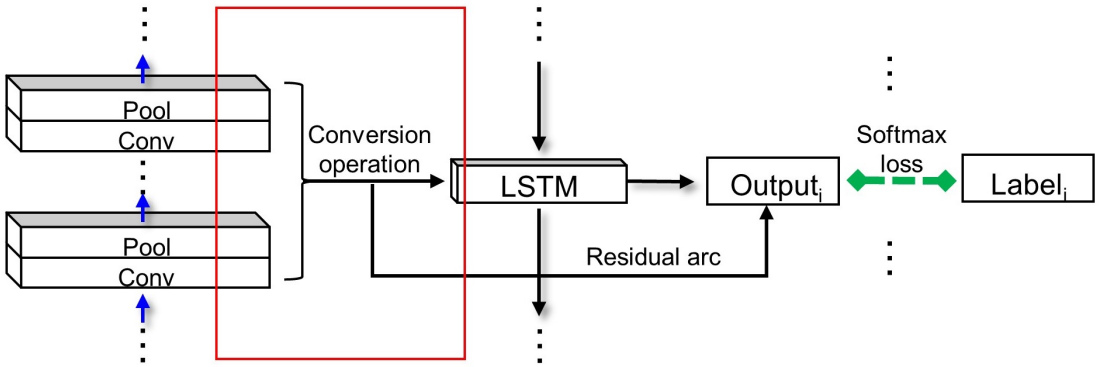


- A network of L convolution-pooling layers (CNN):

$$a_l = f\left(a_{l-1}, \phi_l\right) \quad \text{for } l = 1, 2, \ldots, L$$

  - $a_l \in \mathbb{R}^{D_l \times W_l \times H_l}$: input at l^{th}-layer
  - f: the convolution-pooling function
  - ϕ_l: model parameters at l^{th}-layer

DHNN - Conversion Operations

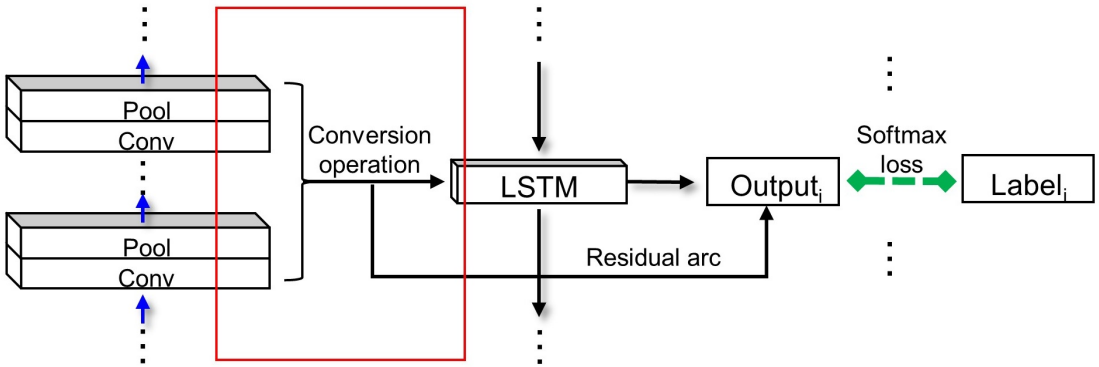


- Conversion of CNN features, a_s, fed to RNN:

$$u_t = \frac{1}{|S_t|} \sum_{s \in S_t} g(a_s) \quad \text{for } t = 1, 2, \ldots, T$$

  - p: RNN input dimension
  - $g_p : \mathbb{R}^{D_s \times W_s \times H_s} \rightarrow \mathbb{R}^p$: a function of converting CNN outputs into RNN inputs
  - T: depth of a label tree
  - S_t: a subset of CNN layers at each step t of RNN

DHNN - Conversion Operations

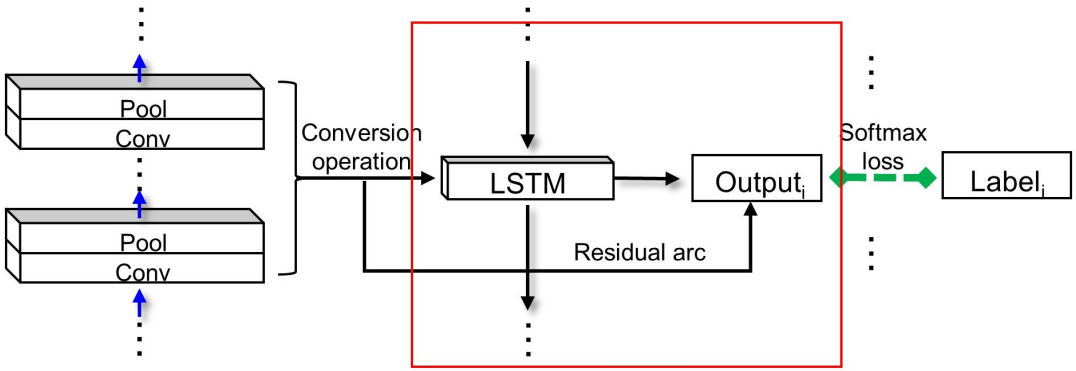


- Conversion of CNN features, a_s, fed to RNN:

Conversions	Remarks
Linear	n-mode product of a tensor, additional trainable weights
Convolutional	Keep spatial information, additional trainable weights
Pooling	Keep spatial information, no trainable weights

Three-Conversion

DHNN - RNN

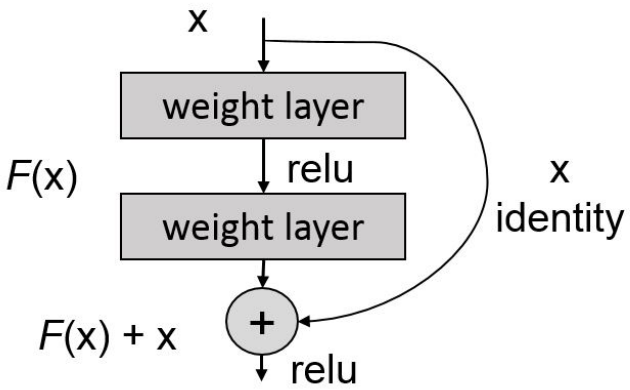


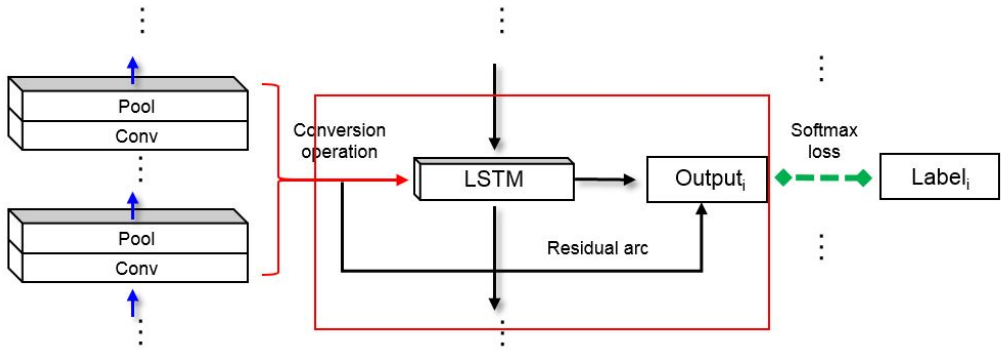
- A network of J RNN layers (LSTM):

$$h_t = r_h\left(u_t, h_{t-1}, \theta_h\right) \text{ and,}$$
$$o_t = r_o\left(h_t, \theta_o\right) \quad \text{for } t = 1, 2, \ldots, T$$

  - h_0: initial hidden state
  - r_h & r_o: a state transition and an output function
  - θ_h & θ_o: related model parameters

DHNN - Residual Arc



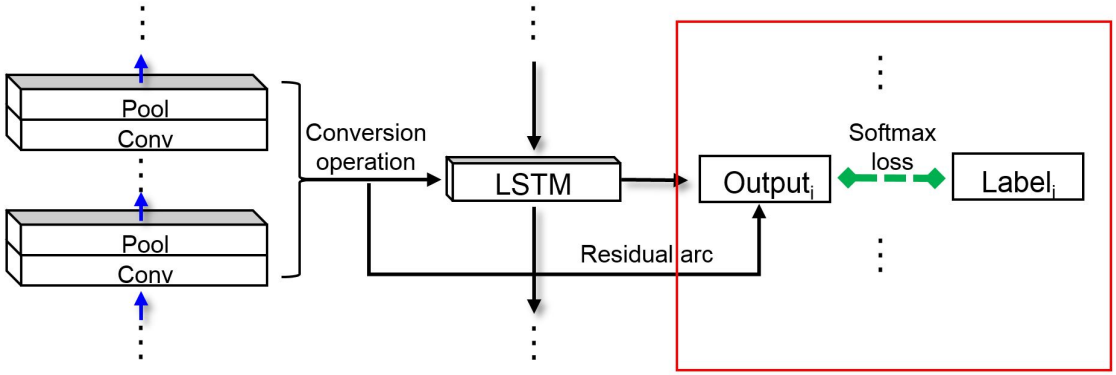


- Outputs of the RNN with residual arc:

$$o_t^{\text{residual}} = u_t + z\left(o_t; \xi\right) \quad \text{for } t = 1, 2, \ldots, T$$

  - z: a linear mapping function to align dimensions
  - ξ: trainable parameters

DHNN - Training and Inference



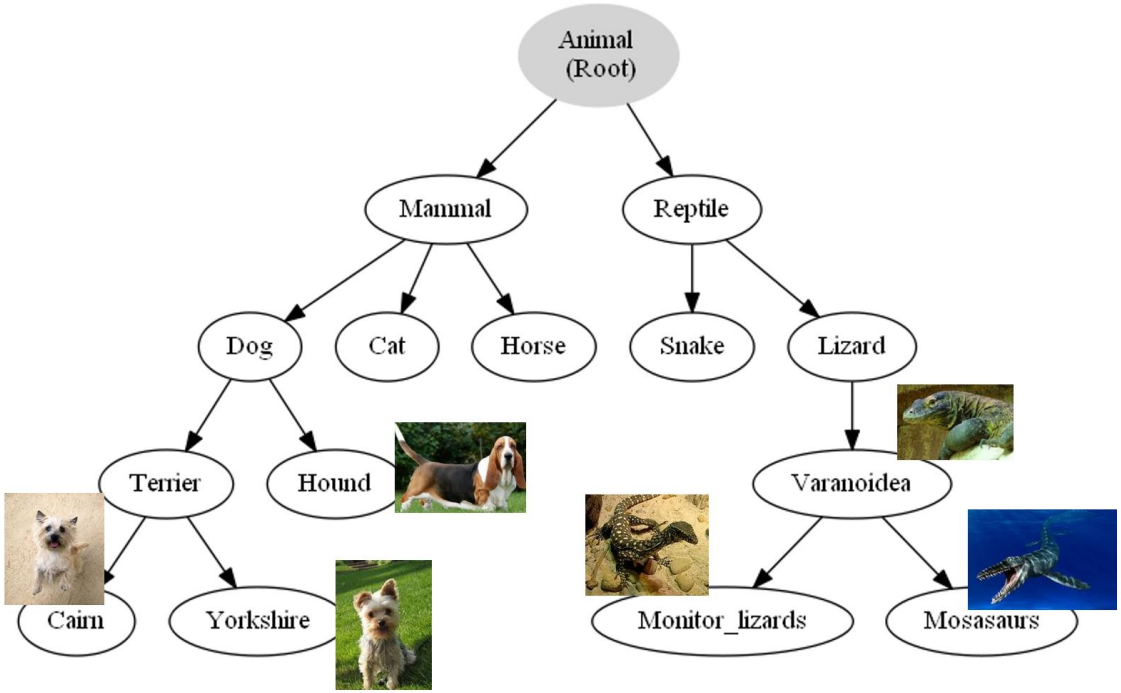
- In training:

$$\min \quad \sum_{t=1}^{T} w_t \; CE(y_t || \text{softmax}(o_t))$$

  - CE: cross-entropy
  - w_t: weight for level t
- In inference:
  - Beam search to find the true path in the tree Beam Encoding

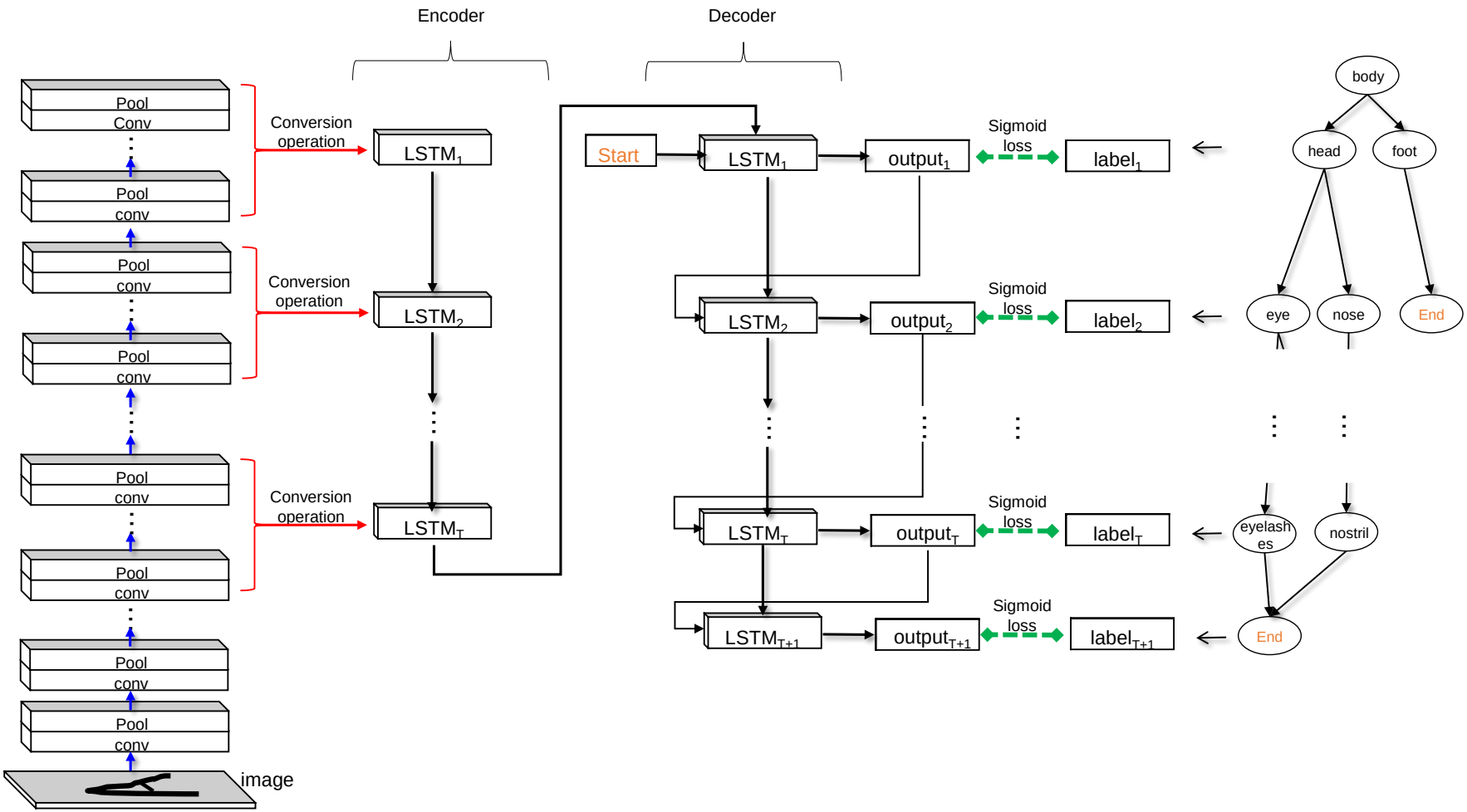
DHNN - General Tree Model

In case, a length of paths is different



- Need to deal with different lengths of input and output sequences
- Sequence to Sequence RNN! S2S

DHNN: CNN-S2S


Encoder
Decoder
Pool
Conv
Pool
conv
Pool
conv
Pool
conv
Pool
conv
Pool
conv
Pool
conv
image
Conversion operation
Conversion operation
Conversion operation
LSTM1
LSTM2
LSTMT
Start
LSTM1
output1
Sigmoid loss
label1
LSTM2
output2
Sigmoid loss
label2
LSTMT
outputT
Sigmoid loss
labelT
LSTMT+1
outputT+1
Sigmoid loss
labelT+1
body
head
foot
eye
nose
End
eyelash es
nostril
End

Computational Study

- Datasets
  - A public dataset, Open Images (OI): 1M images [Google Open Source]
- Architecture
  - CNN:
    - Visual Geometry Group (VGG) [Simonyan et al., 2014]
    - Residual neural networks (Res) [He et al., 2015]
    - Squeeze-and-Excitation networks (SE) [He et al., 2018]
    - Convolutional Block Attention Module (CBAM)[Woo et al., 2018]
  - RNN: a bidirectional RNN with LSTM cells
- Implementation
  - Python Tensorflow
  - Adam optimizer
  - OI: 4-5 days for 15 epochs on NVIDIA Titan XP

Computational Study

- General tree model (OI): CNN-S2S

Method	Mean F1 score and Sd. (%)	
	Path	Node
VGG-16	68.62 (0.09)	72.29 (0.09)
VGG-S2S-Linear	**71.49 (0.10)**	**74.84 (0.12)**
VGG-S2S-Alt-Linear	70.76 (0.06)	73.97 (0.15)
VGG-S2S-Conv	68.61 (0.41)	71.62 (0.51)
VGG-S2S-Alt-Conv	70.70 (0.14)	73.95 (0.06)
VGG-S2S-Pool	71.39 (0.16)	74.76 (0.18)
VGG-S2S-Alt-Pool	**71.49 (0.10)**	**74.84 (0.07)**
VGG-S2S-Pool-$\mathcal{L}$	**71.93 (0.09)**	**75.29 (0.07)**
VGG-S2S-Alt-Pool-$\mathcal{L}$	71.26 (0.13)	74.58 (0.17)
Res-50	71.15 (0.04)	74.47 (0.07)
Res-S2S-Linear	**72.14 (0.14)**	**75.46 (0.16)**
Res-S2S-Alt-Linear	70.78 (0.09)	73.80 (0.14)
Res-S2S-Conv	70.42 (0.39)	73.55 (0.44)
Res-S2S-Alt-Conv	70.75 (0.15)	73.98 (0.18)
Res-S2S-Pool	**71.98 (0.06)**	**75.40 (0.07)**
Res-S2S-Alt-Pool	71.70 (0.06)	74.97 (0.09)
Res-S2S-Pool-$\mathcal{L}$	**72.05 (0.06)**	**75.43 (0.03)**
Res-S2S-Alt-Pool-$\mathcal{L}$	71.66 (0.04)	74.89 (0.05)

Method	Mean F1 score and Sd. (%)	
	Path	Node
SE-Res-50	71.22 (0.10)	74.49 (0.08)
SE-Res-S2S-Linear	**72.05 (0.04)**	**75.33 (0.08)**
SE-Res-S2S-Alt-Linear	69.52 (0.11)	72.45 (0.14)
SE-Res-S2S-Conv	70.49 (0.15)	73.65 (0.14)
SE-Res-S2S-Alt-Conv	70.35 (0.09)	73.51 (0.06)
SE-Res-S2S-Pool	**71.79 (0.15)**	**75.12 (0.16)**
SE-Res-S2S-Alt-Pool	71.42 (0.05)	74.73 (0.04)
SE-Res-S2S-Pool-$\mathcal{L}$	**71.75 (0.09)**	**75.07 (0.04)**
SE-Res-S2S-Alt-Pool-$\mathcal{L}$	71.07 (0.13)	74.27 (0.18)
CBAM-Res-50	71.17 (0.09)	74.42 (0.04)
CBAM-Res-S2S-Linear	**71.71 (0.19)**	**74.98 (0.19)**
CBAM-Res-S2S-Alt-Linear	66.44 (0.43)	68.84 (0.53)
CBAM-Res-S2S-Conv	70.24 (0.07)	73.43 (0.05)
CBAM-Res-S2S-Alt-Conv	72.88 (0.09)	69.82 (0.07)
CBAM-Res-S2S-Pool	**71.53 (0.12)**	**74.82 (0.10)**
CBAM-Res-S2S-Alt-Pool	71.29 (0.18)	74.55 (0.11)
CBAM-Res-S2S-Pool-$\mathcal{L}$	**71.57 (0.07)**	**74.89 (0.07)**
CBAM-Res-S2S-Alt-Pool-$\mathcal{L}$	71.32 (0.05)	74.55 (0.01)

$\mathcal{L}$ denotes models with larger S2S than others.

Exp-REAL OI-tree

Conclusion

Deep Hierarchical Neural Network

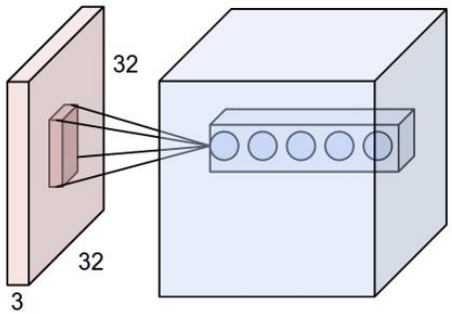
1. Predict a target path in a hierarchical tree of classes
2. Combined CNN as a feature learner with RNN or S2S as a sequence classifier
3. Fixed path length (CNN-RNN) and general tree model (CNN-S2S) for a fixed- and variable-length target path
4. For CNN-RNN, our models with residual arcs perform best
5. Recommend pooling conversion for performance and low memory requirement

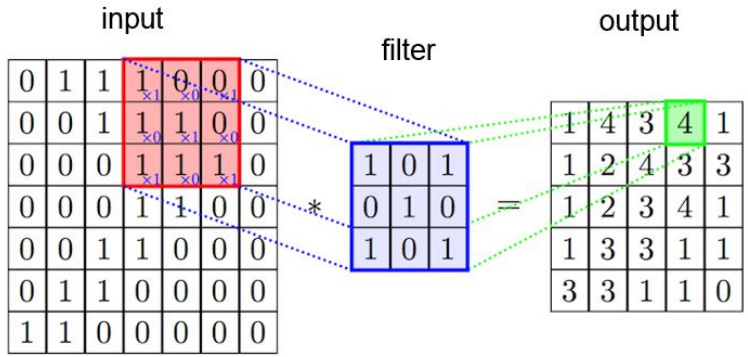
Thank you!

Questions?

Two Building Blocks of DHNN - CNN

- CNN captures local information (not fully connected)





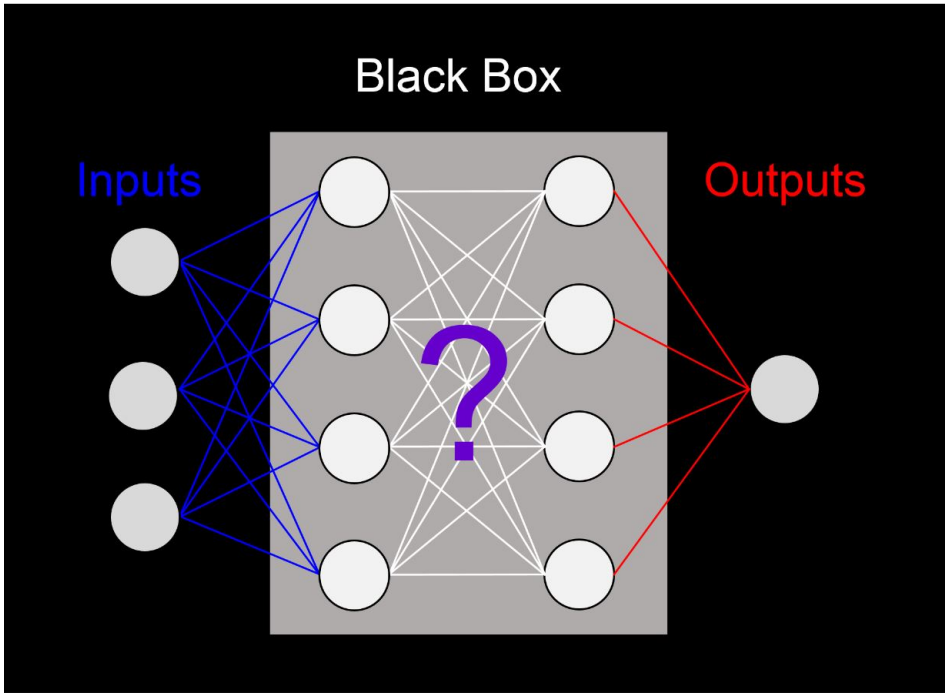
http://cs231n.github.io/convolutional-networks/

https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

Intro-CNNRNN

Two Building Blocks of DHNN - CNN

- What CNN feature learns is "black box"?

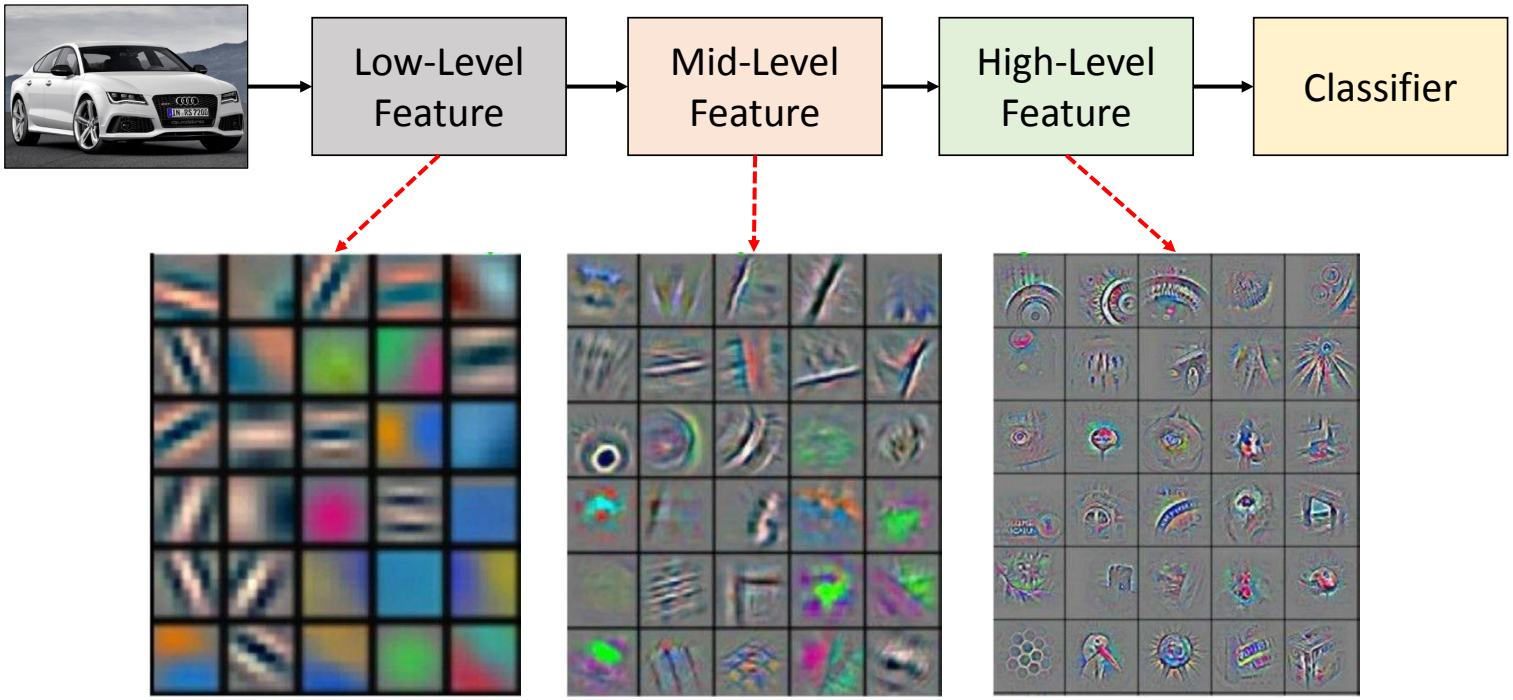


Intro-CNNRNN

Two Building Blocks of DHNN - CNN

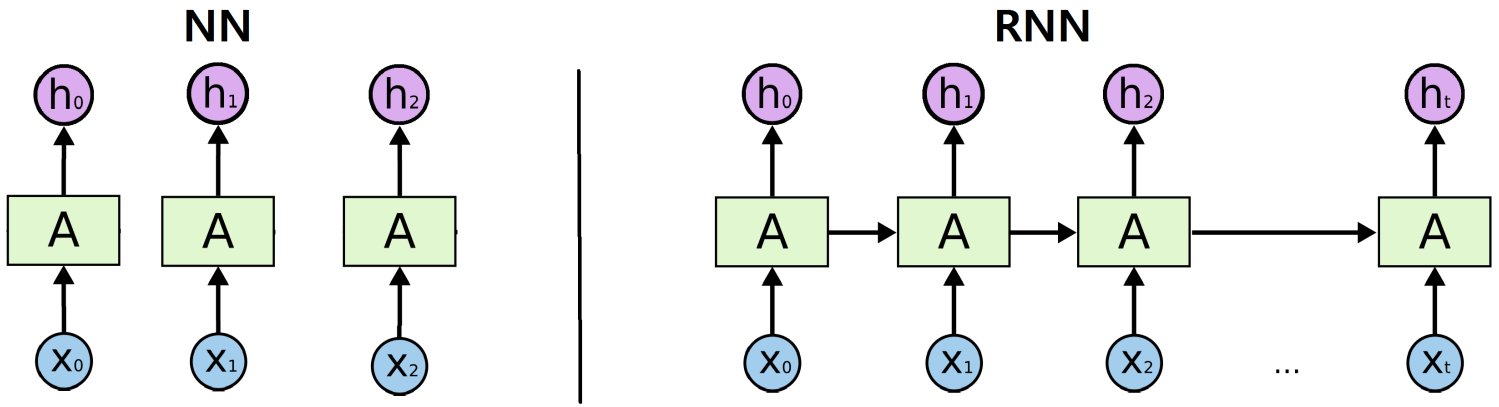
- CNN features learn hierarchical representations of images

  [Lee, 2011; Zeiler & Fergus, 2013; Yosinsky et al., 2015]

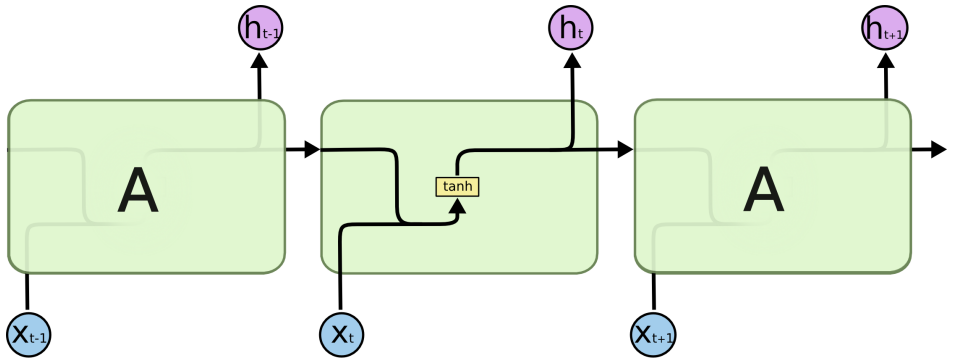


Two Building Blocks of DHNN - RNN

- RNN captures sequential information



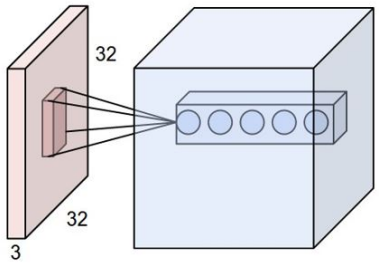
- RNN chain

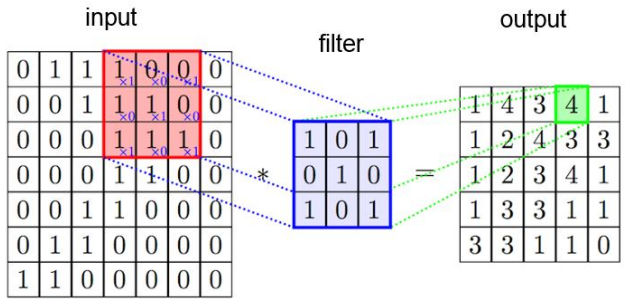


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Intro-CNNRNN

Convolutional layer

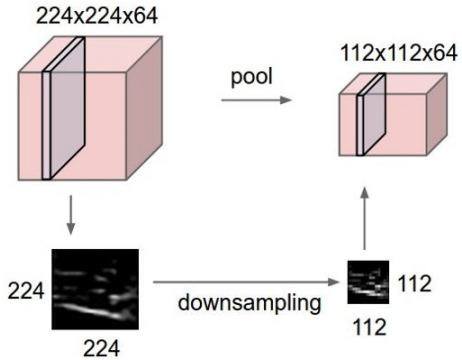


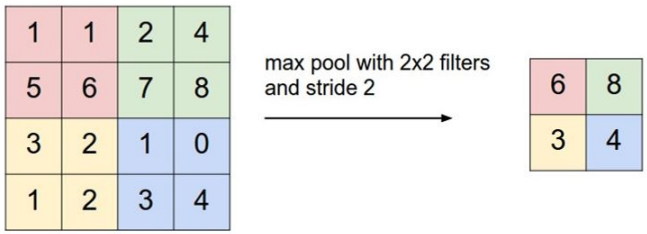


http://cs231n.github.io/convolutional-networks/

https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

Pooling layer





http://cs231n.github.io/convolutional-networks/

Hc-cnn (Ref-link)

DHNN - Conversion Operations

Linear conversion

- Convert CNN features through the n-mode product of a tensor
- Simple to use for modifying dimension of the CNN features
- Need additional trainable weights

$$g(a_s) = \text{vec}(a_s \times_1 U_s^1 \times_2 U_s^2 \times_3 U_s^3)$$

- $p = m \cdot n \cdot v$
- $U_s^1 \in \mathbb{R}^{m \times D_s}$, $U_s^2 \in \mathbb{R}^{n \times W_s}$, & $U_s^3 \in \mathbb{R}^{v \times H_s}$: trainable parameter matrices at CNN layer s
- $\times_k$: k-mode product of a tensor by a matrix
- vec: flattening operation

Conversion

DHNN - Conversion Operations

Convolutional conversion

- Keep spatial information of the CNN features
- Deal with dimensionality efficiently
- Need additional trainable weights

$$g(a_s) = \text{vec}(\text{Conv}(a_s; C_{conv_s}))$$

  - C_{conv_s}: trainable parameters

Pooling conversion

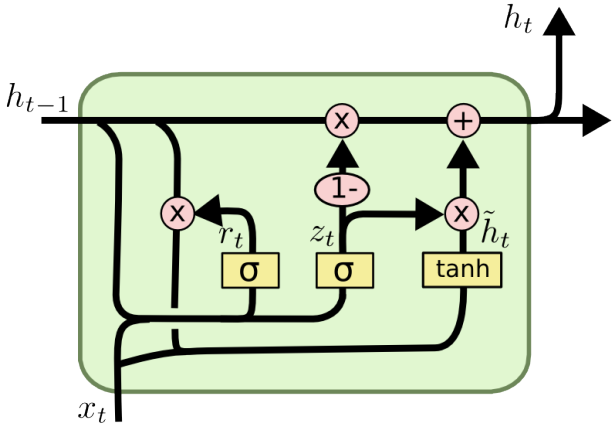
- Keep spatial information of the CNN features
- No additional trainable weights

$$g(a_s) = \text{vec}(\text{Pool}(a_s))$$

Conversion

RNN - LSTM

- LSTM cell



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

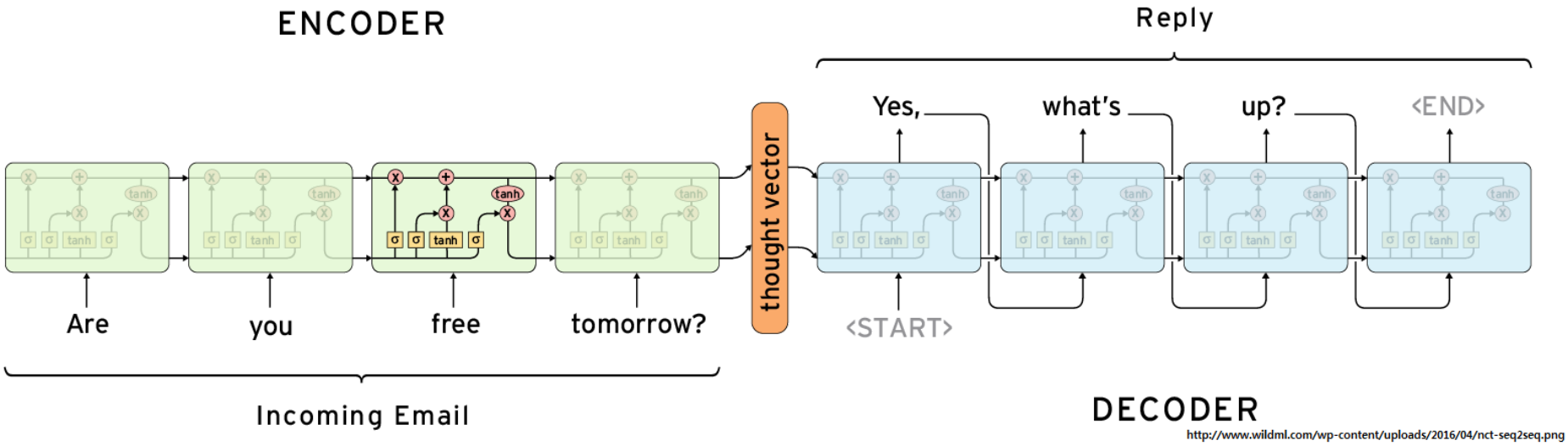
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Hc-rnn

Sequence to Sequence (S2S) [Cho et al., 2014]



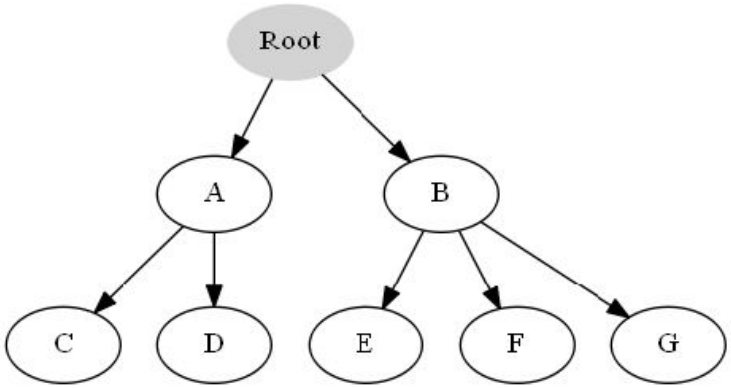
- Consist of two RNNs
  - Encoder processes the input sequence
  - Decoder generates the output sequence
- Learn sequences of variable lengths

CNN-S2S

Encoding Label Paths

C How to encode label paths

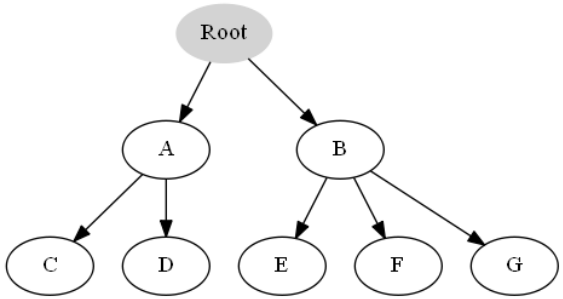
Assume that we have a tree: Root, A, and B; A has children C, and D; B has children E, F, and G.



Here, $L = (V \cup \{\text{Root}\}, E), V = \{\text{A,B,C,D,E,F,G}\}, |V| = 7,$ and $T = 2$. o_t and y_t represent model outputs and encoded true path of (A,B,C,D,E,F,G) at step t, and so they have dimension 7. Say we a sample labeled as F, so the label path is $P = \{\text{Root, B, F}\}$. Then the path is encoded: $y_1 = (0, 1, 0, 0, 0, 0, 0)$; $y_2 = (0, 0, 0, 0, 0, 1, 0)$.

Train-and-inference

Beam Search



Algorithm 1 Beam search inference for a sample

procedure
We have a tree, $L = (V \cup \{\text{Root}\}, E)$, and outputs from a trained model for a sample, o_{tj} for $t = 1, \ldots, T$, & $j \in V_t$ where V_t represents nodes at t. Beam search parameter, k, and a set, Best-path$_t$ keeping the best k paths at t are given.
    Update: Best-path$_1$ caculating argmax-k$_{j \in V_1}$ (o_{1j})
    for $t = 2, \ldots, T-1,$ **do**
        Calculate: argmax-k$_{j \in V_t}$ $(o_{tj}|\text{Best-path}_{t-1})$
        Update: Best-path$_t$
    end for
    Final label = argmax$_{j \in V_T}$ $(o_{Tj}|\text{Best-path}_{T-1})$
end procedure

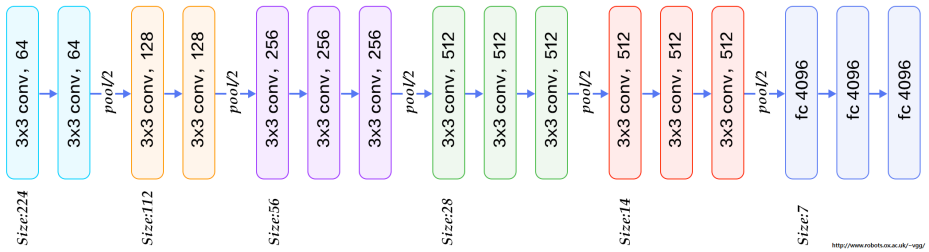
Algorithm 1 describe beam search algorithm. From the tree in Appendix **??**, assume we have: o_1 = (A,B,C,D,E,F,G) = $(0.3, 0.4, 0.1, 0.1, 0.1, 0, 0)$, o_2 = (A,B,C,D,E,F,G) = $(0, 0.1, 0, 0.1, 0.1, 0.2, 0.5)$, and $k = 1$. Final label is {Root,B,G}.

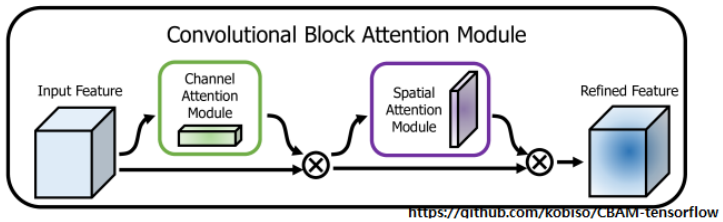
Train-and-inference

CNN Architectures

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
112 × 112	conv, 7 × 7, 64, stride 2		
56 × 56	max pool, 3 × 3, stride 2		
	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \quad C=32 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28 × 28	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \quad C=32 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14 × 14	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \quad C=32 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 1024 \\ \text{conv}, 3\times3, 1024 \quad C=32 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1 × 1	global average pool, 1000-d fc, softmax		

https://arxiv.org/pdf/1709.01507.pdf





https://github.com/kobiso/CBAM-tensorflow

Exp-OI

CNN Architectures

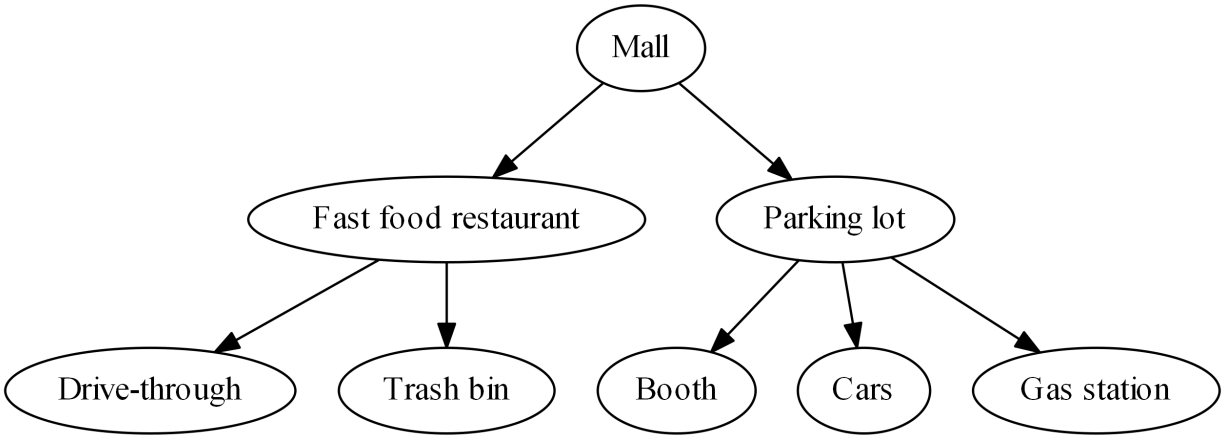
Model	MACC	COMP	ADD	DIV	Activations	Params	SIZE(MB)
SimpleNet	1.9G	1.82M	1.5M	1.5M	6.38M	6.4M	24.4
SqueezeNet	861.34M	9.67M	226K	1.51M	12.58M	1.25M	4.7
Inception v4*	12.27G	21.87M	53.42M	15.09M	72.56M	42.71M	163
Inception v3*	5.72G	16.53M	25.94M	8.97M	41.33M	23.83M	91
Incep-Resv2*	13.18G	31.57M	38.81M	25.06M	117.8M	55.97M	214
ResNet-152	11.3G	22.33M	35.27M	22.03M	100.11M	60.19M	230
ResNet-50	3.87G	10.89M	16.21M	10.59M	46.72M	25.56M	97.70
AlexNet	7.27G	17.69M	4.78M	9.55M	20.81M	60.97M	217.00
GoogleNet	16.04G	161.07M	8.83M	16.64M	102.19M	7M	40
NIN	11.06G	28.93M	380K	20K	38.79M	7.6M	29
VGG16	154.7G	196.85M	10K	10K	288.03M	138.36M	512.2

*Inception v3, v4 did not have any Caffe model, so we reported their size related information from MXNet and Tensorflow respectively. Inception-ResNet-V2 would take 60 days of training with 2 Titan X to achieve the reported accuracy. Statistics are obtained using `http://dgschwend.github.io/netscope`

A.2 Generalization Samples

Computational Study

- Fixed path length tree model (REAL): CNN-RNN



Method	Accuracy (%)	
	Path	Node
VGG-16	64.0	71.3
VGG-RNN	59.9	77.3
VGG-RNN-Alt	61.7	78.1
VGG-RNN-Alt-Resi	**65.3**	**80.5**
Res-50	62.1	68.8
Res-RNN	63.4	78.9
Res-RNN-Alt	62.4	77.0
Res-RNN-Alt-Resi	**64.0**	**78.4**

Exp-OI

Open Image Label Tree

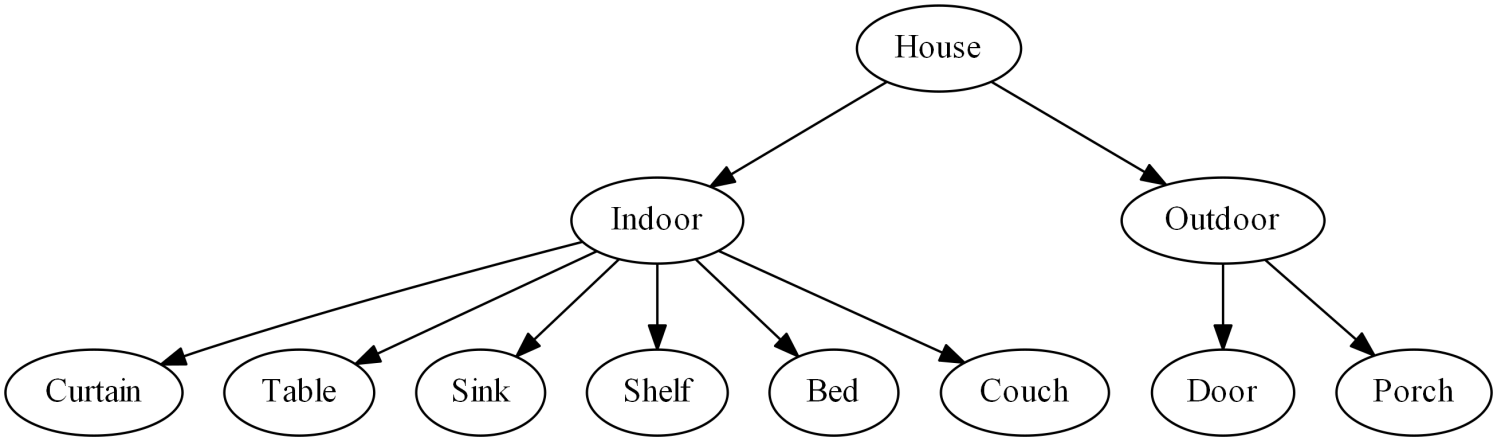


Figure: A part of the tree of Open Images

Exp-OI