# Intro to Recurrent Neural Networks

COS 598B
William Hinthorn

# This Lecture:

1. **Segmentation Review**
2. Intro to Polygon-RNN
3. Modeling Sequences
    1. Vanilla RNN
    2. Training and Problems
    3. LSTM
    4. Convolutional LSTM
4. Vertex Prediction using Conv-LSTM

# Previously:

- Instance-segmentation
- Evaluation Metrics: mAP; mIoU

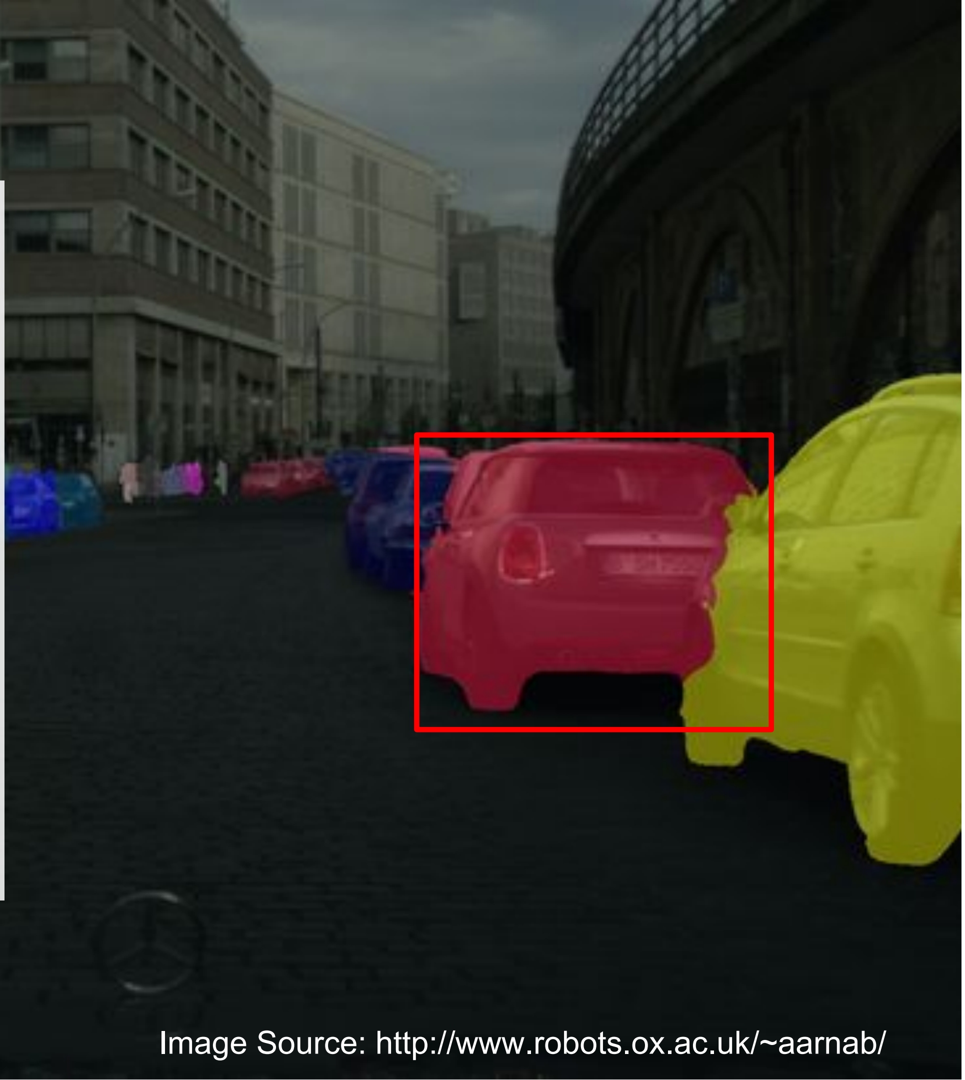
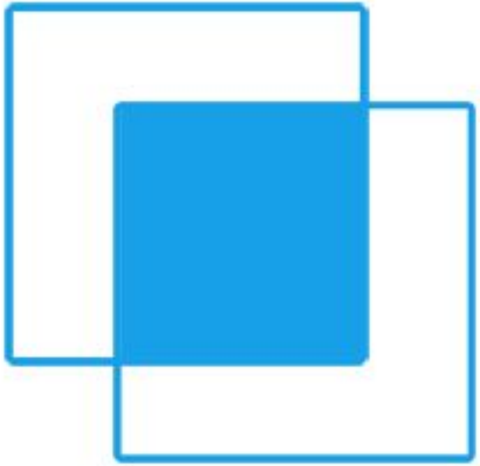


Image Source: http://www.robots.ox.ac.uk/~aarnab/

# Intersection over Union Review



$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Image Source: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

# Problem: Standard Annotation Methods

- Tedious polygon drawing


Can we automate?

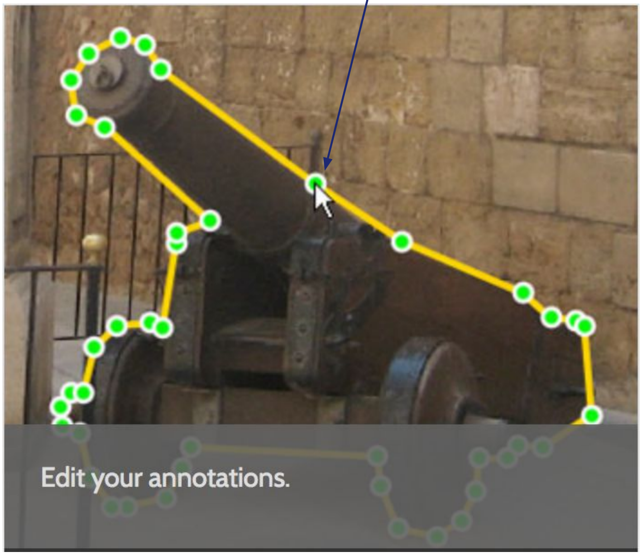
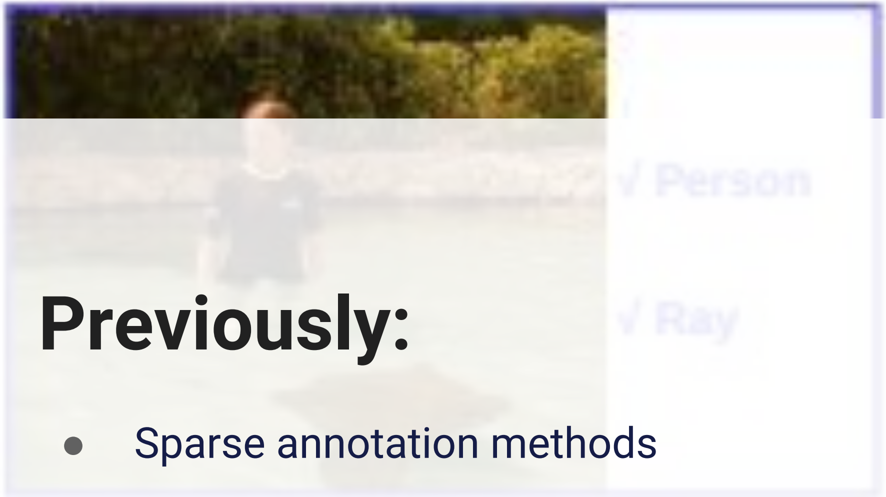
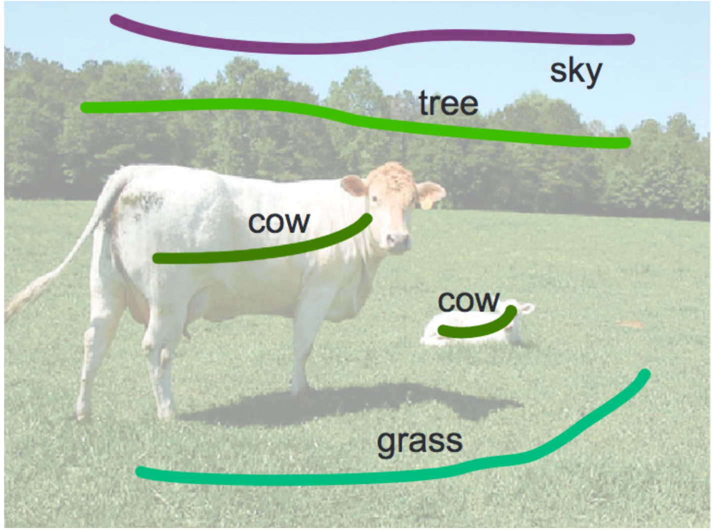



Image Source: Labelme2.csail.mit.edu

# Previously:

- Sparse annotation methods





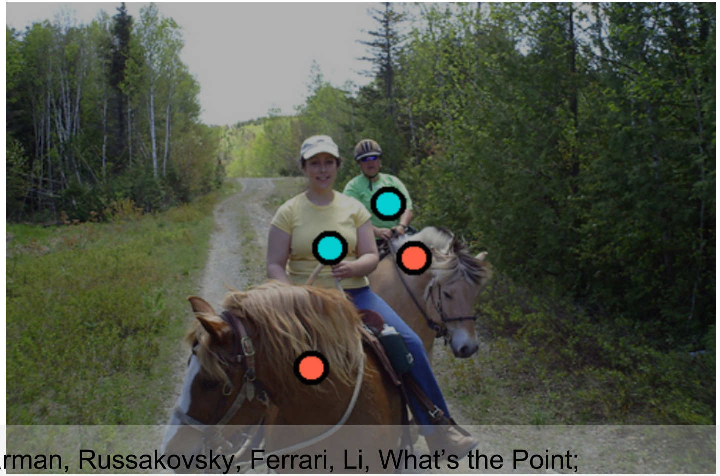


Image Sources: Bearman, Russakovsky, Ferrari, Li, What's the Point; Dai, He, Sun, BoxSup; Ft. Yannis Karakozis

# Previously:

- Semi-automatic segmentation
  - Graphical models with smoothness term
  - **No shape prior**
- GrabCut
  - Too **error-prone** for benchmark creation
- Mistakes are tedious to correct by hand

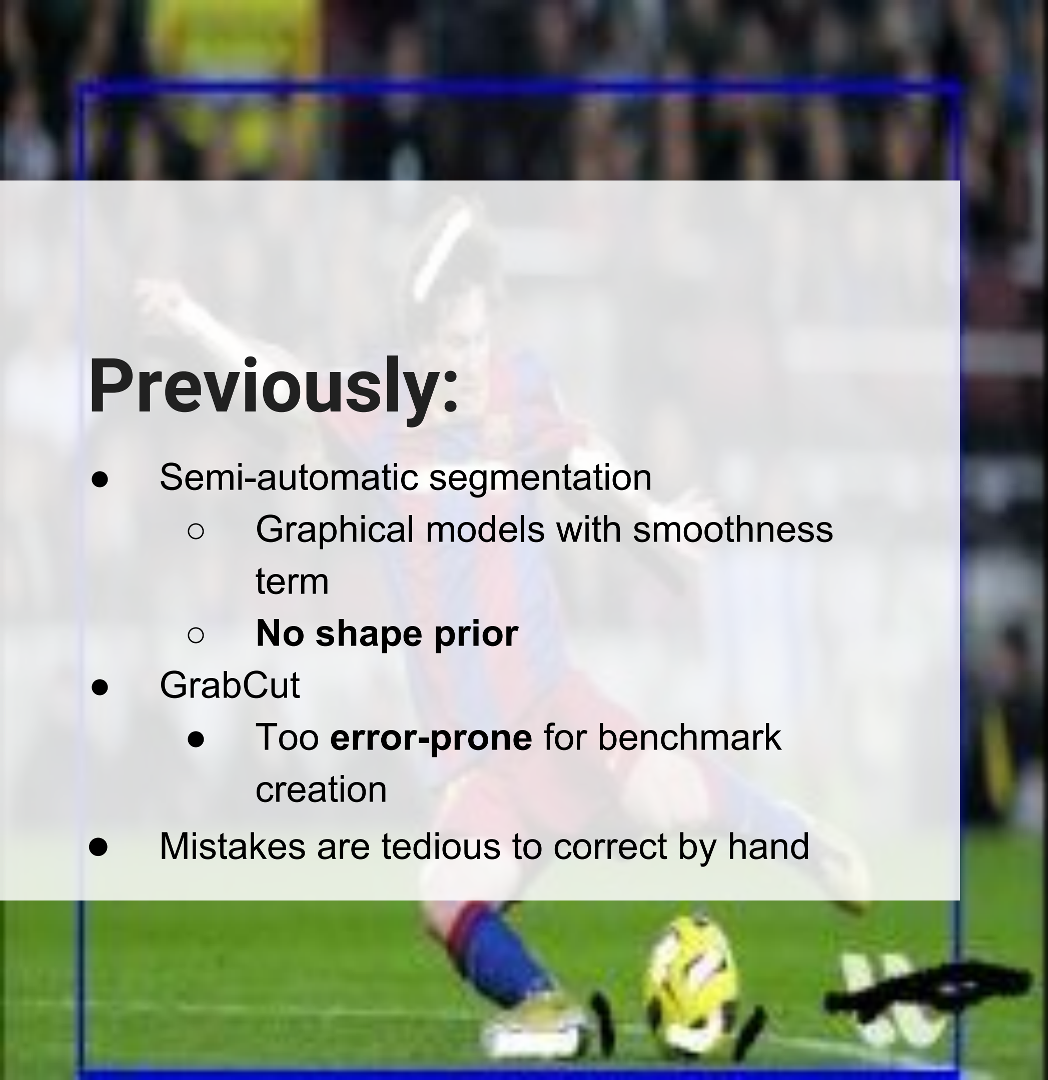


Image Source: Grabcut tutorial from www.opencv.org

# Outline

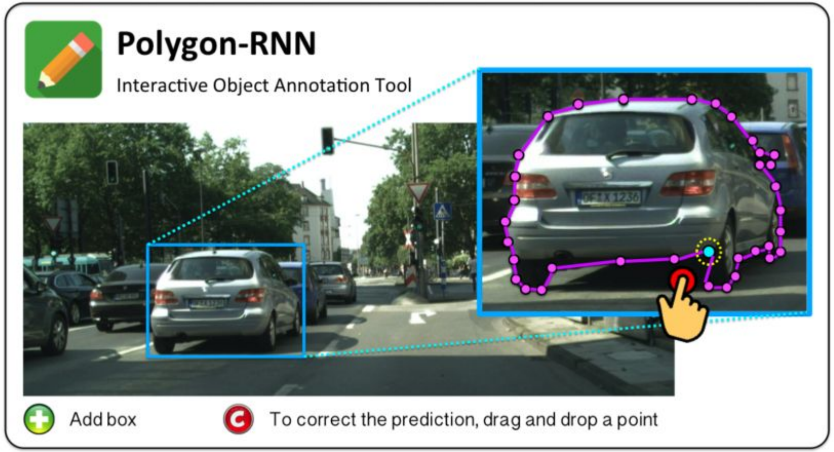
1. Review
2. **Intro to Polygon-RNN**
3. Modeling Sequences
    1. Vanilla RNN
    2. Training and Problems
    3. LSTM
    4. Convolutional LSTM
4. Vertex Prediction using Conv-LSTM

# Polygon-RNN

Castrejón, Kundu, Urtasun, and Fidler
Honorable Mention for Best Paper Award CVPR ‘17

# Polygon-RNN Goals

- Maintain high annotation **accuracy**
    - **Benchmark-grade** masks
    - Evaluation Metric: agreement (in IoU) with ground-truth
- Reduce annotation **cost**
    - Measured using average instance annotation time
    - **Number of clicks** (i.e. corrections) per image



# Model Overview

1. **Adapt VGG-16[1] for feature extraction**
2. Two-layer Convolutional LSTM for polygon vertex inference

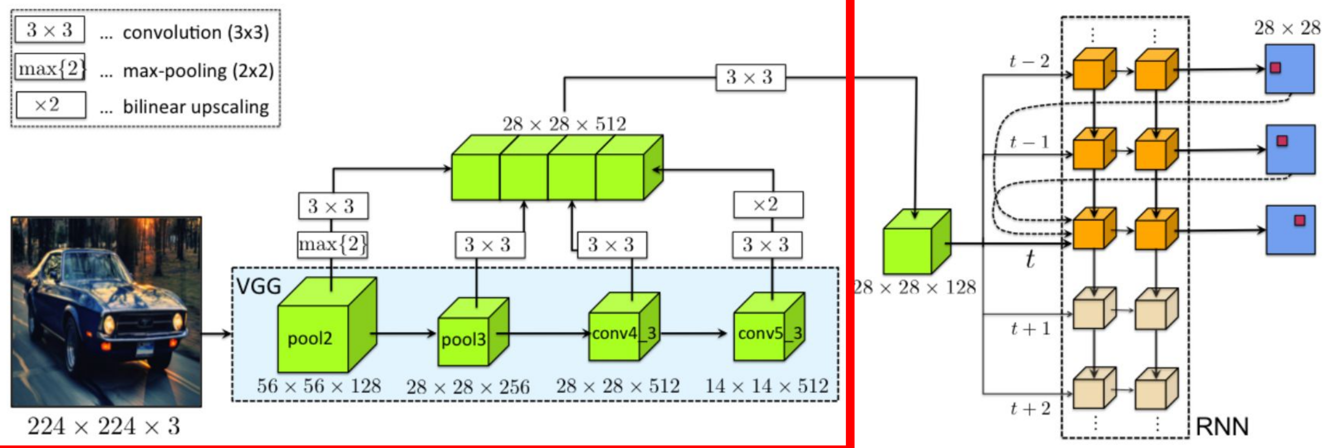



Figure 2. **Our Polygon-RNN model**. At each time step of the RNN-decoder (right), we feed in an image representation using a modified VGG architecture. Our RNN is a two-layer convolutional LSTM with skip-connection from one and two time steps ago. At the output at each time step, we predict the spatial location of the new vertex of the polygon.

[1]K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CVPR '14

# Step 1: Feature Extraction

# Review: VGG-16

Simonyan and Zisserman; ICLR ‘14

- Simple, modular structure (feedforward)
- Single Stride
  - Reduced loss of information
- Stacked with small kernels
  - Large receptive field
  - Fewer learnable parameters
- Rectified Linear Units (ReLU) (like AlexNet)

| |
|---|
| Input Image: (224x224x3) |
| Conv3-64 x 2 |
| Maxpool |
| Conv3-128 x 2 |
| Maxpool |
| Conv3-256 x 3 |
| Maxpool |
| Conv3-512 x 3 |
| Maxpool |
| Conv3-512 x 3 |
| Maxpool |
| FC 4096 x 2 |
| FC 1000 |
| Softmax |

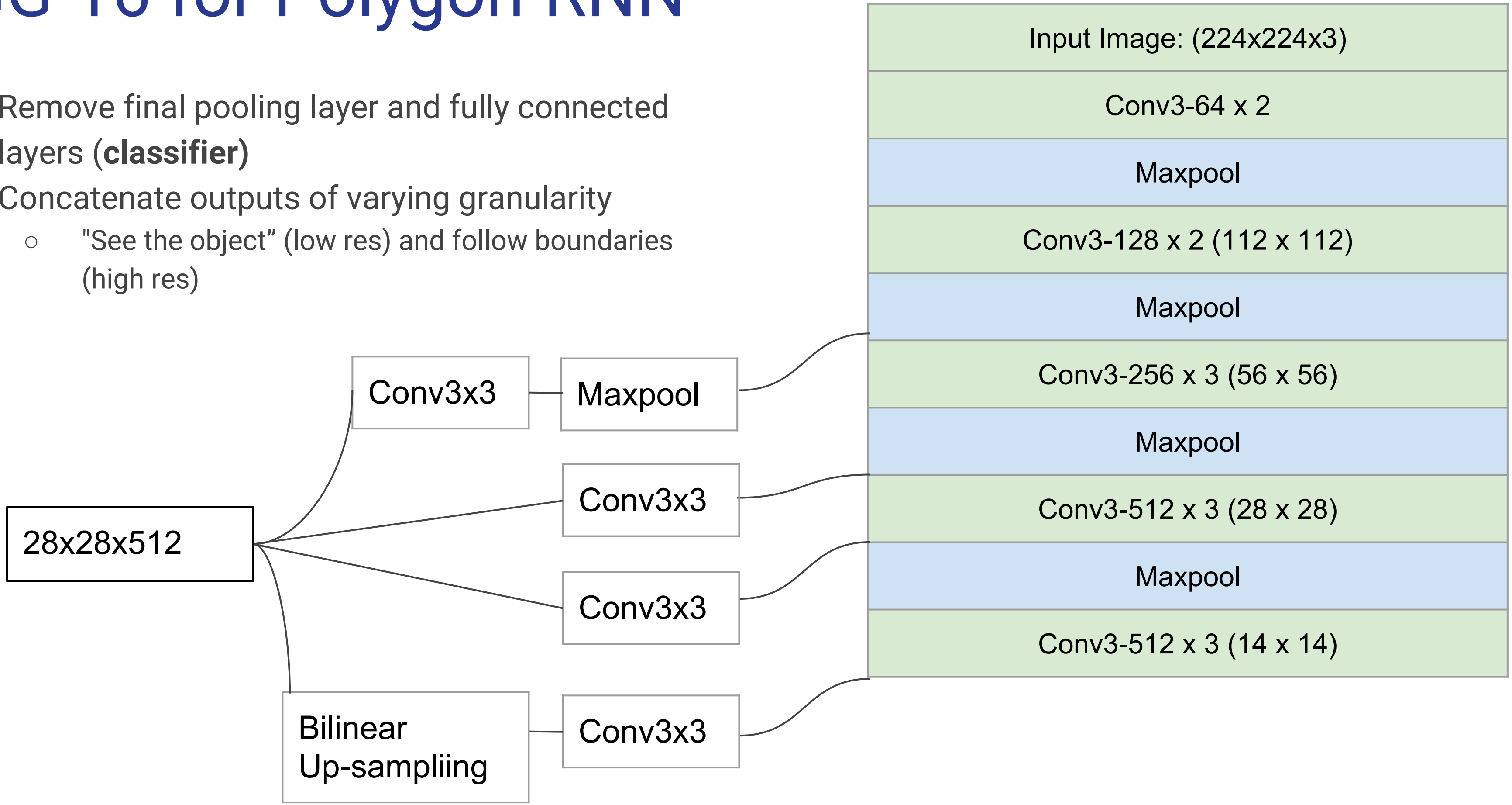
# VGG-16 for Polygon-RNN

- Remove final pooling layer and fully connected layers (**classifier**)

| |
|---|
| Input Image: (224x224x3) |
| Conv3-64 x 2 |
| Maxpool |
| Conv3-128 x 2 |
| Maxpool |
| Conv3-256 x 3 |
| Maxpool |
| Conv3-512 x 3 |
| Maxpool |
| Conv3-512 x 3 |
| ~~Maxpool~~ |
| ~~FC-4096 x 2~~ |
| ~~FC-1000~~ |
| ~~Softmax~~ |

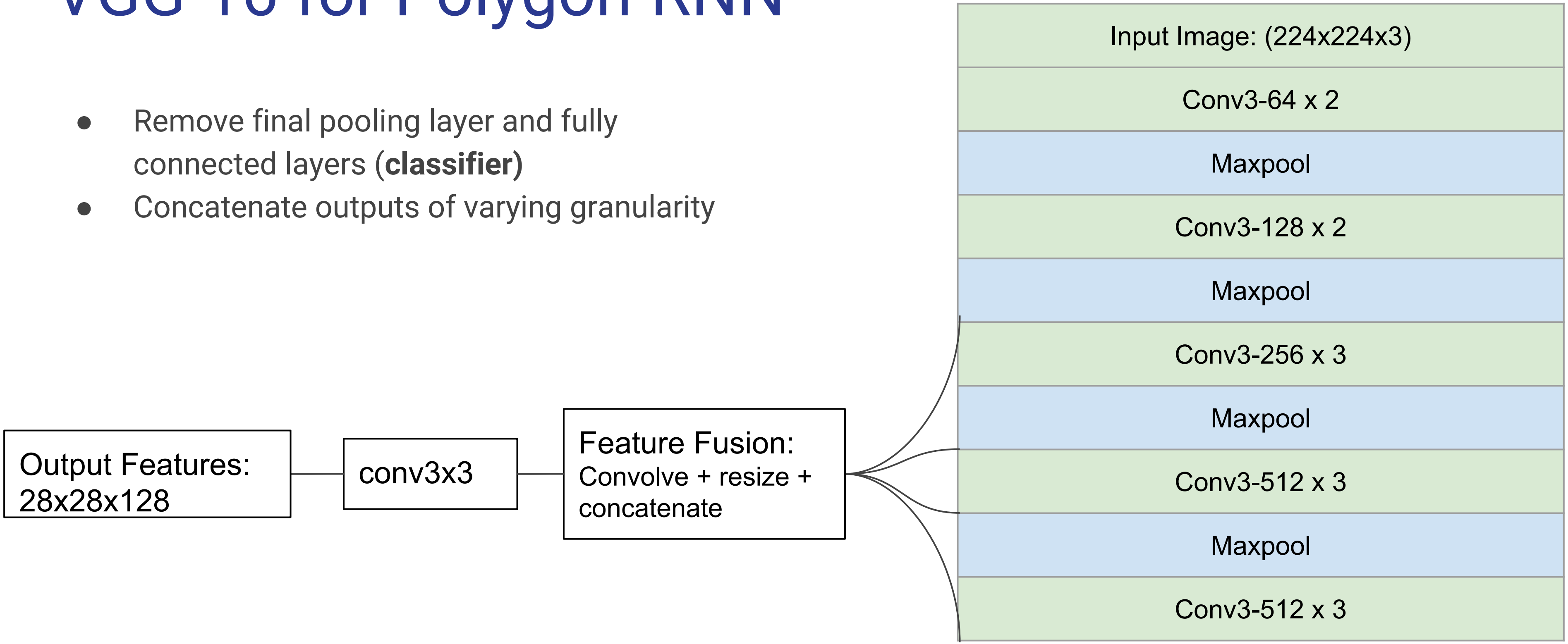
# VGG-16 for Polygon-RNN

- Remove final pooling layer and fully connected layers (**classifier)**
- Concatenate outputs of varying granularity
    - "See the object” (low res) and follow boundaries (high res)

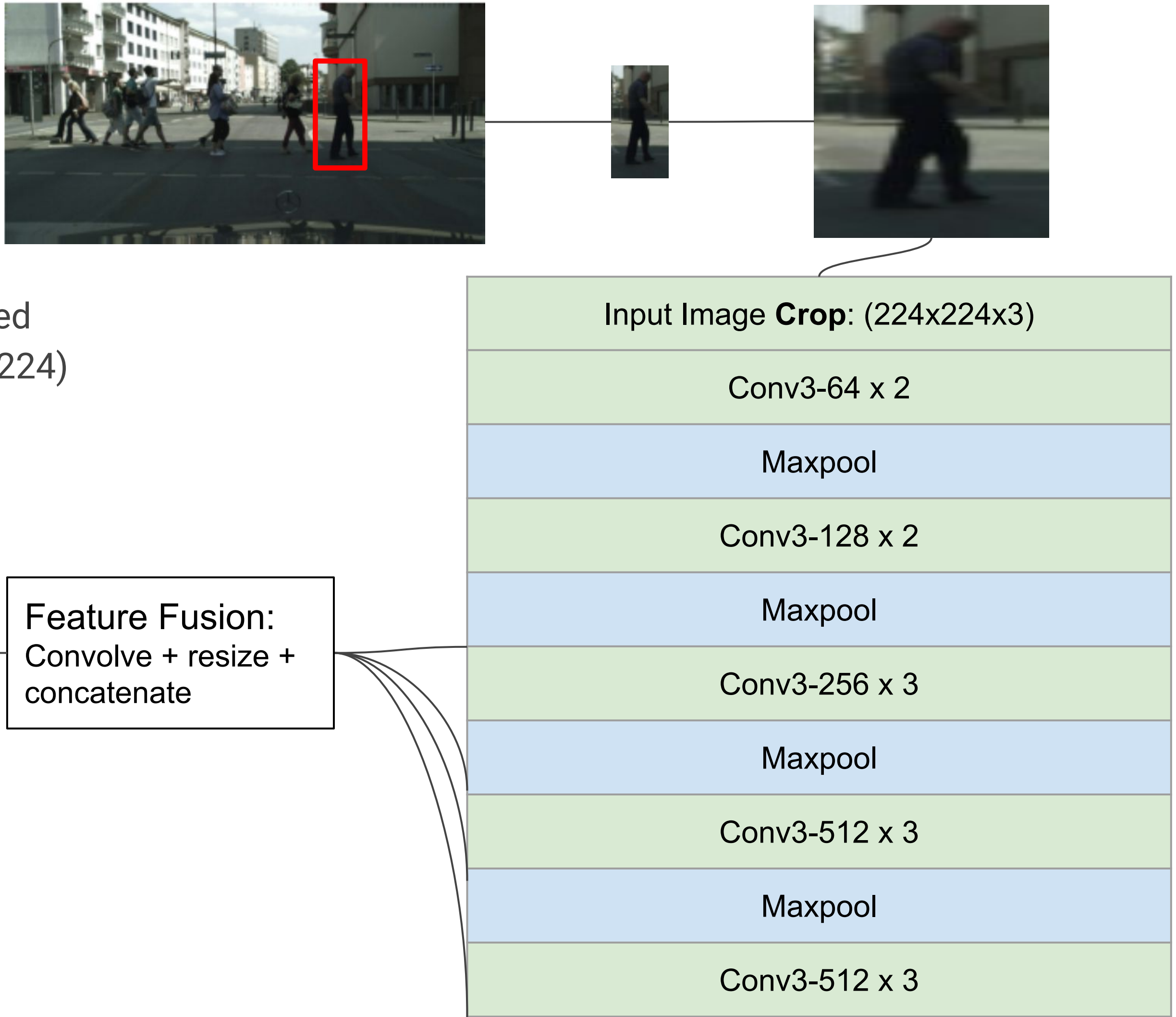


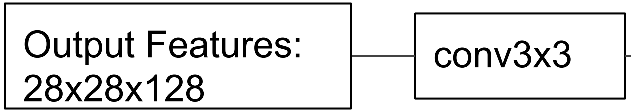
# VGG-16 for Polygon-RNN

- Remove final pooling layer and fully connected layers (**classifier**)
- Concatenate outputs of varying granularity



- **Input**: image crops from annotated bounding boxes (resized to 224x224)



# Step 2: Vertex Prediction

# Choices

- Single Shot
  - Simple and fast
  - **More clicking (For correction)**

- Predict vertices sequentially
  - Requires more design decisions and separate inferences (~250 ms/inf) for each vertex
  - **Allows for human-in-the-loop annotation to increase accuracy**

# Polygon-RNN Steps

1. Extract features using modified VGG-16
2. **Predict polygon vertices using a 2-layer convolutional LSTM (16 channel hidden layer)**

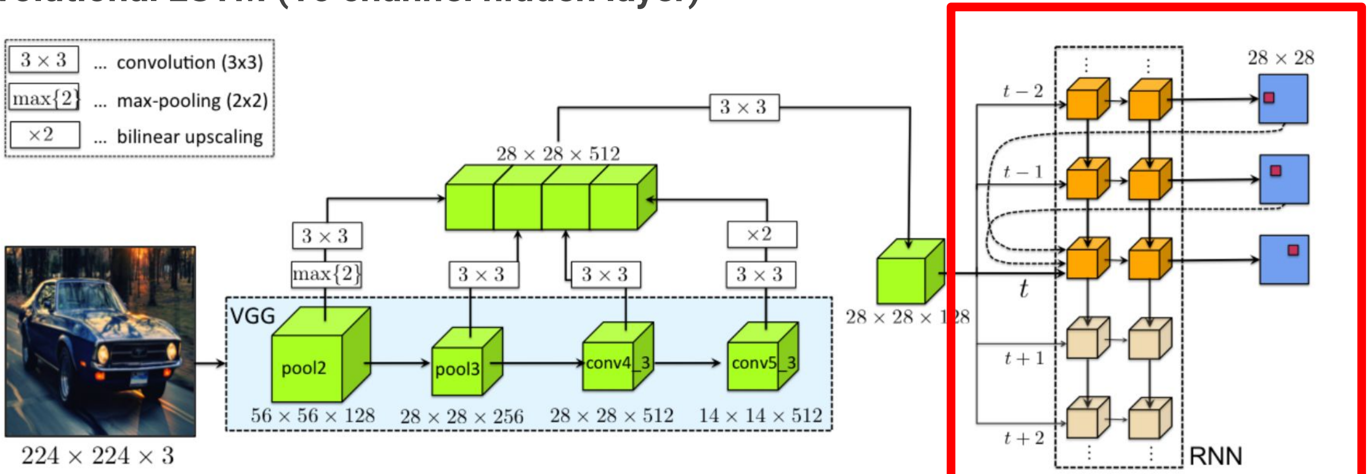



Figure 2. **Our Polygon-RNN model**. At each time step of the RNN-decoder (right), we feed in an image representation using a modified VGG architecture. Our RNN is a two-layer convolutional LSTM with skip-connection from one and two time steps ago. At the output at each time step, we predict the spatial location of the new vertex of the polygon.

# Outline

1. Review
2. Intro to Polygon-RNN
3. **Modeling Sequences**
    1. Vanilla RNN
    2. Training and Problems
    3. LSTM
    4. Convolutional LSTM
4. Vertex Prediction using Conv-LSTM

# What is an RNN?

# RNN Overview

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

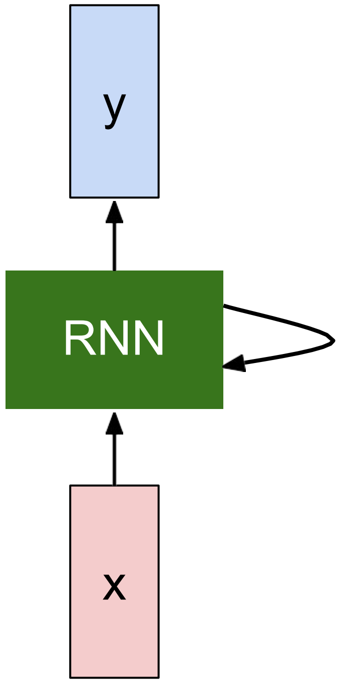
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state old state

input vector at some time step

some function

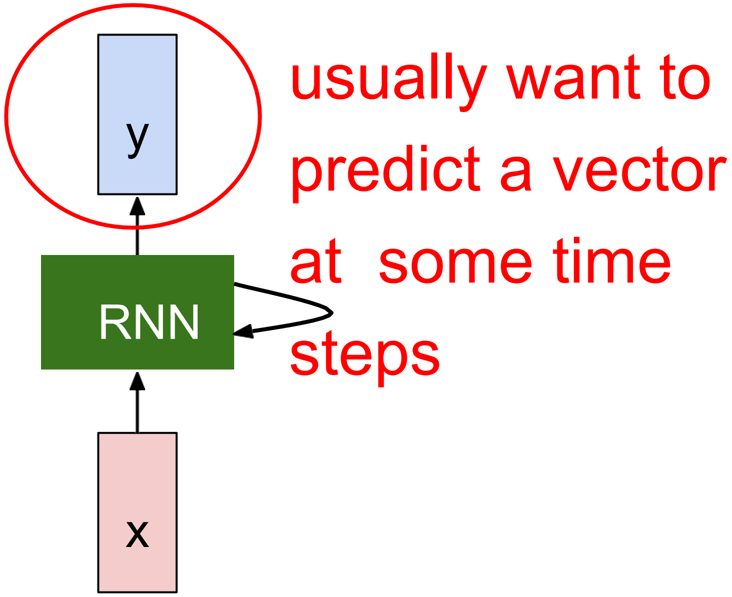
with parameters W




Contents stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

# Why RNN?

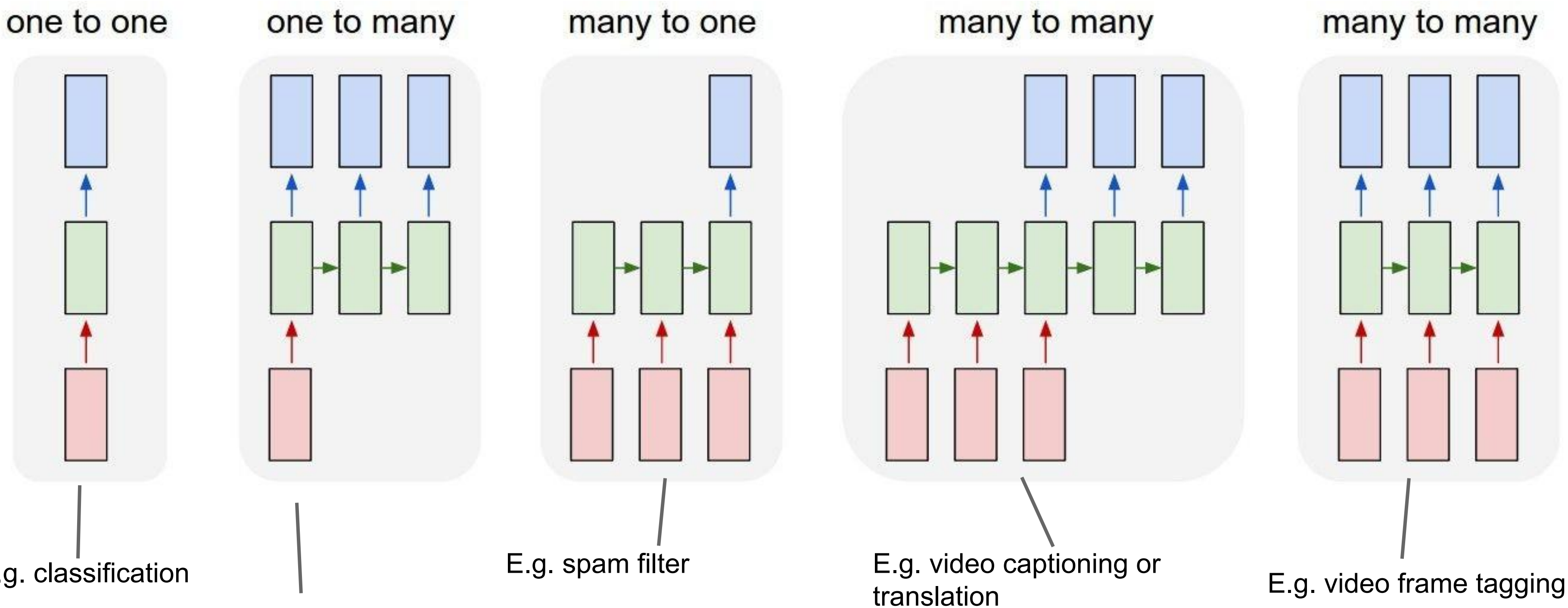
1. Linear Dynamical Systems (LDS)
    1. **Limited to linear state updates (linear dynamics…)**
2. Hidden Markov Models (HMM)
    1. Discrete hidden states - **can only remember log(N) bits** about prior data
3. RNNs address these limitations
    1. **Efficient information storage** for “long-range” dependencies
    2. **Nonlinear** state updates
    3. Turing Complete




Reference: https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

# RNN Flexibility




E.g. classification

E.g. spam filter

E.g. video captioning or translation

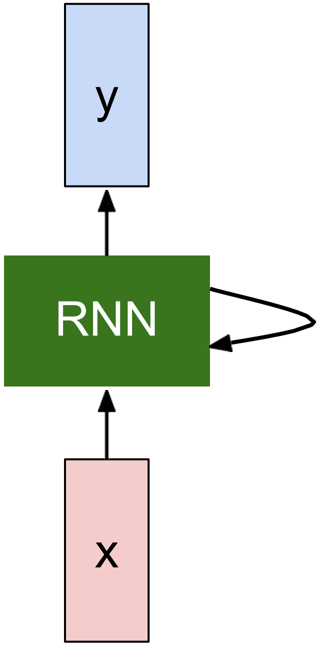
E.g. video frame tagging

**Vertex Prediction!**
**(without annotator)**

Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

# Elman Networks[1] (Vanilla RNN)

[Elman 1990]




$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

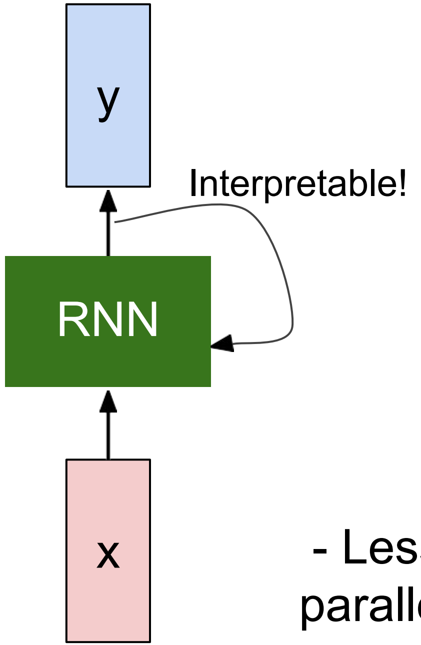
The state consists of a single *"hidden"* vector **h**.

Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

[1] *Elman, Jeffrey L. (1990). "Finding Structure in Time". Cognitive Science.* ***14*** *(2): 179–211.*

# Jordan Networks[1]

[Jordan 1997]




$$h_t = f_w(y_{t-1}, x_t)$$

$$\downarrow$$

$$h_t = tanh(W_{yh}y_{t-1} + W_{xh}x_t)$$

$$y_t = \sigma(W_y h_t)$$

- Less powerful than Elman Networks BUT easier to train (via parallelization)[2]
- Allows for **external** intervention

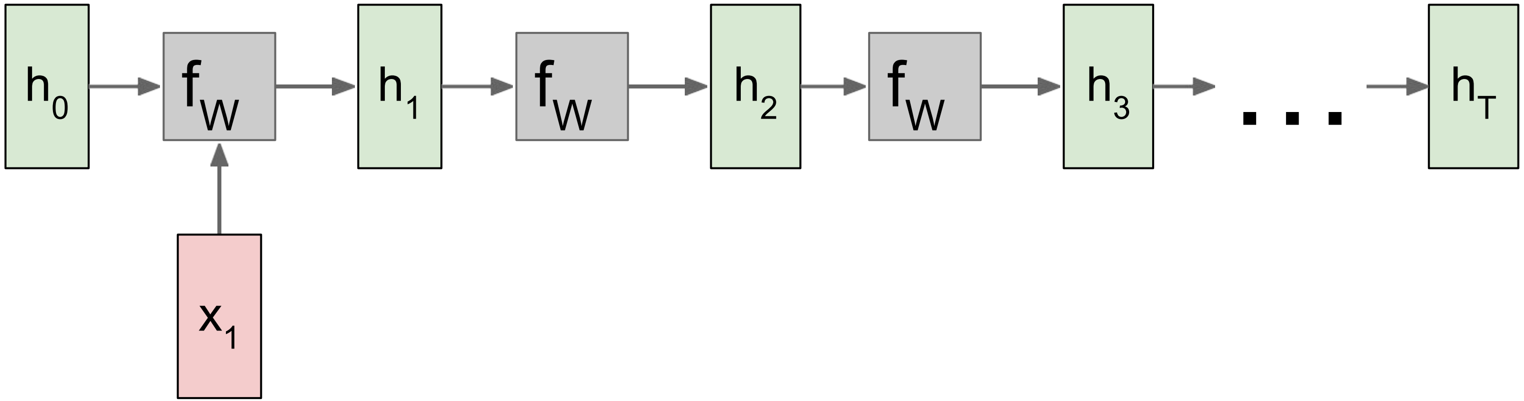
Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

[1] *Jordan, Michael I. (1997-01-01). "Serial Order: A Parallel Distributed Processing Approach".*

[2] see e.g. http://www.deeplearningbook.org/contents/rnn.html

# Computational Graph

- Note: by removing h_0, the graph becomes equivalent to that of a normal MLP



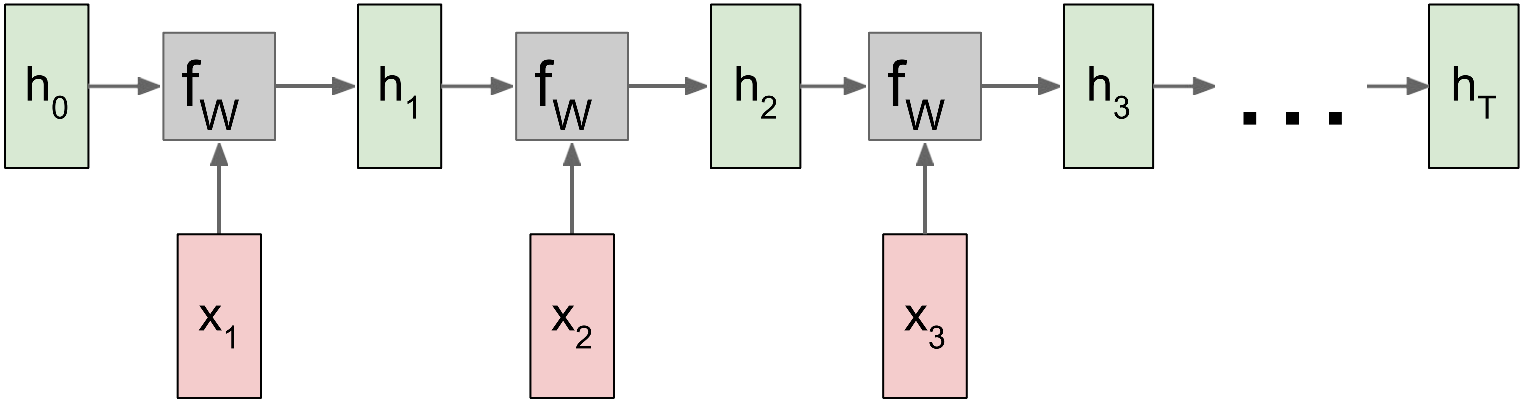

Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

# What about $h_0$?

- Could arbitrarily choose a starting value (e.g. 0.5)
- Could start with a computed average
- Could treat as a parameter to learn via backprop[1]
- Polygon-RNN chooses to start with 0
    - Represents "total ignorance"
    - Midpoint for range of tanh

* Hinton recommends/ed this in CS2535

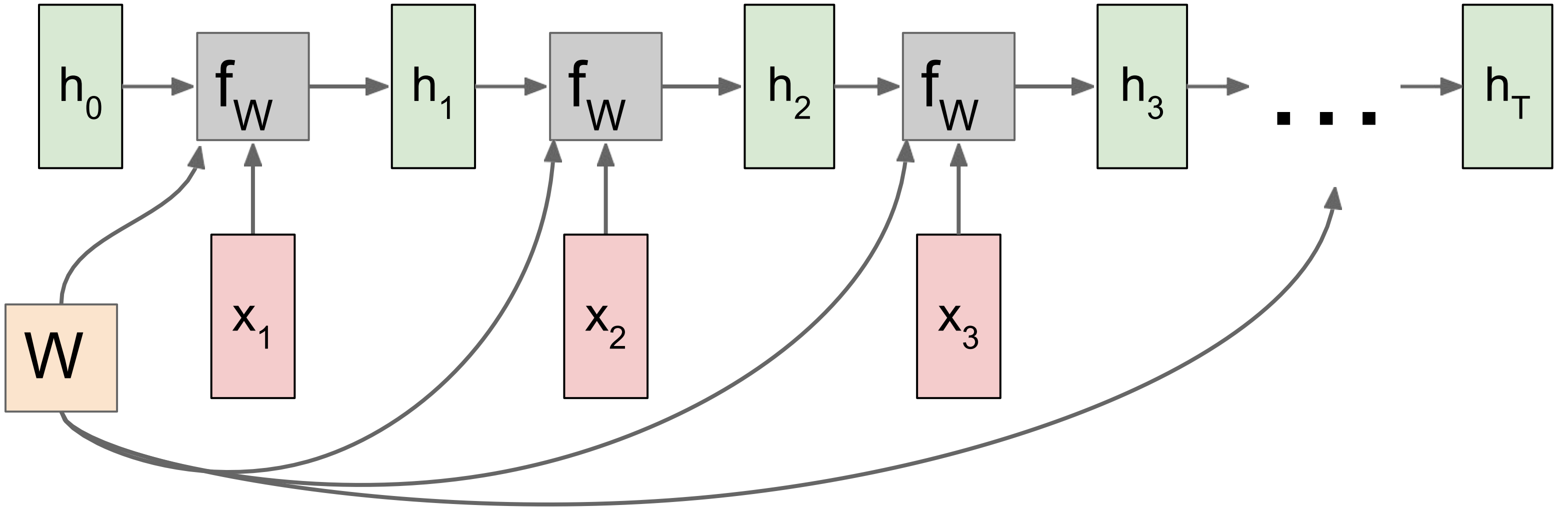
# Computational Graph




Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

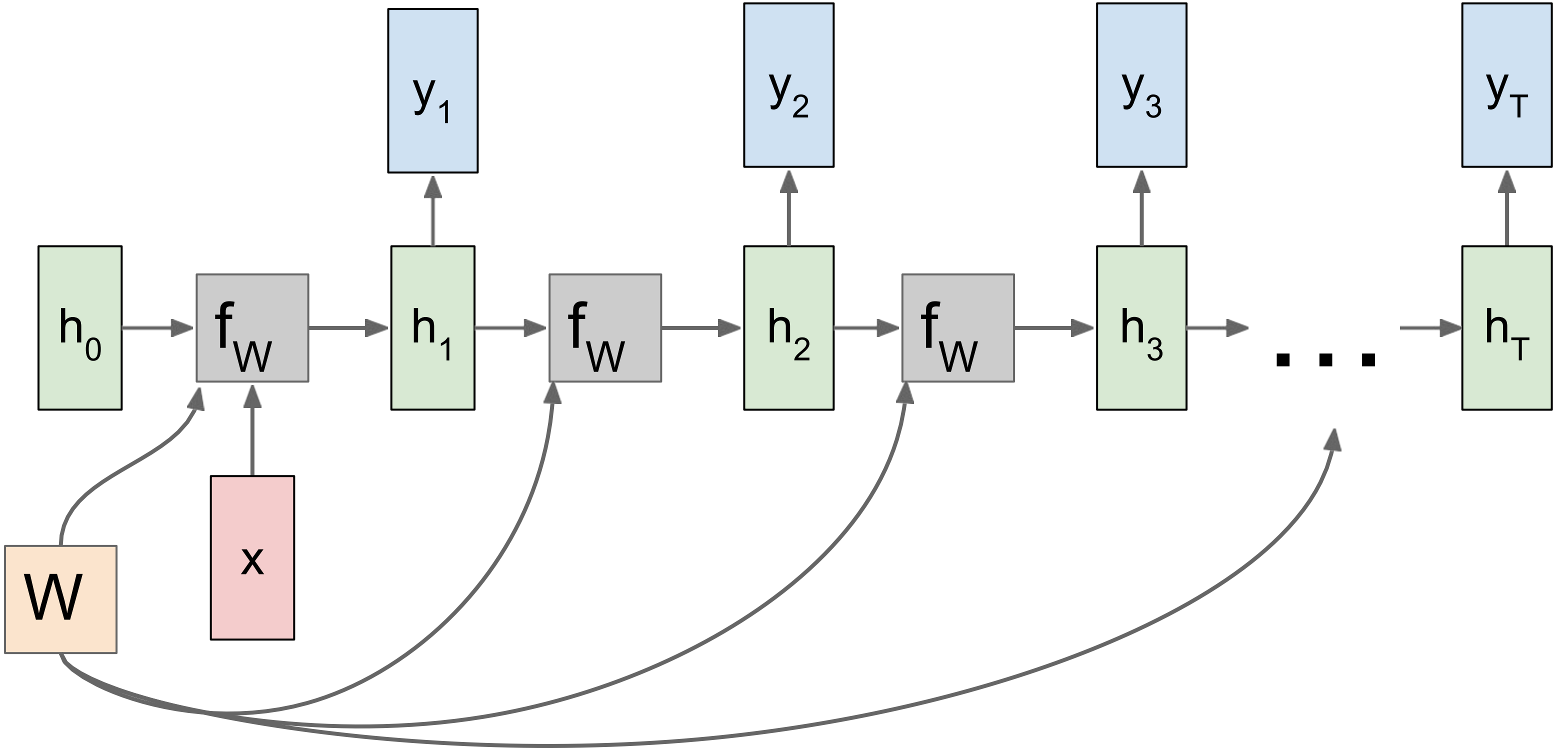
# Computational Graph

“Share the weights” - Re-use the same weight matrix at every time-step




Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

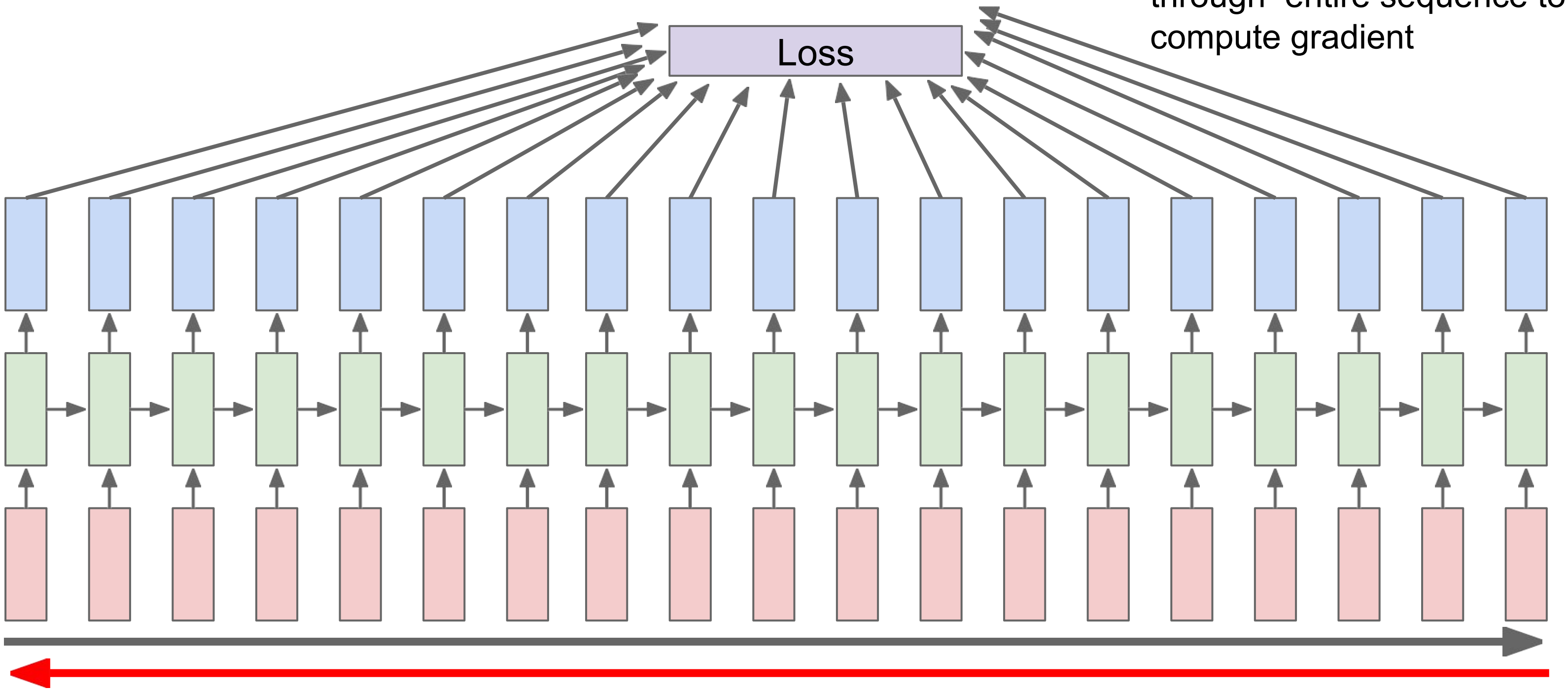
# Computational Graph For Vertex Prediction




Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

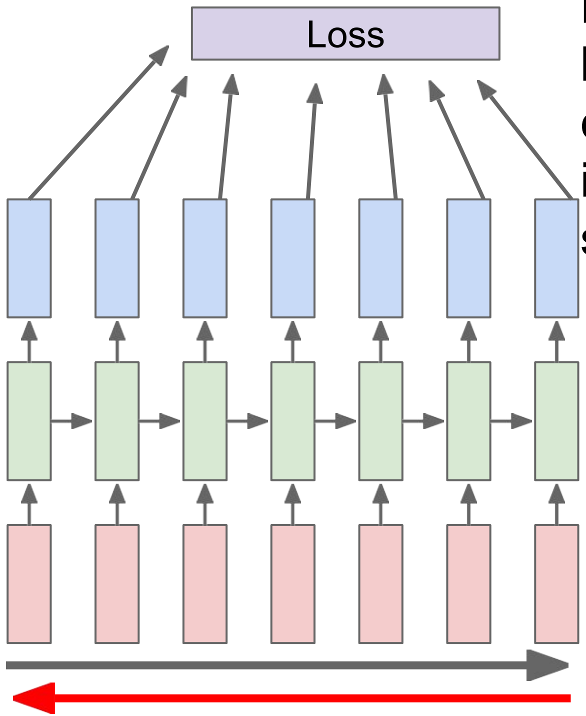
# Backpropagation through time (BPTT)

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient




Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

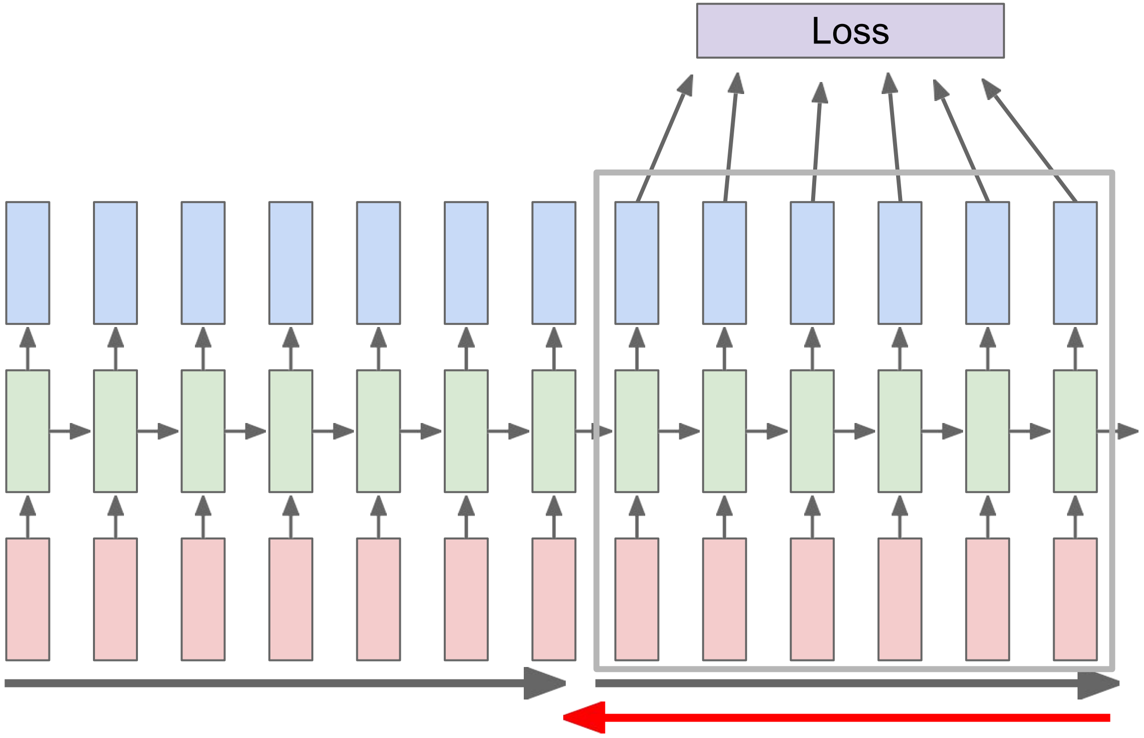
# Truncated BPTT




Run forward and backward through chunks of the sequence instead of whole sequence

Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

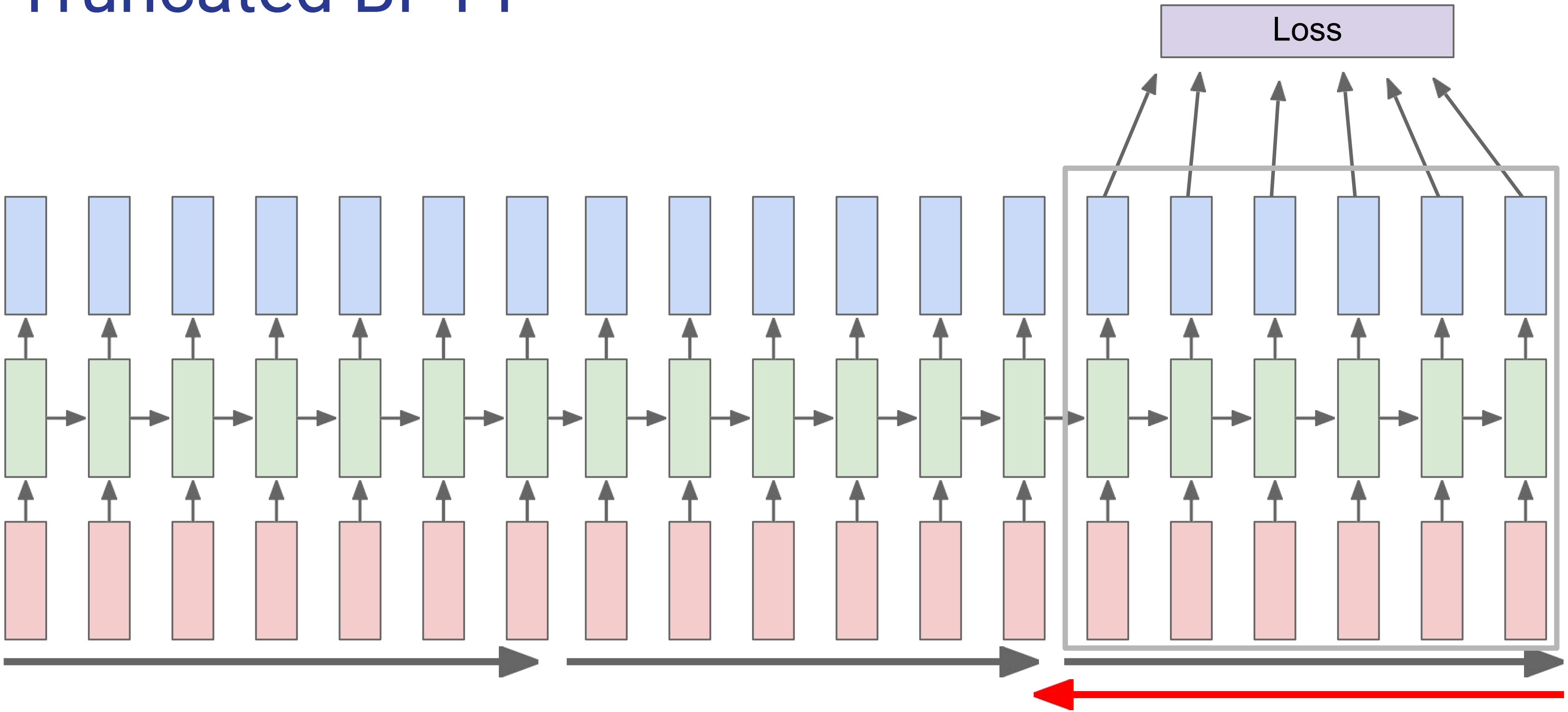
# Truncated BPTT




Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

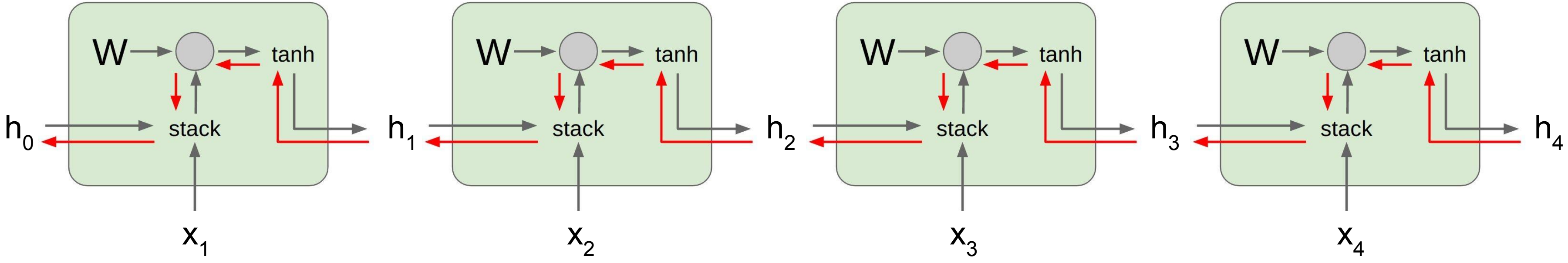
Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

# Truncated BPTT




Graphics stolen from CS231n. Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

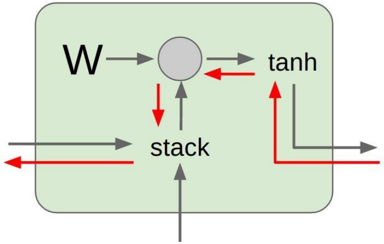
# RNN Gradient Flow




Computing gradient of $h_0$ involves many factors of W (and repeated tanh)

# BPTT Problems

- Gradient will contain terms that grow like $\mathbf{W}^k$
- What happens for singular values >> 1 (e.g. 4)?



# “For Whoever Shares, to Him More Gradient Will Be Given

- corruption of Mark 4:25”

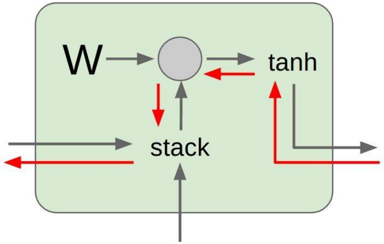
- Prof. Sebastian Seung

(But sometimes you can have too much of a good thing)

Quote taken completely out of context from COS485. Credit and apologies to Sebastian Seung

# BPTT Problems

- Gradient will contain terms that grow like $\mathbf{W}^k$
- What happens for singular values >> 1 (e.g. 4)?
  - **Exploding Gradients**
  - Gradient Clipping[1] (threshold the norm of the gradient)
- What happens for singular values << 1 (e.g. 0.2)?
  - **Vanishing Gradients**
  - LSTM (Next)

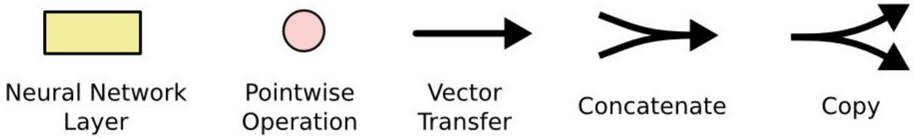


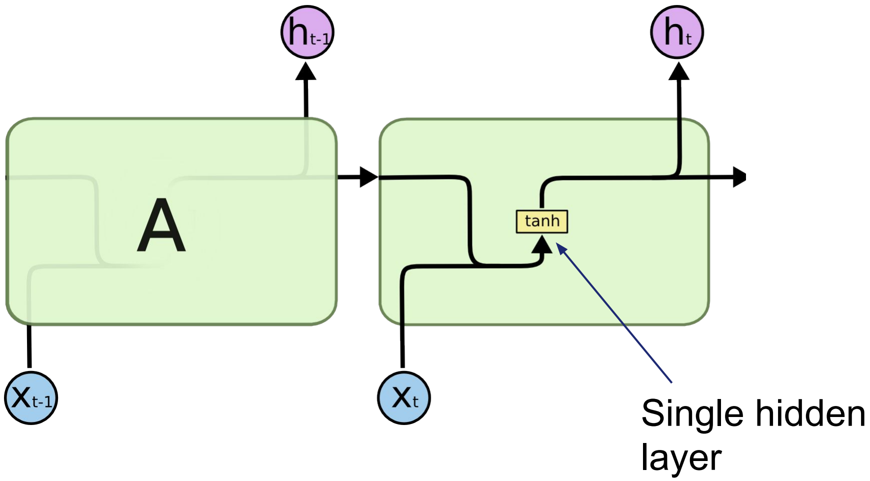

Slide stolen from CS519. Credit: Fuxin Li

[1]Pascaon, Mikolov, Bengio. “On the difficulty of training recurrent neural networks”

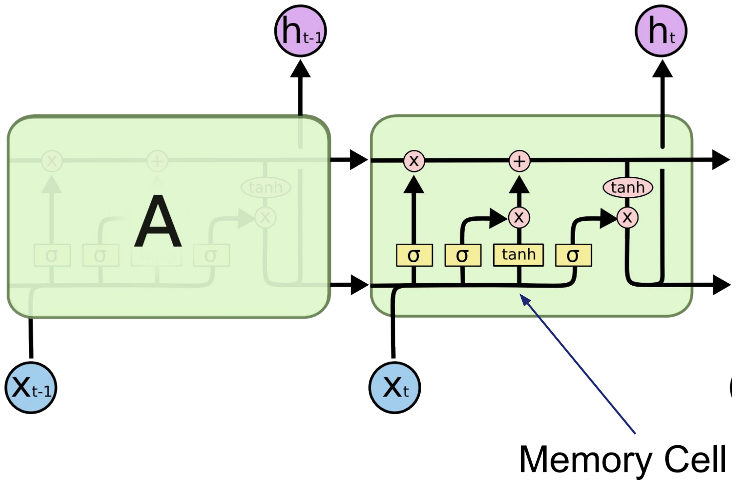
# Adding Memory

# Vanilla RNN to LSTM[1]




RNN

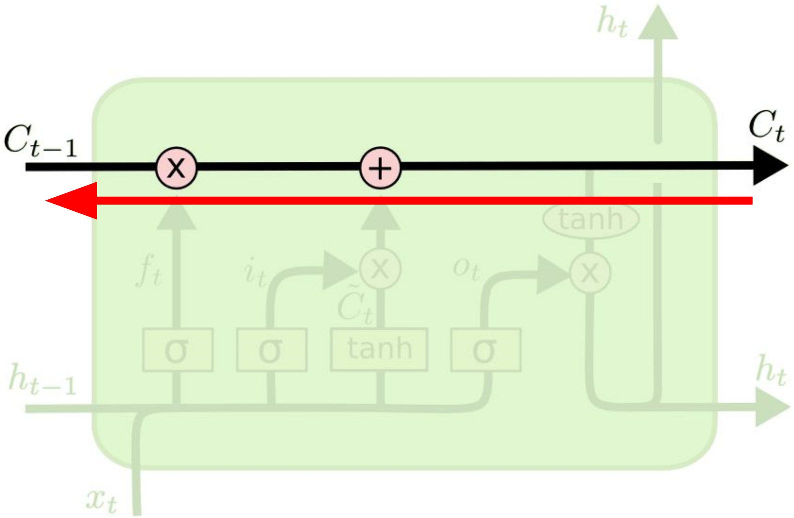



LSTM

Image credit to Christopher Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[1]Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

# LSTM Gradient Flow
[Hochreiter et al., 1997]




Core idea: Add a **cell state** to allow for uninterrupted gradient flow

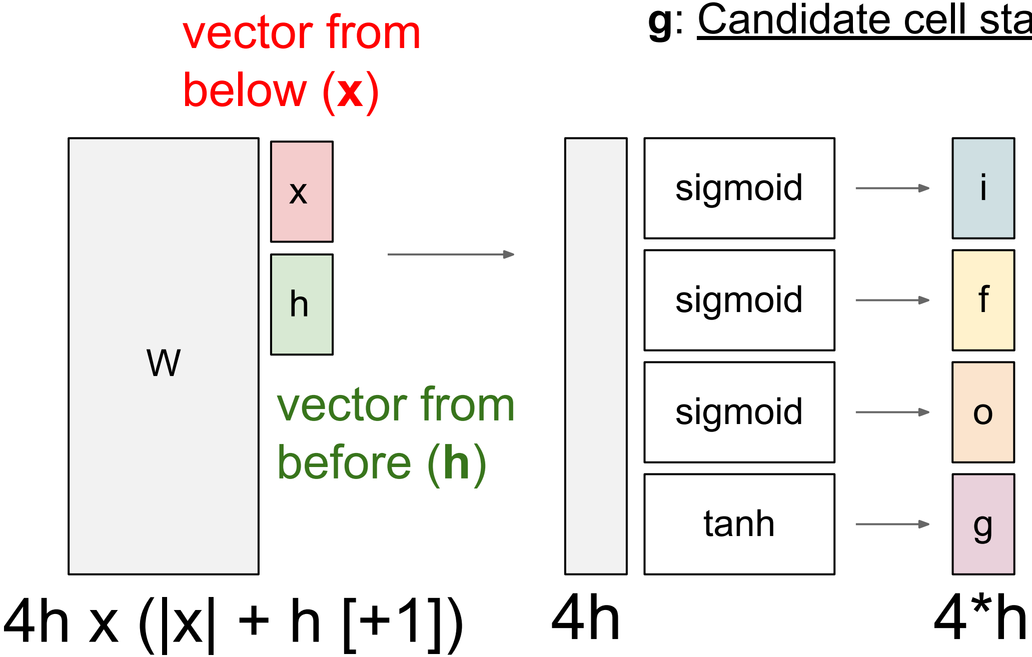
Note: Additive interactions are reminiscent of identity connections in ResNet[1]

Content credit: Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997
Image credit to Christopher Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[1]Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. CVPR '15

# Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

**f**: <u>Forget gate</u>, Whether to erase cell
**i**: <u>Input gate</u>, whether to write to cell
**o**: <u>Output gate</u>, How much to reveal from cell
**g**: <u>Candidate cell state</u> , What to write to cell (later $\tilde{C}_t$ )




$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

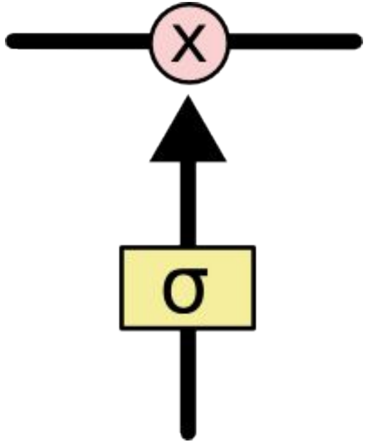
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997 Ft. CS231n and Fei-Fei Li & Justin Johnson & Serena Yeung

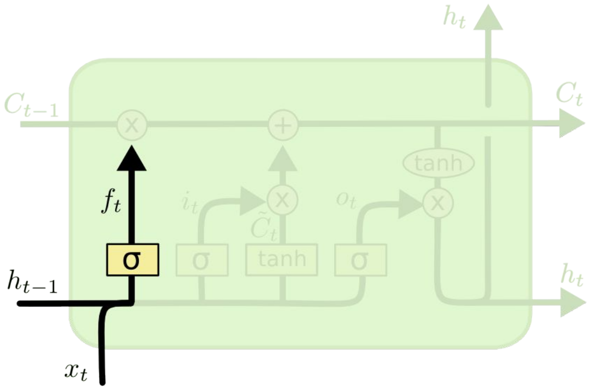
# LSTM Gating and Activation Functions
[Hochreiter et al., 1997]

- Sigmoid ($\sigma$) function  outputs number in (0,1)
  - Selectively attenuates (or ”gates”) signal
- Hyperbolic Tanget (tanh)?
  - Empirically better…
  - LSTM architecture already designed to avoid vanishing gradient
  - Hadamard product easily explodes if values are unbounded
    - So ReLU is out.




Content credit: Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Image credit to Christopher Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# LSTM Walkthrough
## [Hochreiter et al., 1997]

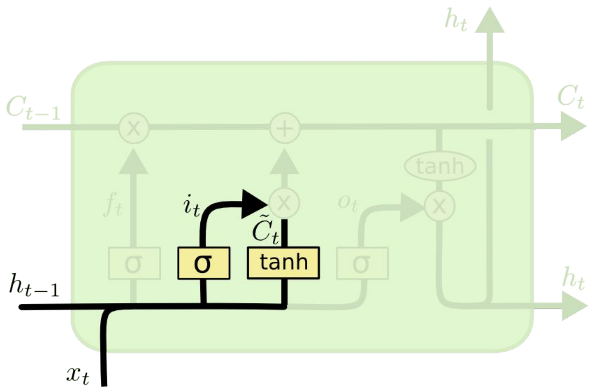



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- The **forget gate** $f_t$ erases information from the cell state
  - Imagine annotator altered a vertex; LSTM must forget its past trajectory

Content credit: Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Image credit to Christopher Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# LSTM Walkthrough
## [Hochreiter et al., 1997]

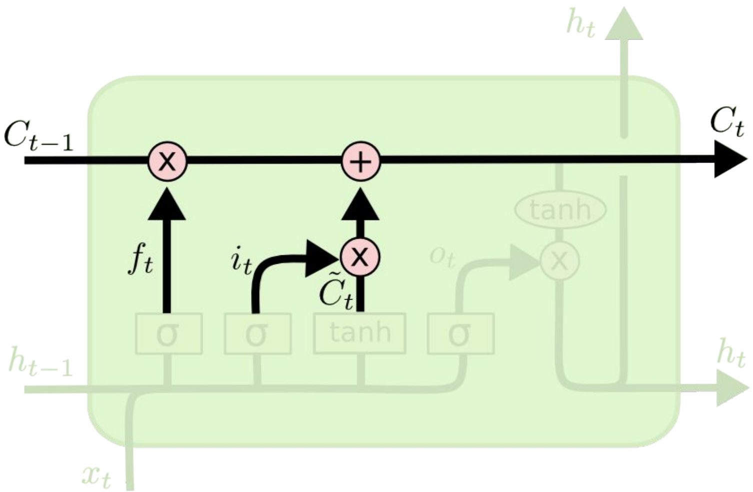



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- The **input gate** $i_t$ scales the candidate values to modulate how much to update the cell state
- The **candidate cell state** $\tilde{C}_t$ represents the new values which will replace/augment the existing cell state
  - = **Output of a vanilla RNN**

Content credit: Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Image credit to Christopher Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# LSTM Walkthrough
[Hochreiter et al., 1997]

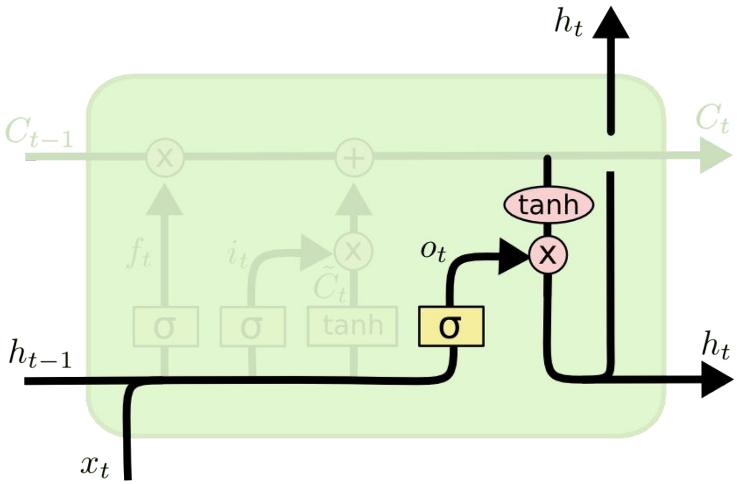



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- The **cell state** $C_t$ is the sum of the information *remembered* and the information *observed* which is deemed to be important

Content credit: Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Image credit to Christopher Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# LSTM Walkthrough
[Hochreiter et al., 1997]




$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- The **output gate** $o_t$ determines what information from the cell state the cell wishes to output
- The cells state $C_t$ is squashed by a hyperbolic tangent before being gated
- If we wish to make an inference $y_t$, simply pass $h_t$ through an fc classifier layer

Content credit: Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997
Image credit to Christopher Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# LSTM Summary
## [Hochreiter et al., 1997]

- Sigmoid functions clearly model gating
- Tanh empirically works better than ReLU
- Cell state allows for long(er)-term dependency modeling
- Other variants: peepholes, projection layers, and more
  - Is LSTM redundant? (**GRUs)**

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o) \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

Content credit: Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

# Outline

1. Review
2. Intro to Polygon-RNN
3. Modeling Sequences
    1. Vanilla RNN
    2. Training and Problems
    3. LSTM
    4. **Convolutional LSTM**
4. Vertex Prediction using Conv-LSTM

# Convolutional LSTM

Xingjian, Chen, Wang, Yeung, Wong, and Woo.; NIPS ‘15

# Modeling  Sequences of 2D Observations

- Capture spatio-temporal relationships
- Reduce parameter space

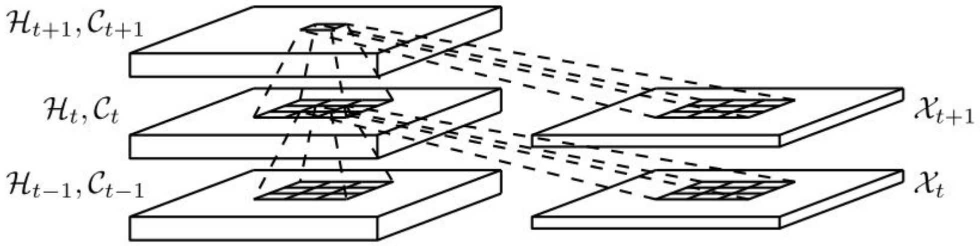



Figure 2: Inner structure of ConvLSTM

# Equations:

Vanilla LSTM

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i)\\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f)\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)\\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o)\\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

Convolutional LSTM

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i)\\
f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f)\\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c)\\
o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o)\\
\mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t)
\end{aligned}
$$

# Comments

- Initialize cell and hidden states to 0
  - “Total ignorance” of the past
- Pad the hidden states
  - “Same” type convolution
  - Represents the ignorance of the outside world
- Larger convolutional kernels can capture greater activity
- Only proven to work for low-res features (16x16 to 28x28)

# RNN Review

- RNNs allow for a lot of **flexibility** in architecture design
- Vanilla RNNs are **simple** but don't work very well
- Backward flow of gradients in RNN can explode or vanish.
  - Exploding? -> Clip gradients.
  - Vanishing? -> use additive interactions (LSTM, GRU, etc.)
- Better/simpler/faster architectures are a hot topic of current research
  - Convolutional LSTM improves on over-parametrized LSTM BUT  requires low-res images
- Better understanding (both theoretical and empirical) is needed.

# Back to Polygon-RNN

# Polygon-RNN Steps

1. Extract features using modified VGG-16
2. **Predict polygon vertices using a 2-layer convolutional LSTM (16 channel hidden layer)**

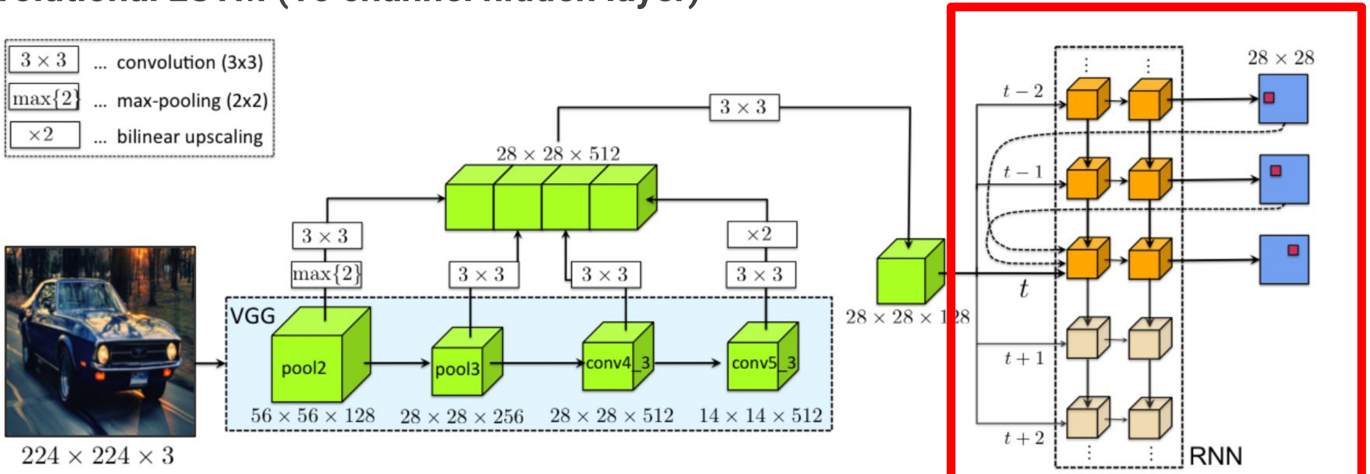
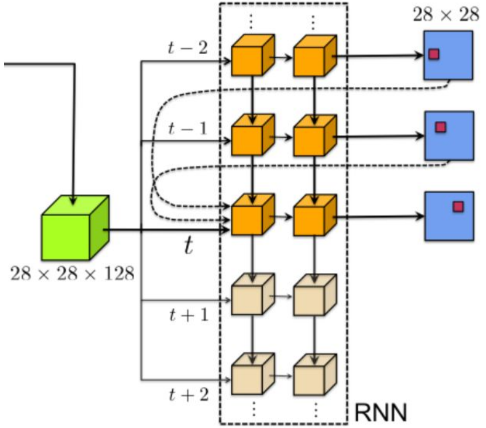



Figure 2. **Our Polygon-RNN model**. At each time step of the RNN-decoder (right), we feed in an image representation using a modified VGG architecture. Our RNN is a two-layer convolutional LSTM with skip-connection from one and two time steps ago. At the output at each time step, we predict the spatial location of the new vertex of the polygon.

# Vertex Prediction at step *t*

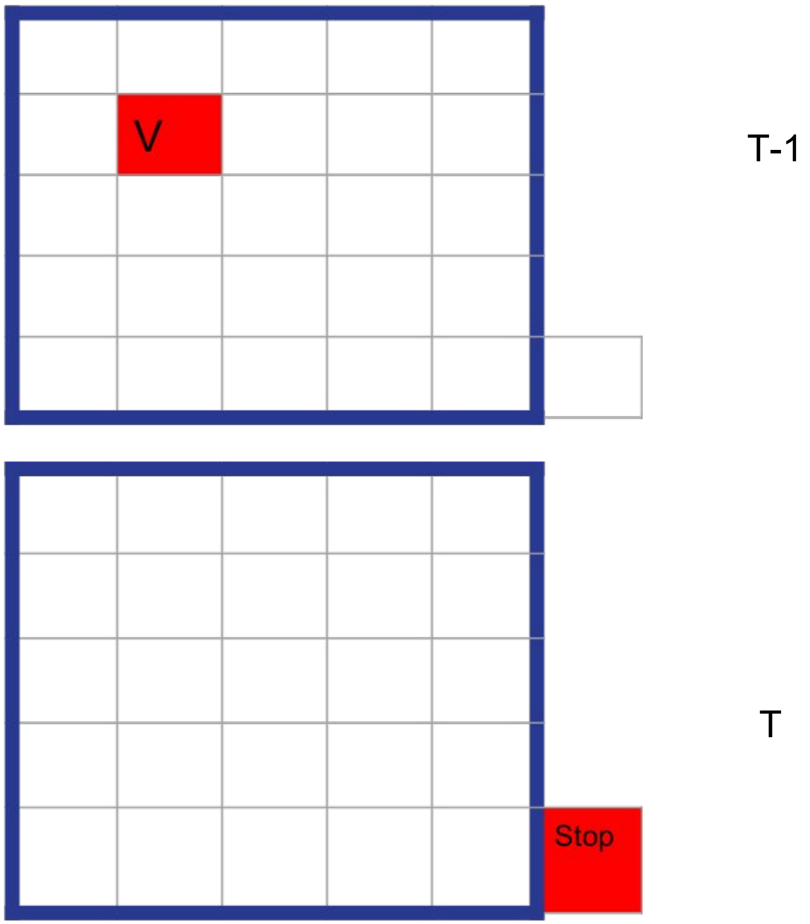
- Input $x$ is a concatenated tensor:
    - Extracted features $f$ from VGG (128 channels)
    - Vertices $y_1$, $y_{t-2}$, & $y_{t-1}$ or prior outputs of the ConvLSTM
        - *Similar to a Jordan Network!*

$$h_t = f_W(h_{t-1}, x_t)$$



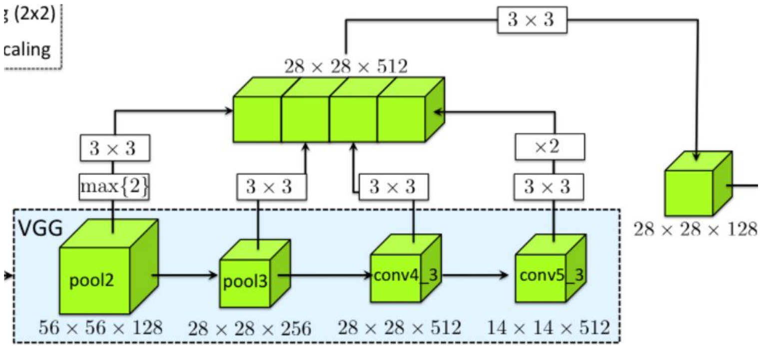
# Output Format

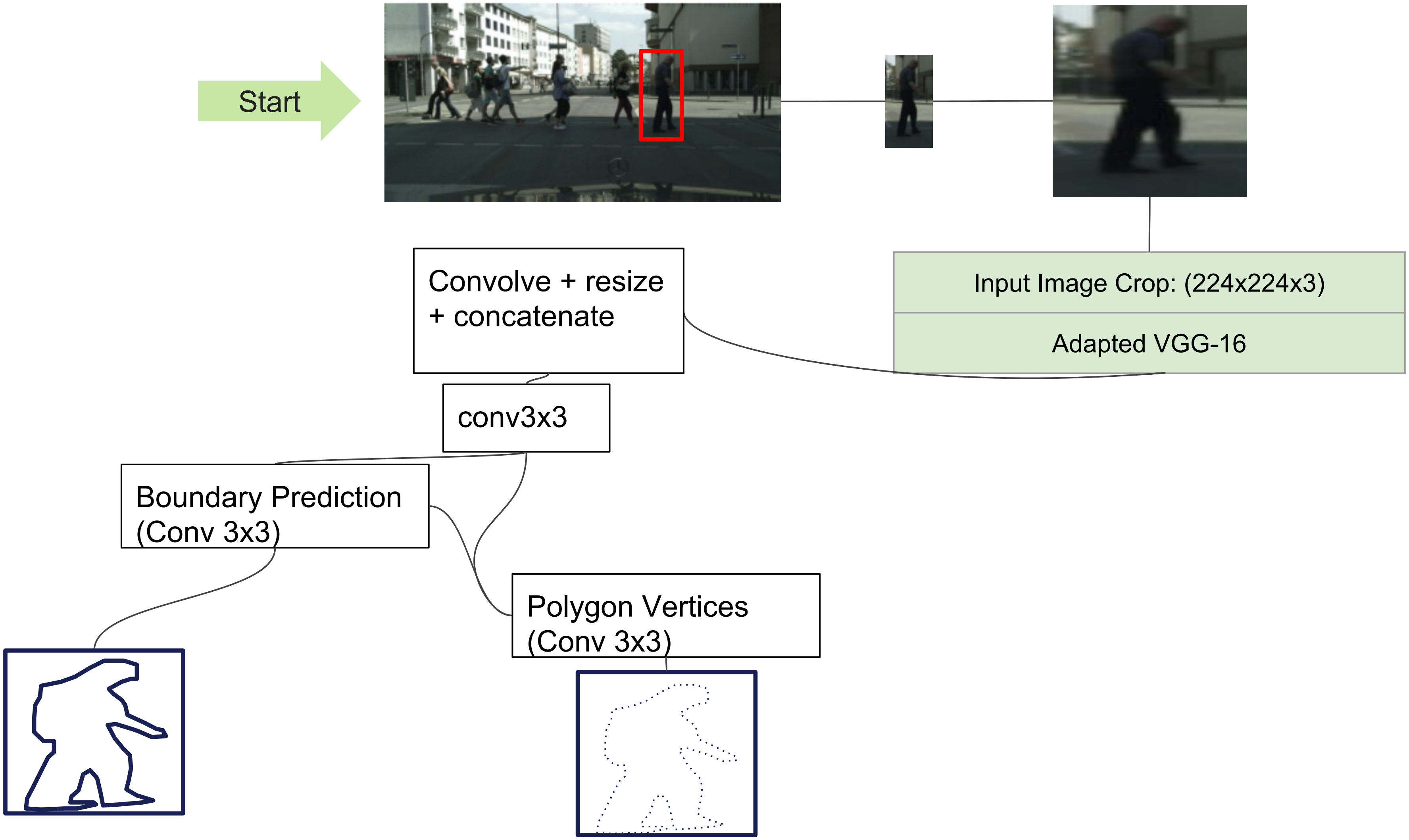
- Vertex prediction = Classification
- D x D+1 OHE grid
- D+1 for End-of-sequence token



# What about t=1?

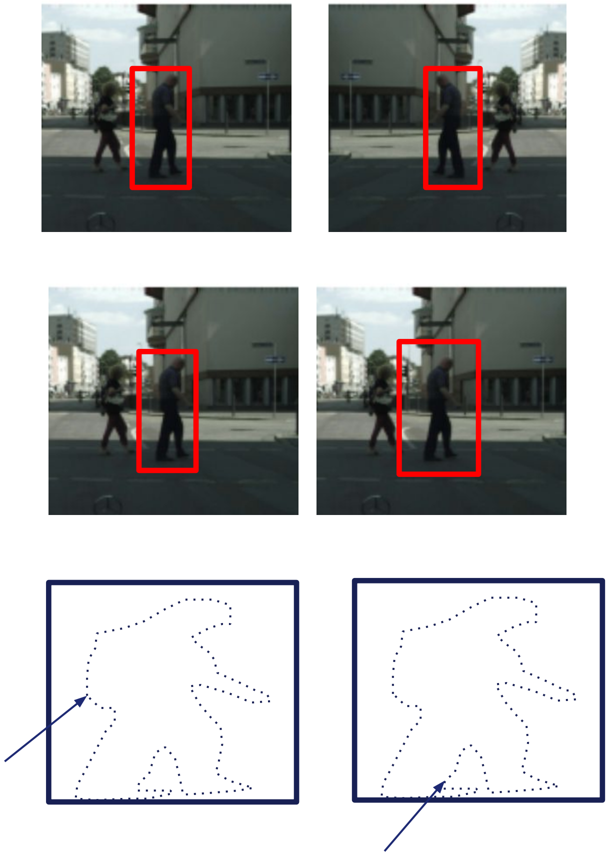
- First vertex of polygon is not uniquely defined
    - Use a **second** (identical) CNN for initial inference
    - *Note: the weights are NOT shared between the two VGG nets




Start
Input Image Crop: (224x224x3)
Adapted VGG-16
Convolve + resize + concatenate
conv3x3
Boundary Prediction (Conv 3x3)
Polygon Vertices (Conv 3x3)

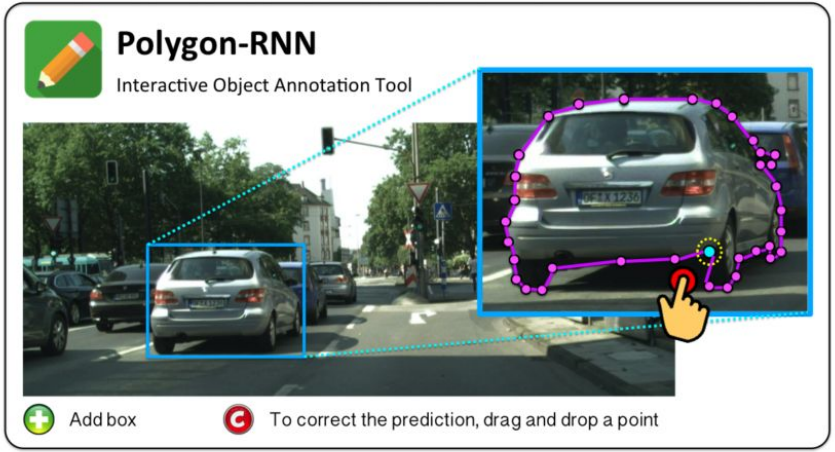
# Training

- Cross-Entropy at each time-step (for RNN)
  - No explicit distance metric
- Smooth ground-truth labels
- Data Augmentation
  - Random flips
  - Random context expansion (10%-20% of original BB)
  - Random selection of starting index



# Inference

- "Prediction Mode"
  - Automatically predict **all vertices** until end-of-sequence token is generated
- Annotator in the Loop
  - Predict vertices **one-at-a-time**
  - Annotator may move a vertex, which is then fed into the Conv-LSTM



# Evaluation Baselines

# DeepMask
## [Pinheiro et al., 2015]

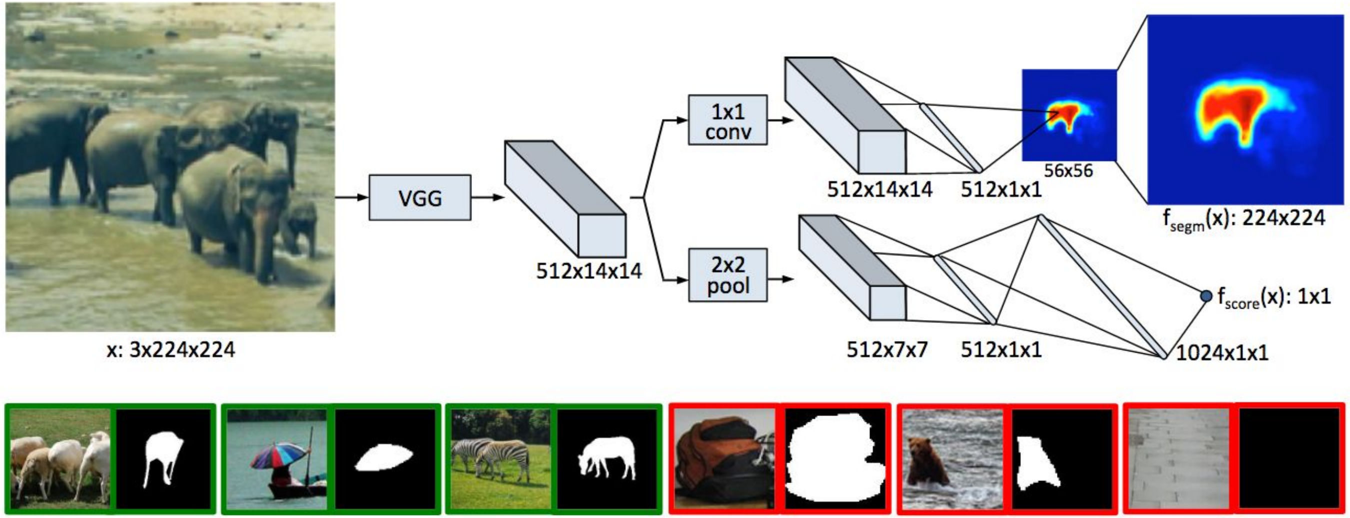
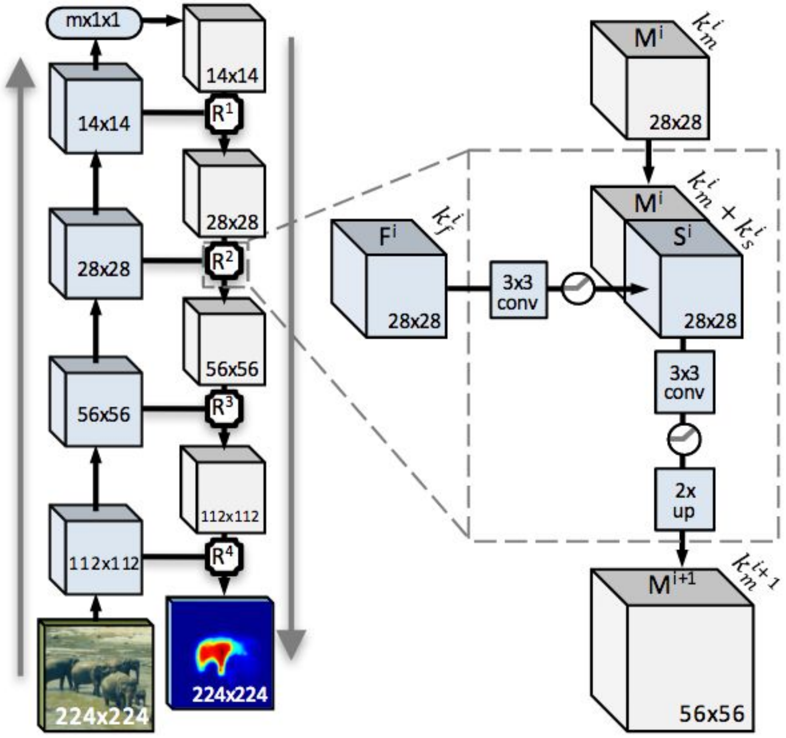



Figure 1: **(Top)** Model architecture: the network is split into two branches after the shared feature extraction layers. The top branch predicts a segmentation mask for the the object located at the center while the bottom branch predicts an object score for the input patch. **(Bottom)** Examples of training triplets: input patch $x$, mask $m$ and label $y$. Green patches contain objects that satisfy the specified constraints and therefore are assigned the label $y = 1$. Note that masks for negative examples (shown in red) are not used and are shown for illustrative purposes only.

P. O. Pinheiro, R. Collobert, and P. Dollar, "Learning to segment object candidates," in NIPS, 2015.

# SharpMask
[Pinheiro et al., 2016]

- Skip layers mainly average outputs, which hurts instance-level segmentation accuracy
- Refinement module w/ inputs $k_m$ and $k_f$:
  1. convolving $k_f$
  2. concatenate $k_f$ and $k_m$,
  3. Convolve to reduce number of channels,
  4. Up-sample



(c) proposed network (d) refinement module

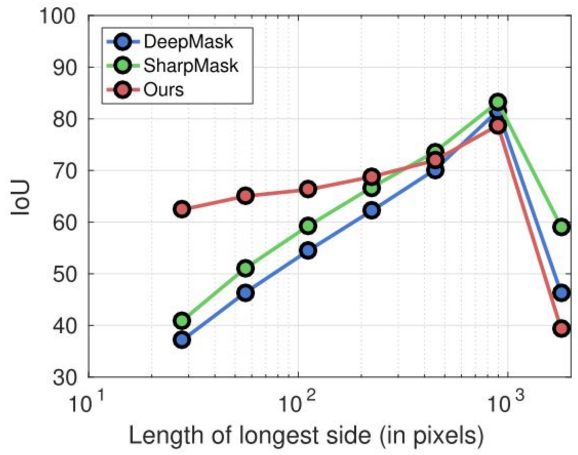
P. O. Pinheiro, T.-Y. Lin, R. Collobert, and Piotr Dollar, "Learning to refine object segments," in ECCV, 2016.

# Quantitative Evaluation

| Model | Bicycle | Bus | Person | Train | Truck | Motorcycle | Car | Rider | Mean |
|---|---|---|---|---|---|---|---|---|---|
| Square Box | 35.41 | 53.44 | 26.36 | 39.34 | 54.75 | 39.47 | 46.04 | 26.09 | 40.11 |
| Dilation10 | 46.80 | 48.35 | 49.37 | 44.18 | 35.71 | 26.97 | 61.49 | 38.21 | 43.89 |
| DeepMask [20] | 47.19 | 69.82 | 47.93 | 62.20 | 63.15 | 47.47 | 61.64 | 52.20 | 56.45 |
| SharpMask [20] | 52.08 | **73.02** | 53.63 | **64.06** | 65.49 | 51.92 | 65.17 | 56.32 | 60.21 |
| Ours | **52.13** | 69.53 | **63.94** | 53.74 | **68.03** | **52.07** | **71.17** | **60.58** | **61.40** |

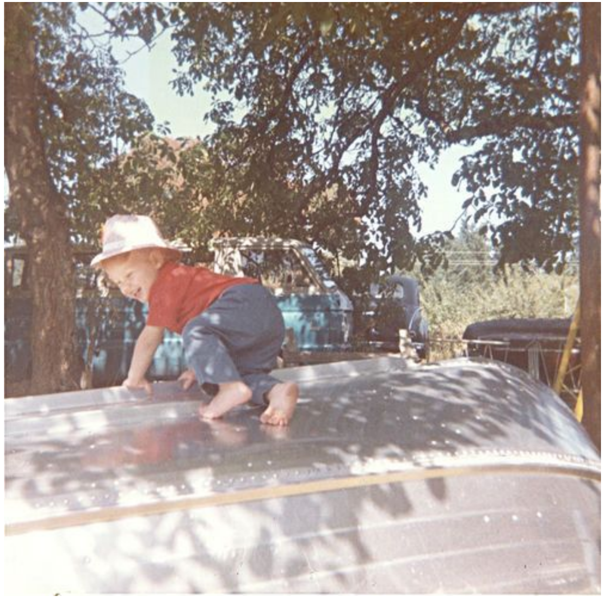
Table 3. **Performance** (IoU in %) on all the Cityscapes classes **without the annotator in the loop**.

*IoU is computed for each *instance*
**Analysis**: Polygon-RNN degrades for large instances due to small (28x28) output resolution



# Reminder:

- IoU is upper-bounded when using bilinear up-sampling
- Small output resolution of Polygon-RNN is a hindrance



| factor | mean IU |
| --- | --- |
| 128 | 50.9 |
| 64 | 73.3 |
| 32 | 86.1 |
| 16 | 92.8 |
| 8 | 96.4 |
| 4 | 98.5 |

Source: Long, Shelhammer, Darrell. Fully Convolutional Neural Networks for Semantic Segmentation

# Annotator in the Loop Results

Subset of Images

| | Grabcut | P-RNN (T=4) | P-RNN(T=1) |
|---|---|---|---|
| Avg. Time | 42.2 | **N/A** | **N/A** |
| Clicks | 17.5 | 5 | 9.6 |
| mIoU | 70.7% | 79.7% | 85.8% |

Full Dataset

| Threshold | Num. Clicks | Mean IOU |
|---|---|---|
| 1 | 15.79 | 84.74 |
| 2 | 11.77 | 81.43 |
| 3 | 9.39 | 78.40 |
| 4 | 7.86 | 75.79 |

Table 4. **Annotator in the loop:** Average number of corrections per instance and IoU, computed across all classes. Threshold indicates chessboard distance to the closest GT vertex.

**Note**: The authors claim that each click requires “comparable time” to Grabcut but do not report annotation time results, choosing instead to focus on the number of clicks. They do report that it takes ~250ms per rnn inference.

**Note 2:** Authors don’t compare # of clicks with GrabCut where the algorithm is initialized with a deeply-learned mask (e.g. SharpMask)

# Annotator in the Loop Results

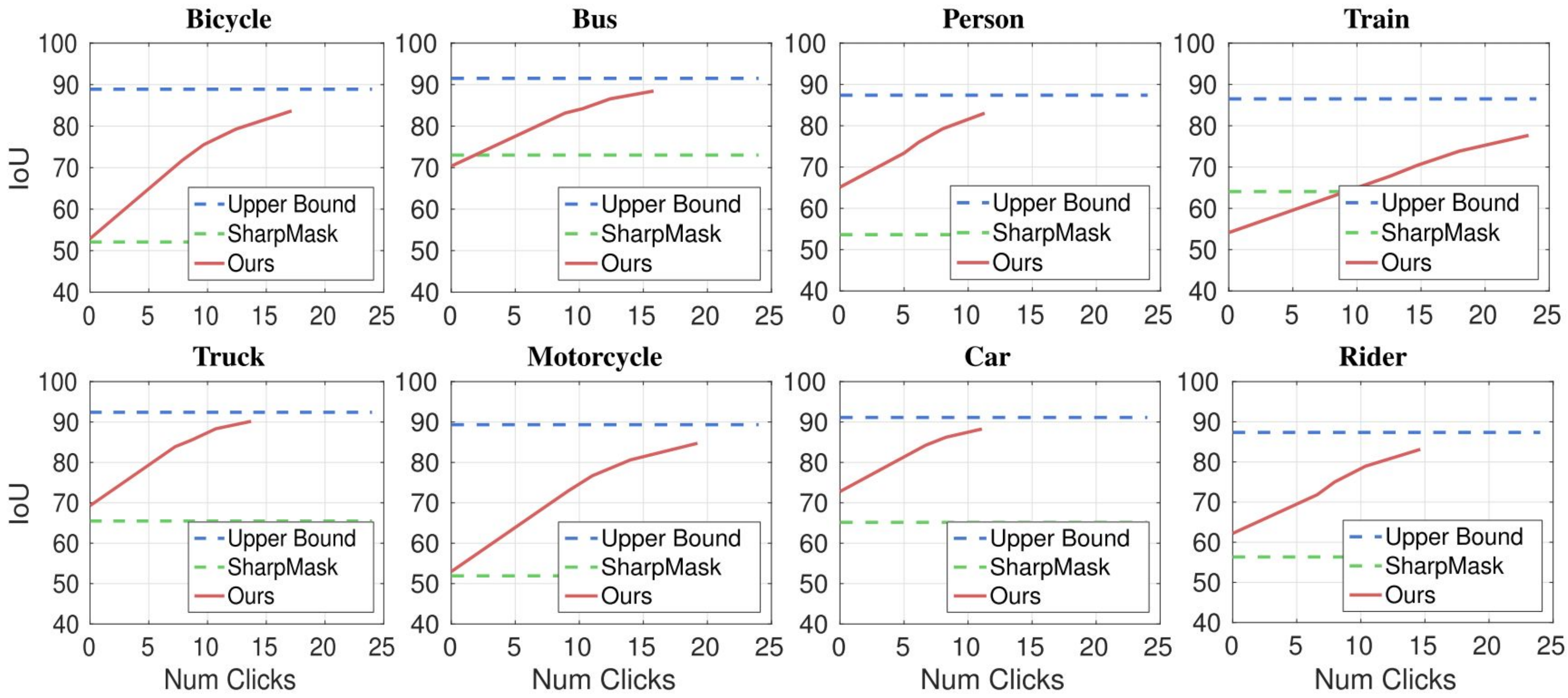



Figure 5. **Annotator in the loop:** We show IoU as a function of the number of clicks/corrections.

# Comparison with an Expert Annotator

| Method | Num. Clicks | IoU | Annot. Speed-Up |
|---|---|---|---|
| Cityscapes GT | 33.56 | 100 | - |
| Ann. full image | 79.94 | 69.5 | - |
| Ann. crops | 96.09 | 78.6 | - |
| Ours (Automatic) | 0 | 73.3 | No ann. |
| Ours (T=1) | 9.3 | 87.7 | x3.61 |
| Ours (T=2) | 6.6 | 85.7 | x5.11 |
| Ours (T=3) | 5.6 | 84.0 | x6.01 |
| Ours (T=4) | 4.6 | 82.2 | x7.31 |

Table 5. **Our model vs Annotator Agreement**: We hired a highly trained annotator to label *car* instances on additional 10 images (101 instances). We report IoU agreement with Cityscapes GT, and report polygon statistics. We compare our approach with the agreement between the human annotators.

# Testing on Kitti[1]

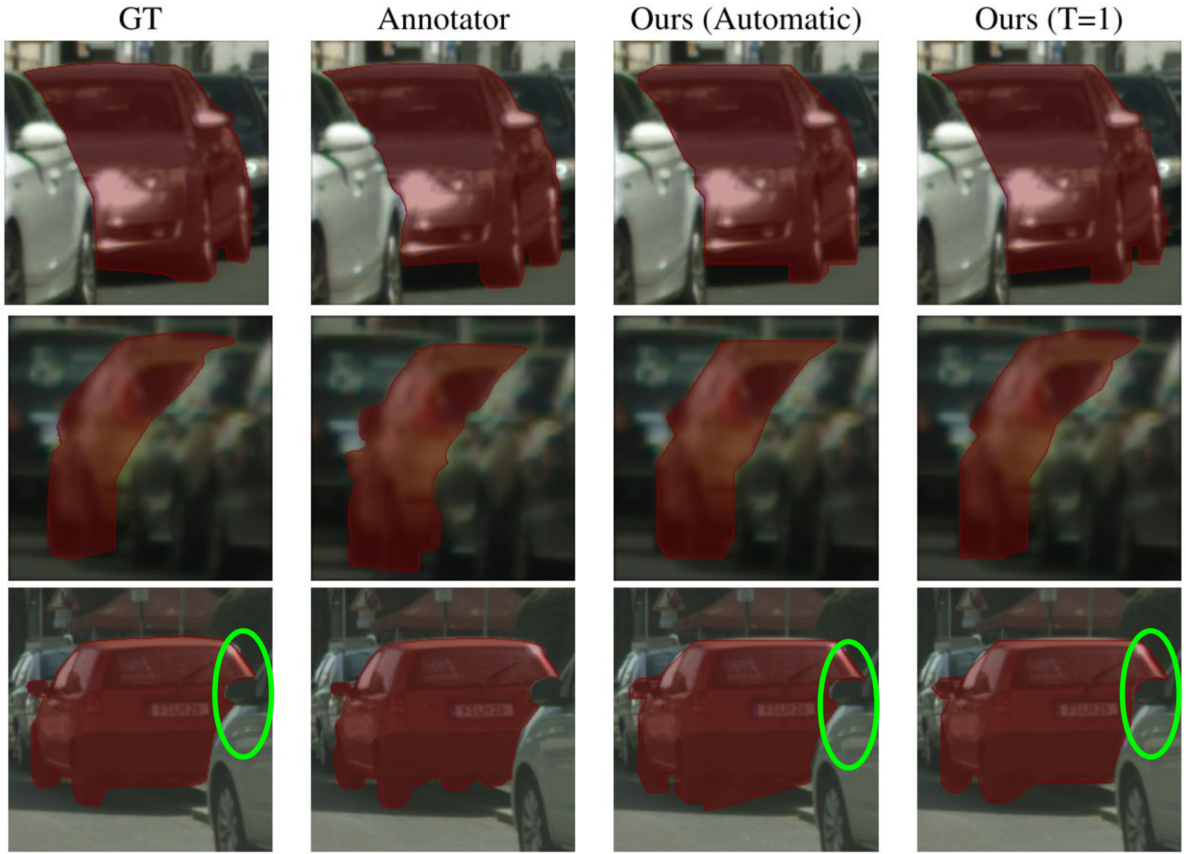
- Larger average instances put DeepMask and SharpMask back ahead of automatic Polygon-RNN

| Method | # of Clicks | IOU |
|---|---|---|
| DeepMask [20] | - | 78.3 |
| SharpMask [21] | - | 78.8 |
| Beat the MTurkers [4] | 0 | 73.9 |
| Ours (Automatic) | 0 | 74.22 |
| Ours (T=1) | 11.83 | 89.43 |
| Ours (T=2) | 8.54 | 87.51 |
| Ours (T=3) | 6.83 | 85.70 |
| Ours (T=4) | 5.84 | 84.11 |

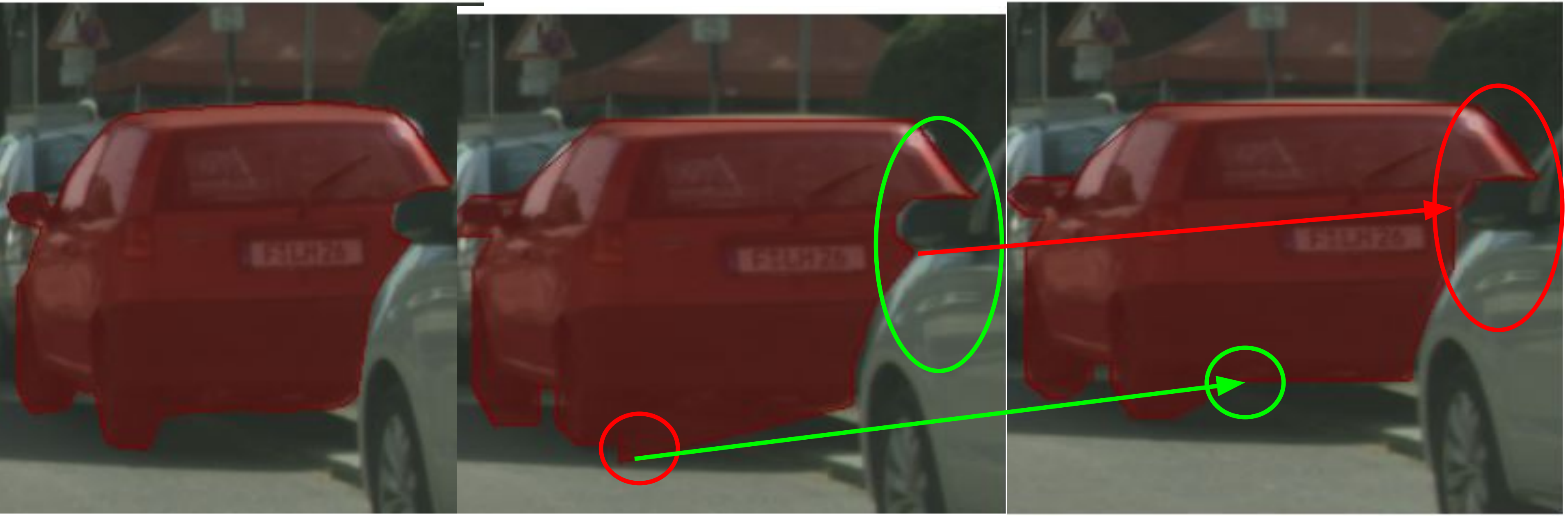
Table 6. Car annotation results on the **KITTI dataset**.

[1] Geiger, Lenz, and Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite; CVPR, '12

# Qualitative Results




# Qualitative Results



GT Automatic T=1

- Authors' threshold-based correction is limited.
- Memory of LSTM is complicated

# Summary and Extensions

- Polygon-RNN frames vertex prediction as a **binary classification** task
- Interesting that no explicit distance metric is used for the loss
    - Only cross-entropy and local smoothing
- Annotation speedup of 4.74 (measured in clicks)
    - Though per-click speed may be slower to allow for RNN inference time…

- Questions and extensions:
    - Close analysis: what amount of error is attributable to the **feature extraction** module, and what amount is due to weaknesses in the **RNN** inference module?
        - Could swap VGG for e.g. ResNet, DenseNet, or a stacked hourglass network
        - Could inference be improved at **higher resolution** using larger, stacked, or dilated ConvLSTM kernels?
        - Could we adapt e.g. Grid LSTM to better model spatial dependencies?
    - Should thresholding analysis be done using an area-approach rather than the closest **vertex?**
    - Could the framework be extended from “**things**” to “**stuff**” (semantic segmentation boundaries)?
    - How does this model perform on more complicated datasets (e.g. ADE20K)?
    - Amodal Polygon-RNN?

# Appendix:
# Other Cool RNN Applications in Vision

# RNNs on Spatial Sequences

(a) smoothing (b) denoising (c) inpainting (d) color interpolation

Liu, Pan, Yang, Learning Recursive Filters for Low-Level Vision via a Hybrid Neural Network; ECCV ‘16

# RNN as a CRF

A single iteration of the mean field algorithm can be modeled as a stack of CNN filters

A CRF post-processor can be constructed by sharing the filter weights

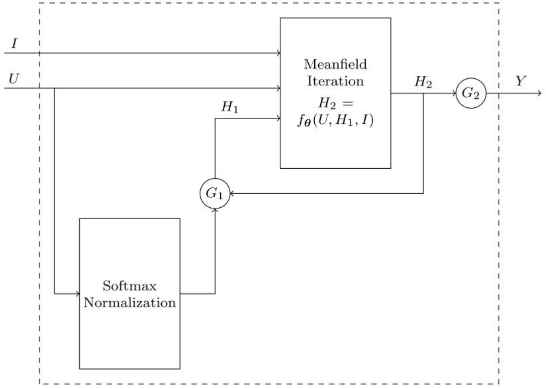
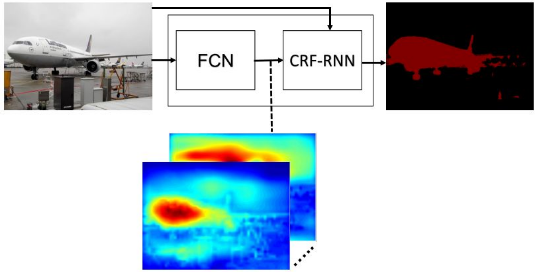
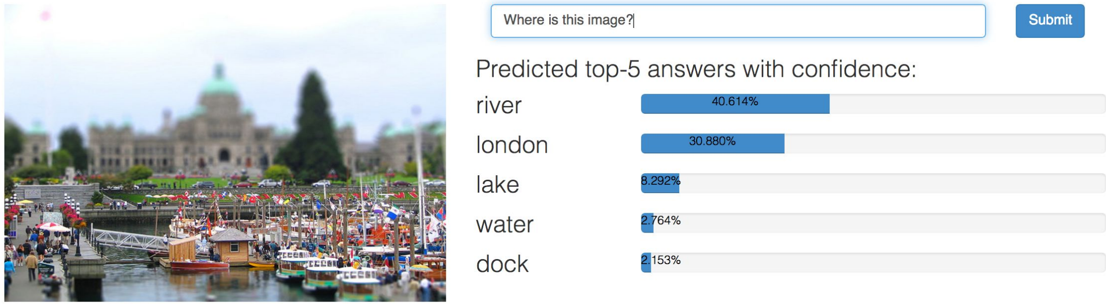



Figure 2. **The CRF-RNN Network.** We formulate the iterative mean-field algorithm as a Recurrent Neural Network (RNN). Gating functions $G_1$ and $G_2$ are fixed as described in the text.




Zheng, Jayasumana, Romera-Paredes, Vineet, Su, Du, Huang, Torr; Conditional Random Fields as Recurrent Neural Networks; ICCV '15

# VQA (Module 2)




With surprising accuracy for some questions …


…. but replete with learned biases.

Zheng, Jayasumana, Romera-Paredes, Vineet, Su, Du, Huang, Torr; Conditional Random Fields as Recurrent Neural Networks; ICCV '15