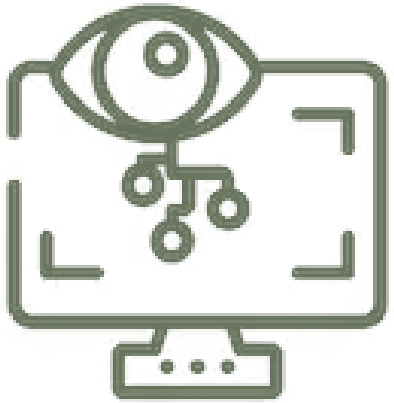


CS 554
Computer Vision

Deep Learning

Hamdi Dibeklioğlu

So, What is DEEP Machine Learning

A few different ideas:

- **(Hierarchical) Compositionality**
  - Cascade of non-linear transformations
  - Multiple layers of representations
- **End-to-End Learning**
  - Learning (goal-driven) representations
  - Learning feature extraction
- **Distributed Representations**
  - No single neuron “encodes” everything
  - Groups of neurons work together

Slide by Dhruv Batra

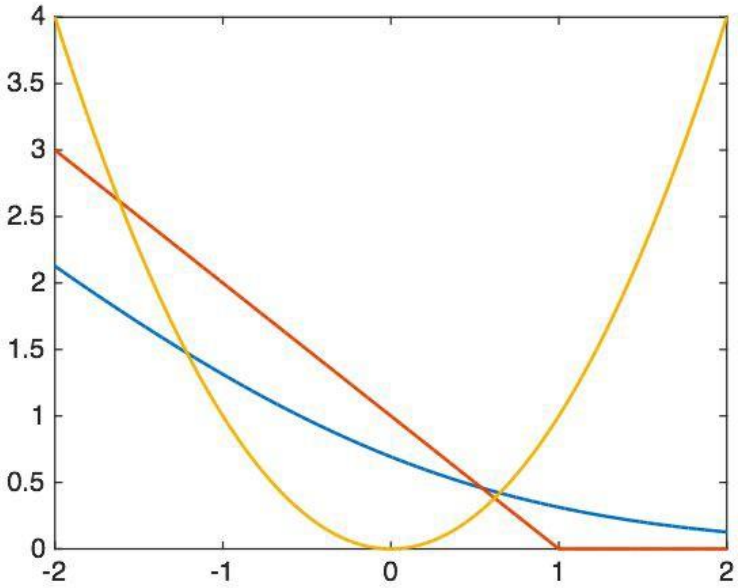
$$f(x, W) = Wx$$

- Loss function
- Optimization
- Convolutional Nets
- Recurrent Nets

Loss Functions

Loss functions

- There are many different loss functions



- Log Loss / Cross Entropy
- Hinge Loss
- Square Loss

Classification Losses

Hinge Loss/Multi class SVM Loss

$$SVMLoss = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

- s_j – Computed score of the training example for jth class.
- y(i) - Ground truth label for ith training example.

Classification Losses

Cross Entropy Loss/Negative Log Likelihood

$$CrossEntropyLoss = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

- s_j – Computed score of the training example for jth class.
- y(i) - Ground truth label for ith training example.

Regression Losses

Mean Square Error/Quadratic Loss/L2 Loss

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$

- n - Number of training examples.
- i - ith training example in a data set.
- y(i) - Ground truth label for ith training example.
- y_hat(i) - Prediction for ith training example.

Regression Losses

Mean Absolute Error/L1 Loss

$$MAE = \frac{\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid}{n}$$

- n - Number of training examples.
- i - ith training example in a data set.
- y(i) - Ground truth label for ith training example.
- y_hat(i) - Prediction for ith training example.

Regression Losses

Mean Bias Error

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}$$

- n - Number of training examples.
- i - ith training example in a data set.
- y(i) - Ground truth label for ith training example.
- y_hat(i) - Prediction for ith training example.

Weight Regularization

$$L = \frac{1}{N}\sum_{i=1}^{N} \text{Loss}_i + \lambda R(W)$$

\lambda = regularization strength (hyperparameter)

Some reg. types:

L2 regularization

L1 regularization

Elastic net (L1 + L2)

…

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

L2 regularization: motivation

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

L2 regularization: motivation

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

Which one does L2 regularization choose?

$$w_1^T x = w_2^T x = 1$$

L2 regularization: motivation

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

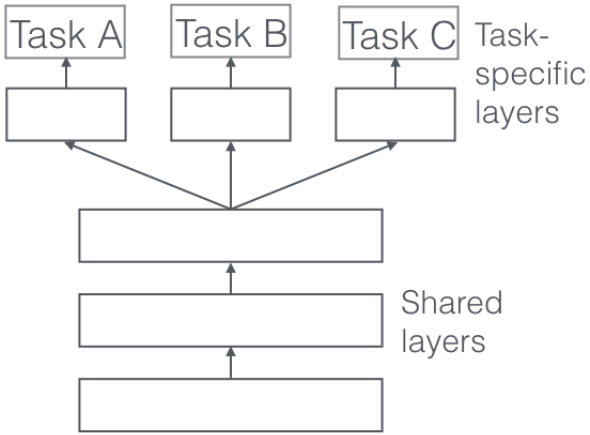
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

Why does it make sense?

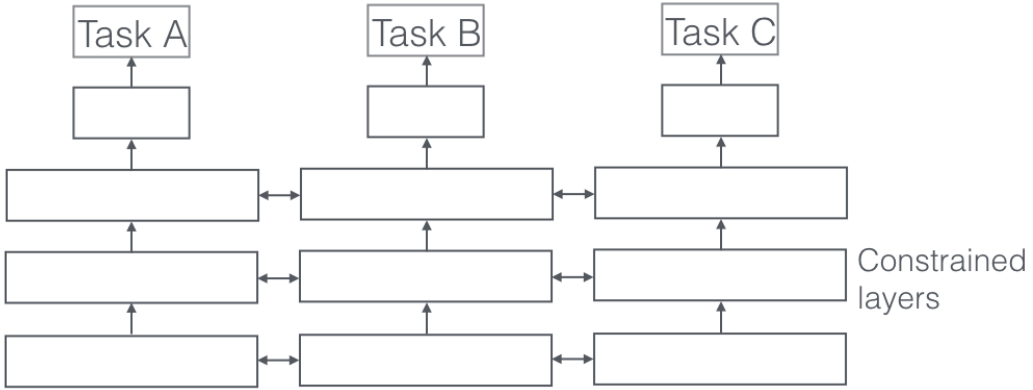
$$w_1^T x = w_2^T x = 1$$

Multi-task Learning

Jointly minimize the losses of different tasks



Hard parameter sharing for multi-task learning in deep neural networks

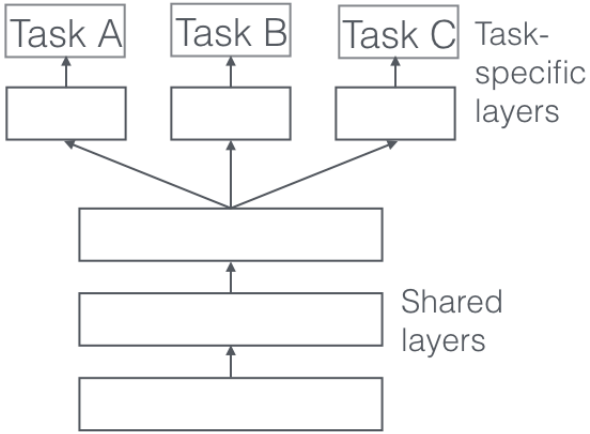


Soft parameter sharing for multi-task learning in deep neural networks

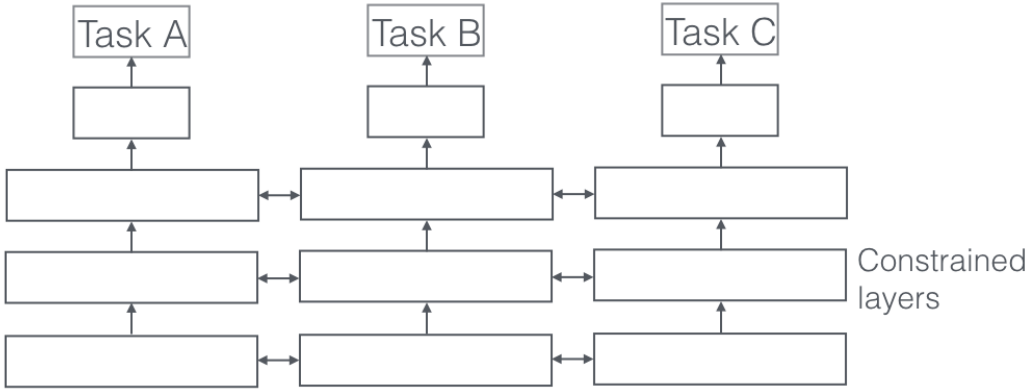
Multi-task Learning

Jointly minimize the losses of different tasks (combine loss terms)

$$L = l_a + \alpha l_b + \beta l_c + \cdots$$



Hard parameter sharing for multi-task learning in deep neural networks

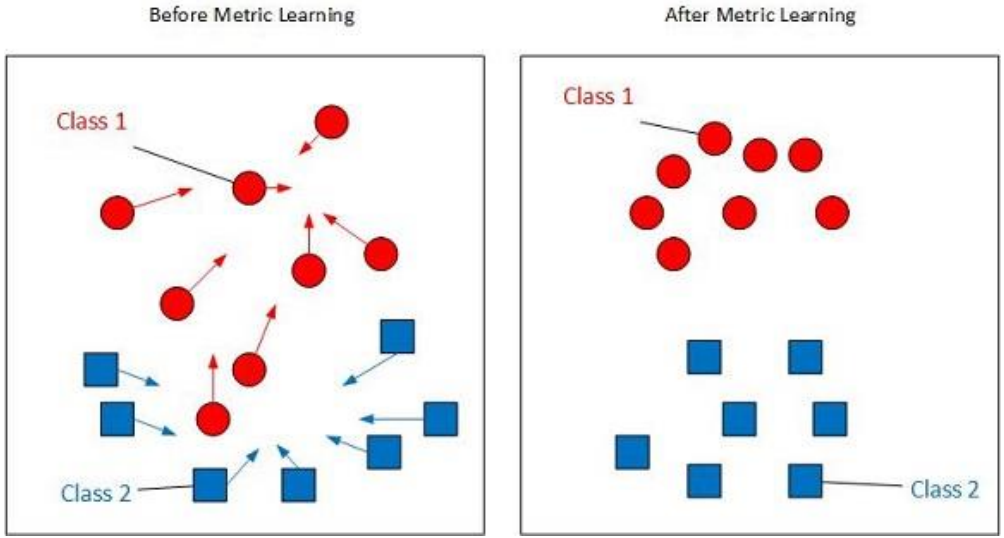


Soft parameter sharing for multi-task learning in deep neural networks

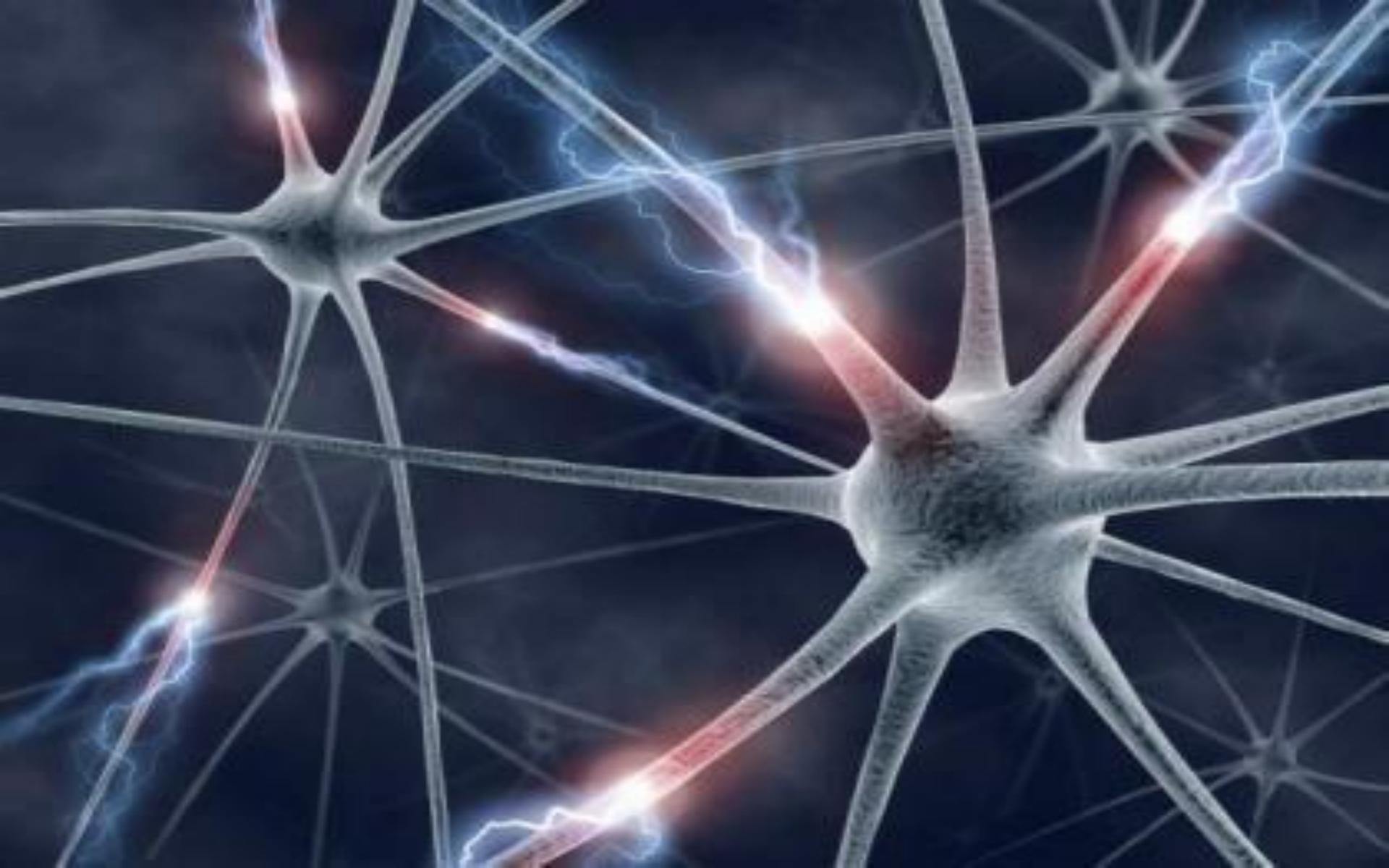
Metric/Contrastive Learning

Learn *distinctiveness*

1. A distance-based loss function (as opposed to prediction error-based loss functions like Logistic loss or Hinge loss used in Classification).
2. Like any distance-based loss, it tries to ensure that semantically similar examples are embedded close together.
3. Defined based on pairs (+/- class pairs) or groups of samples.



$$L_i = \sum_{i \neq j} \left\| w^T x_{i,c1} - w^T x_{j,c1} \right\| - \sum_{k} \left\| w^T x_{i,c1} - w^T x_{k,c2} \right\|$$



Neural Networks

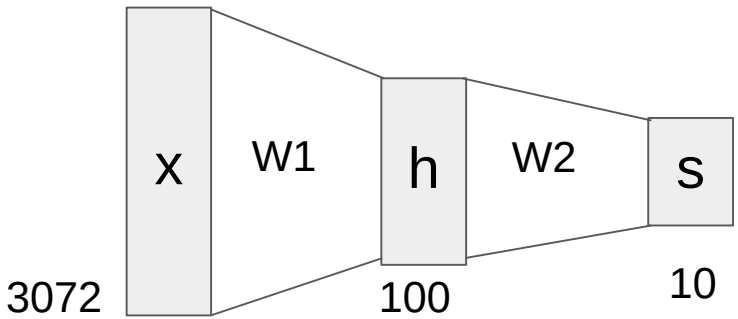
Linear score function: $f = Wx$

2-layer Neural Network: $f = W_2 \max(0, W_1 x)$

Neural Networks

Linear score function: $f = Wx$

2-layer Neural Network: $f = W_2 \max(0, W_1 x)$



Neural Networks

Linear score function:

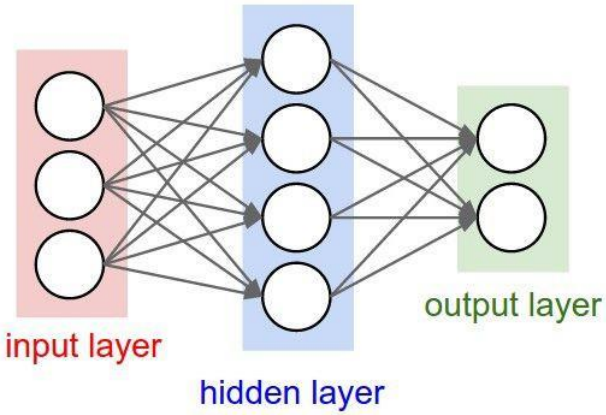
$$f = Wx$$

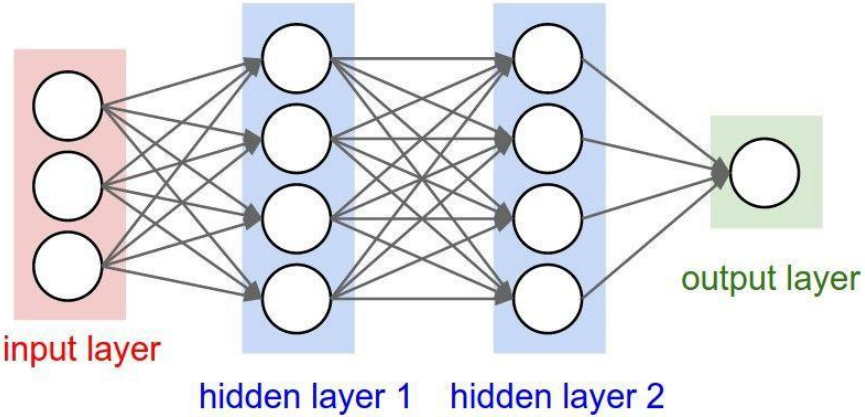
2-layer Neural Network
or 3-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Neural Networks: Architectures



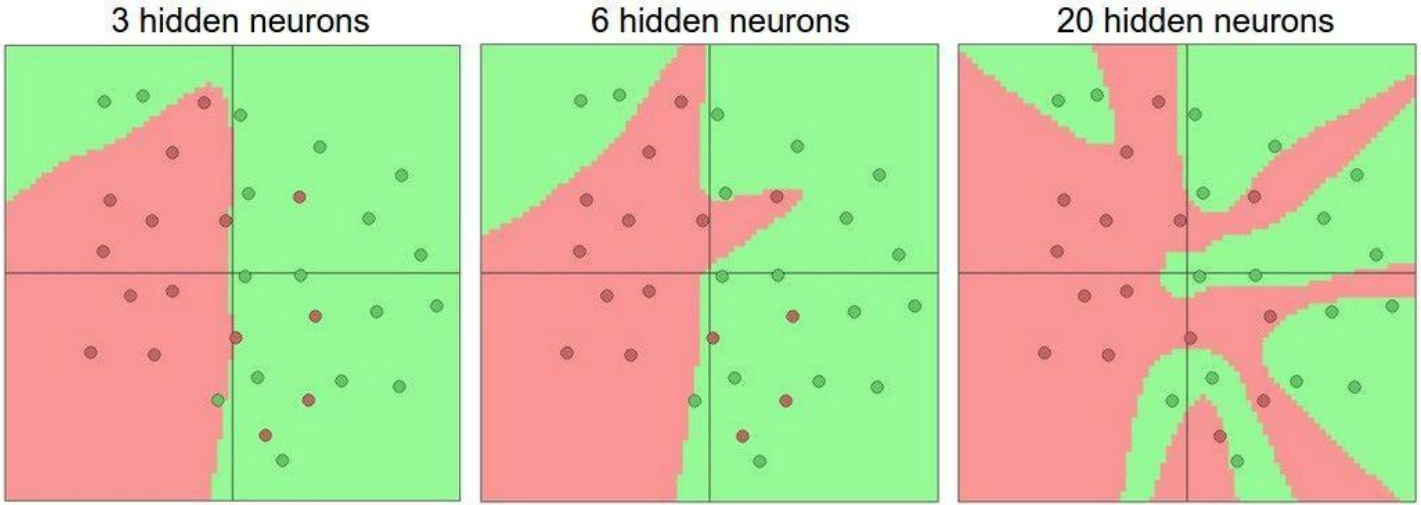


"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

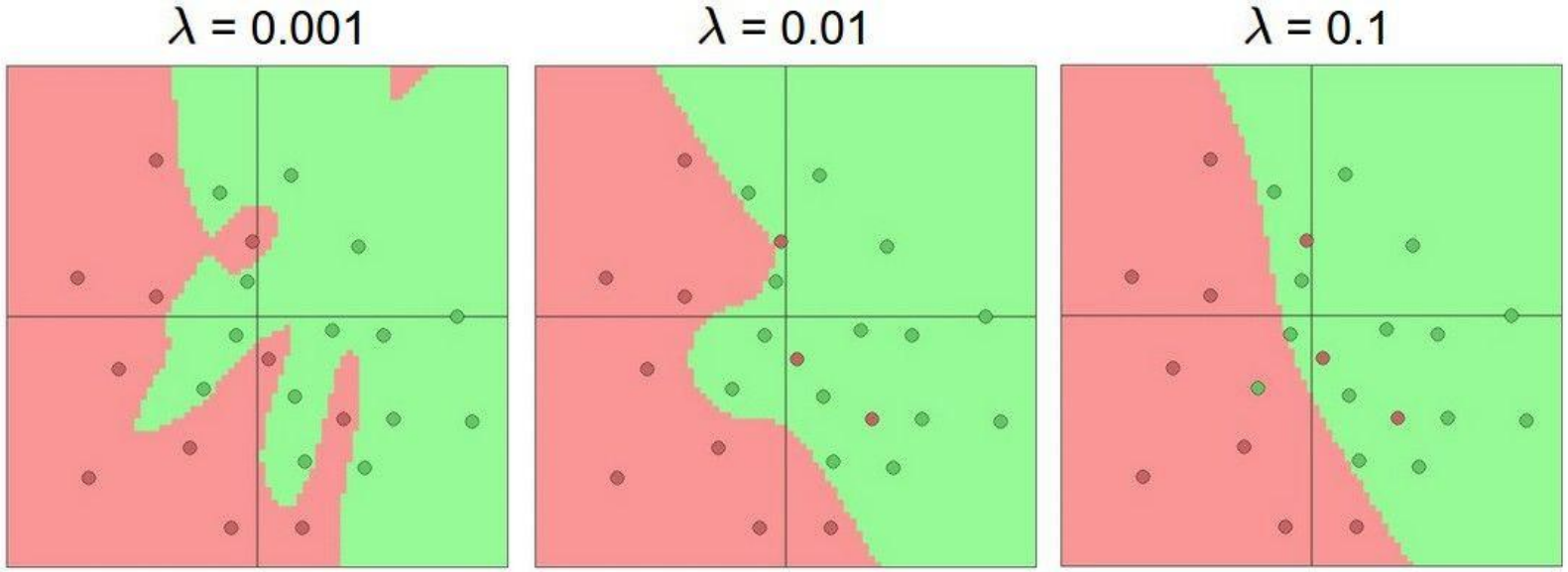
"Fully-connected" layers

Setting the number of layers and their sizes



more neurons = more capacity

Do not use size of neural network as a regularizer. Use stronger regularization instead:

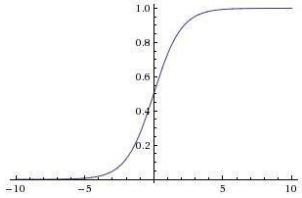


Activation Functions

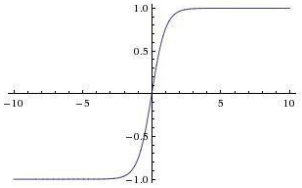
Activation Functions

Sigmoid

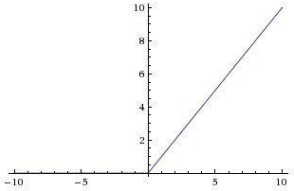
$$\sigma(x) = 1/(1+e^{-x})$$



tanh tanh(x)

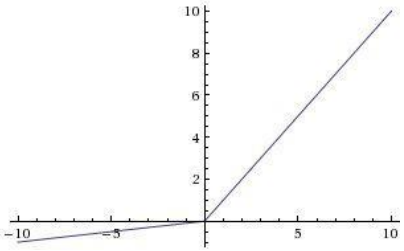


ReLU max(0,x)



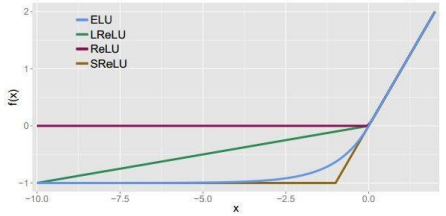
Leaky ReLU

max(0.1x, x)



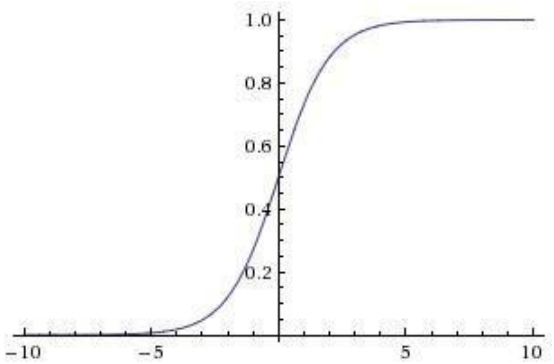
Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$



Activation Functions

$$\sigma(x) = 1/(1+e^{-x})$$



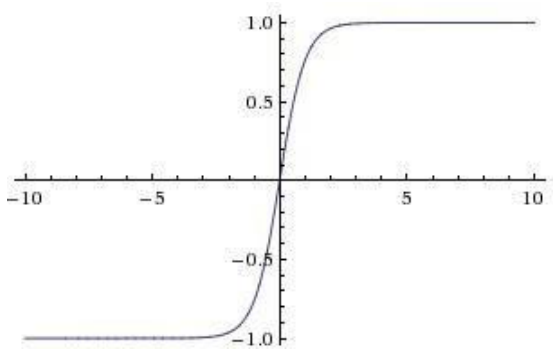
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit computationally expensive

Activation Functions

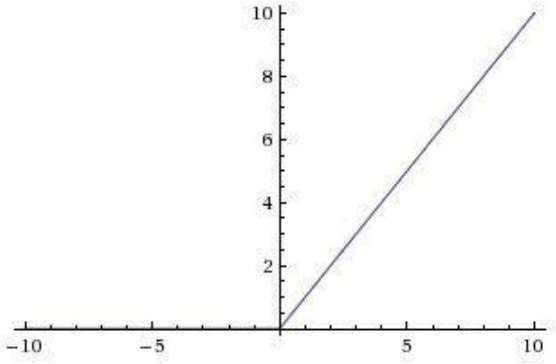


tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

[LeCun et al., 1991]

Activation Functions



ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

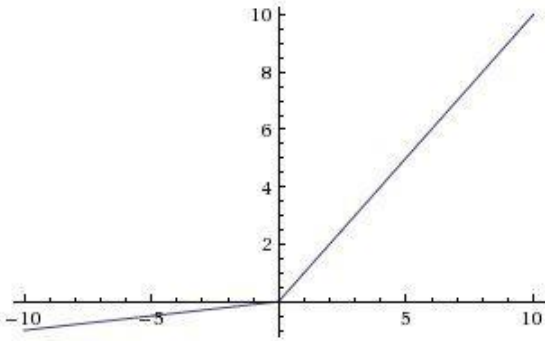
- Not zero-centered output
- An annoyance:

hint: what is the gradient when x < 0?

[Krizhevsky et al., 2012]

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



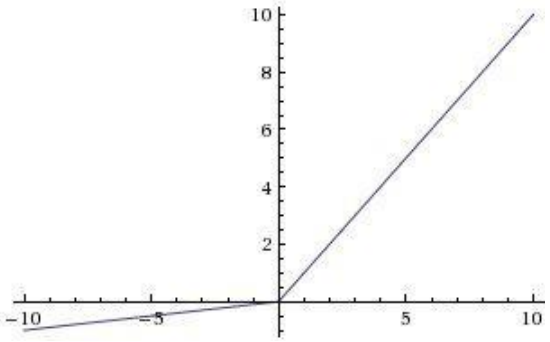
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

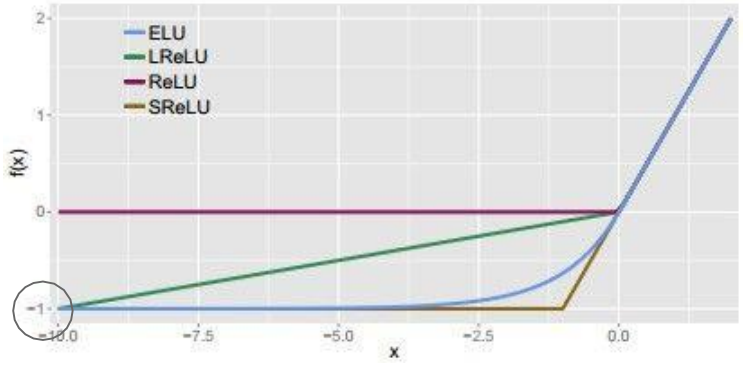
$$f(x) = \max(\alpha x, x)$$

backprop into \alpha (parameter)

Activation Functions

[Clevert et al., 2015]

Exponential Linear Units (ELU)



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right) & \text{if } x \leq 0 \end{cases}$$

- All benefits of ReLU
- Does not die
- Closer to zero mean outputs

- Computation requires exp()

Maxout “Neuron”

[Goodfellow et al., 2013]

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

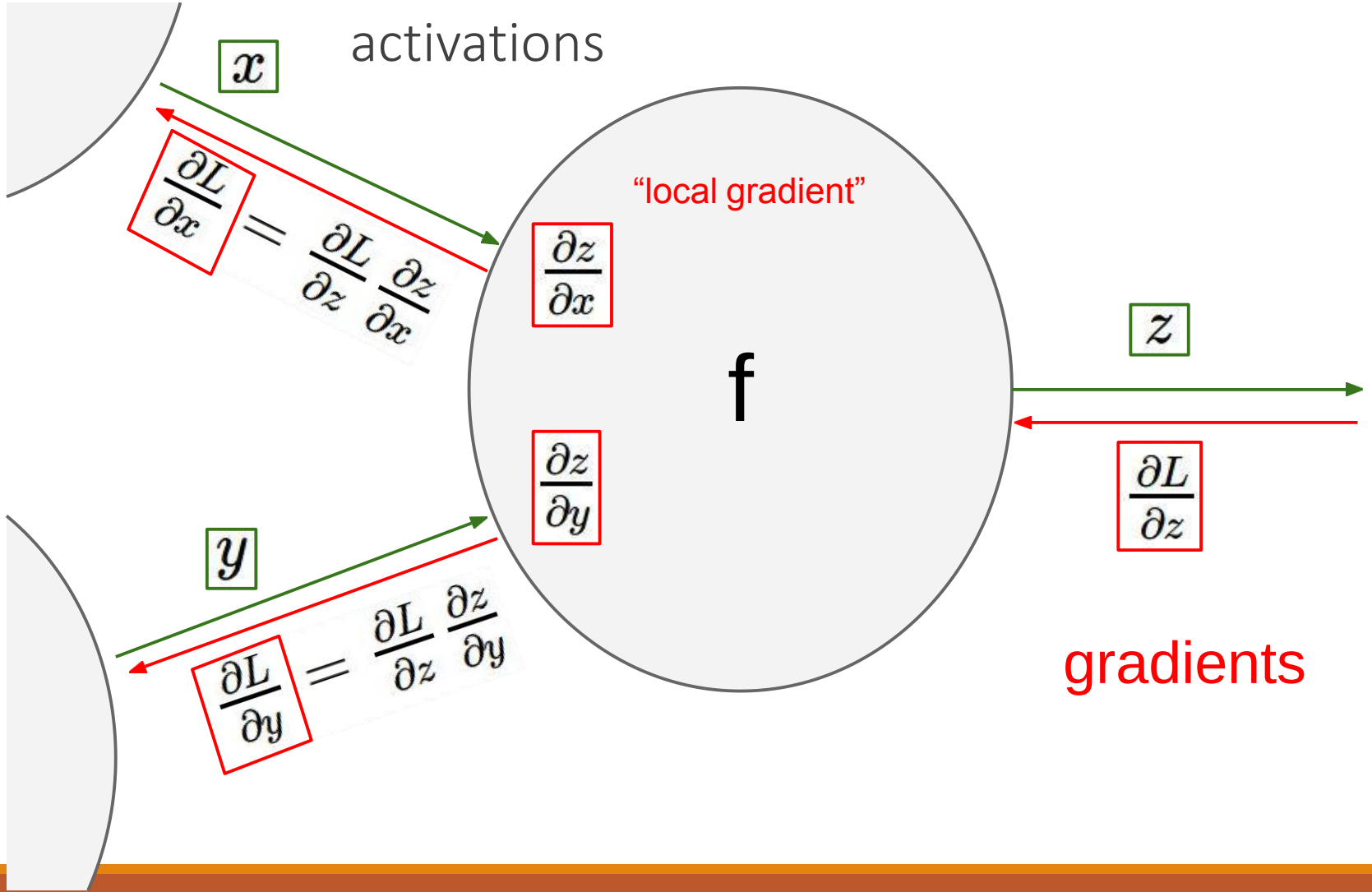
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Problem: doubles the number of parameters/neuron :(

In practice

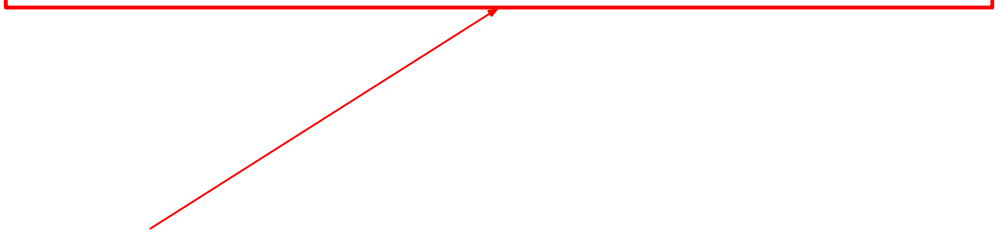
- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid

Parameter Updates


activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Training a neural network, main loop:

```
while True:
    data_batch = dataset.sample_data_batch()
    loss = network.forward(data_batch)
    dx = network.backward()
    x += - learning_rate * dx
```



simple gradient descent update

Optimize the parameters using one of the SGD variants

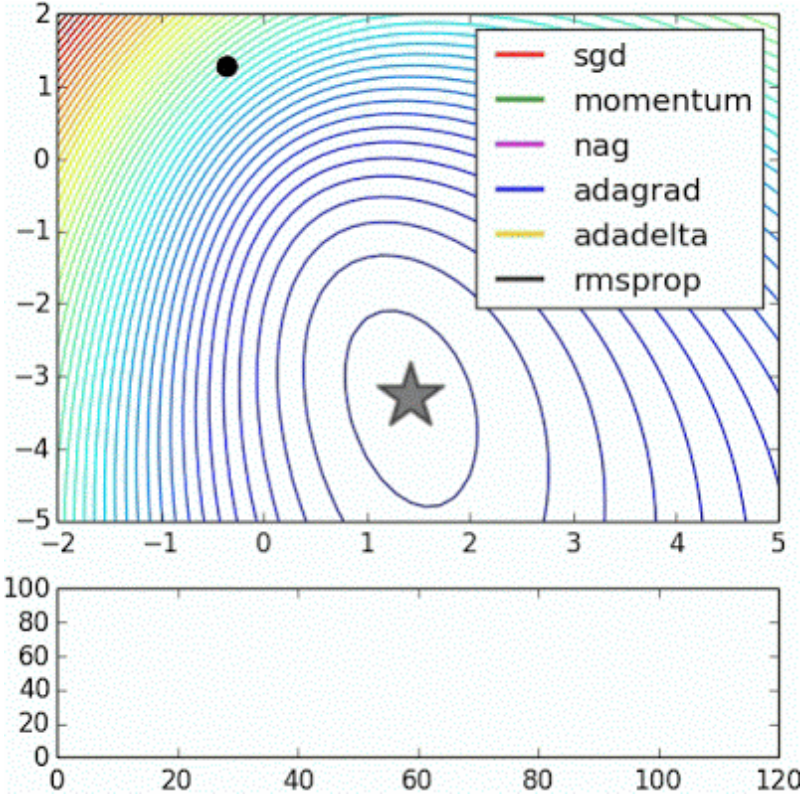
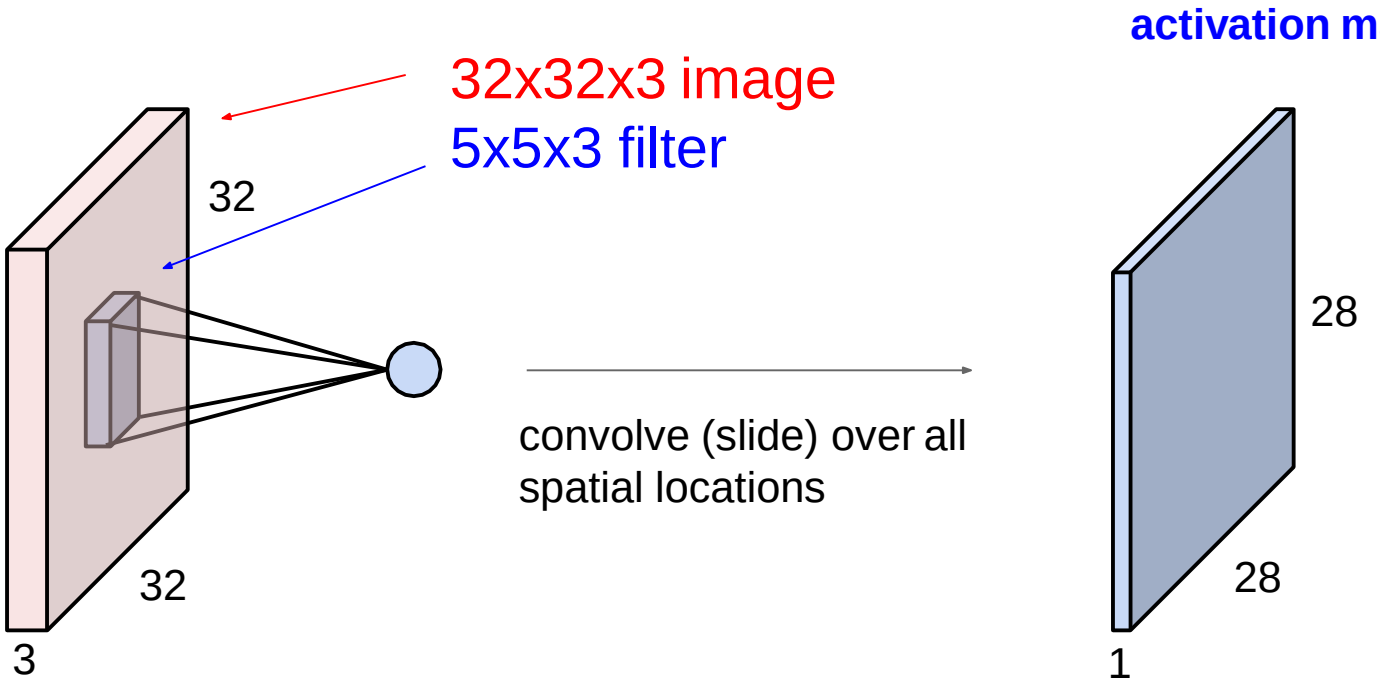


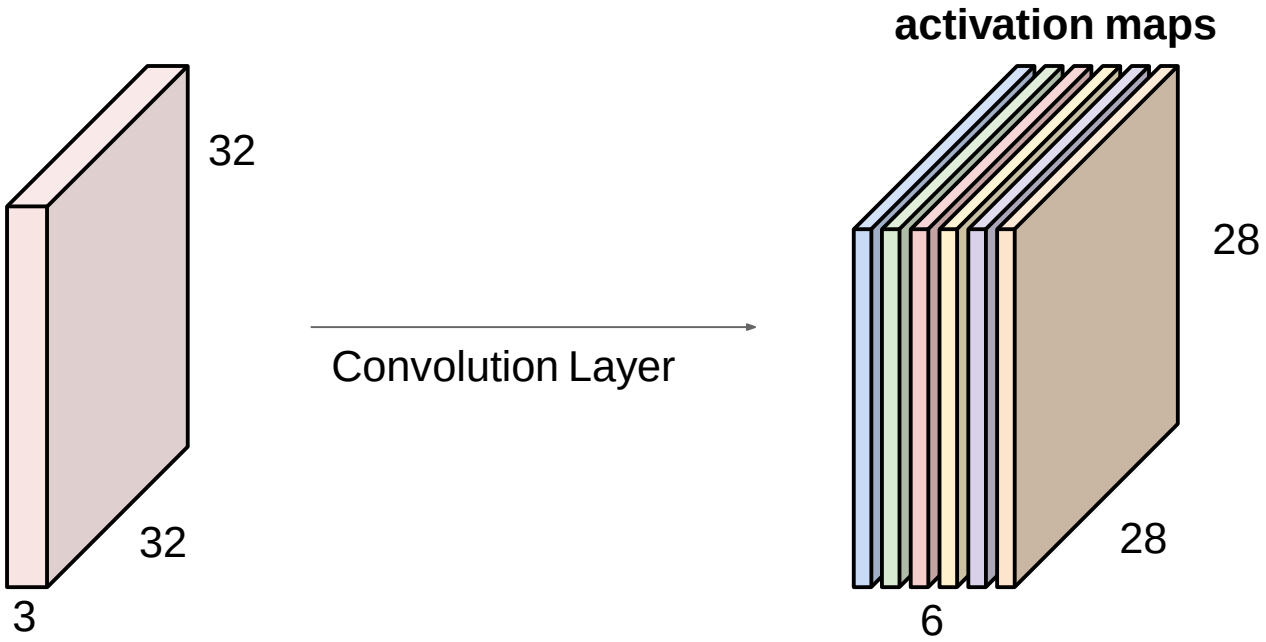
Image credits: Alec Radford

Convolutional Neural Networks

Convolution Layer

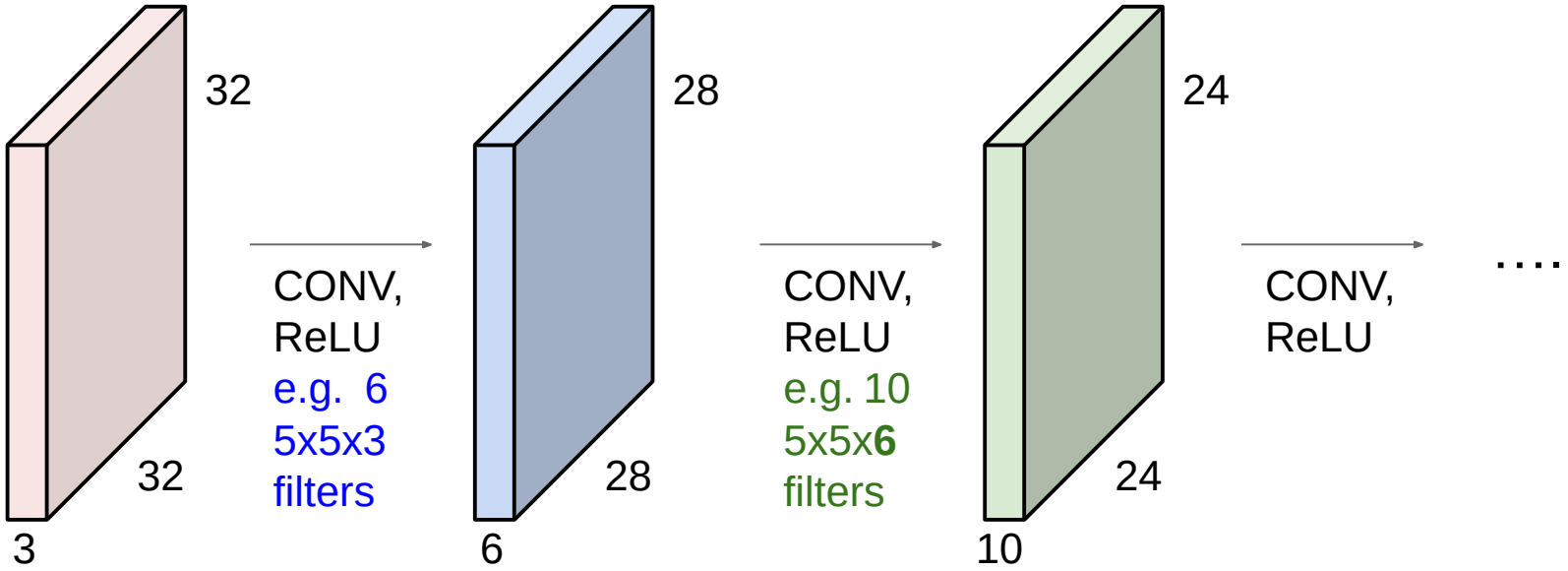


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



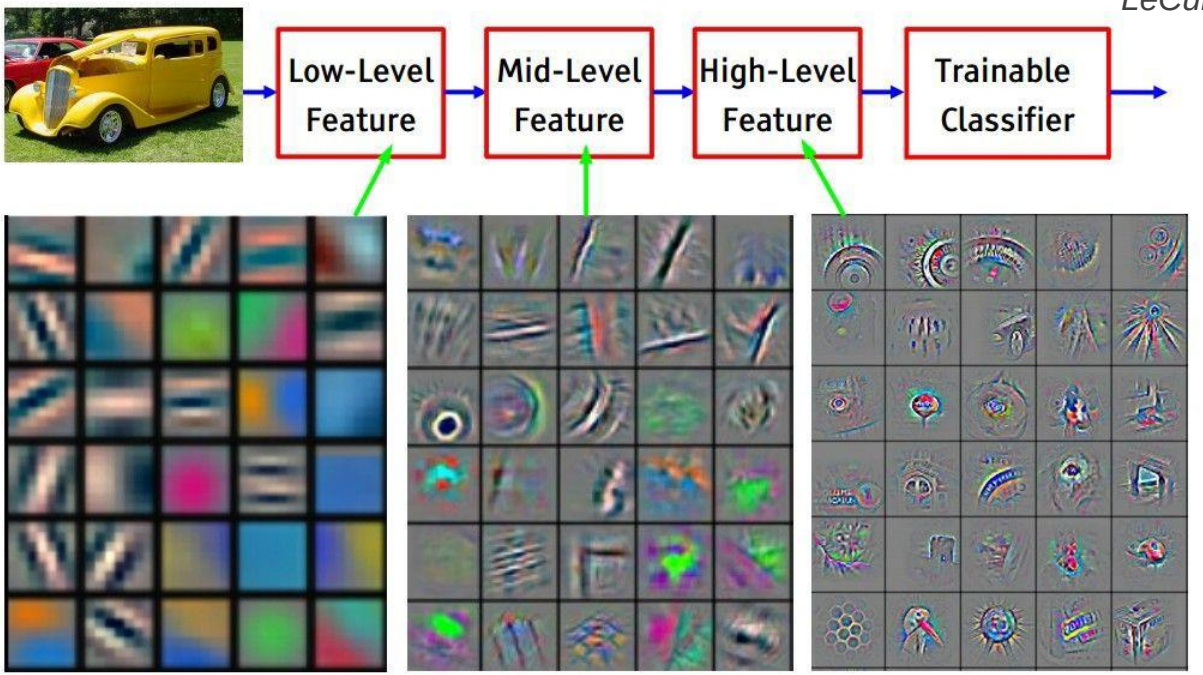
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



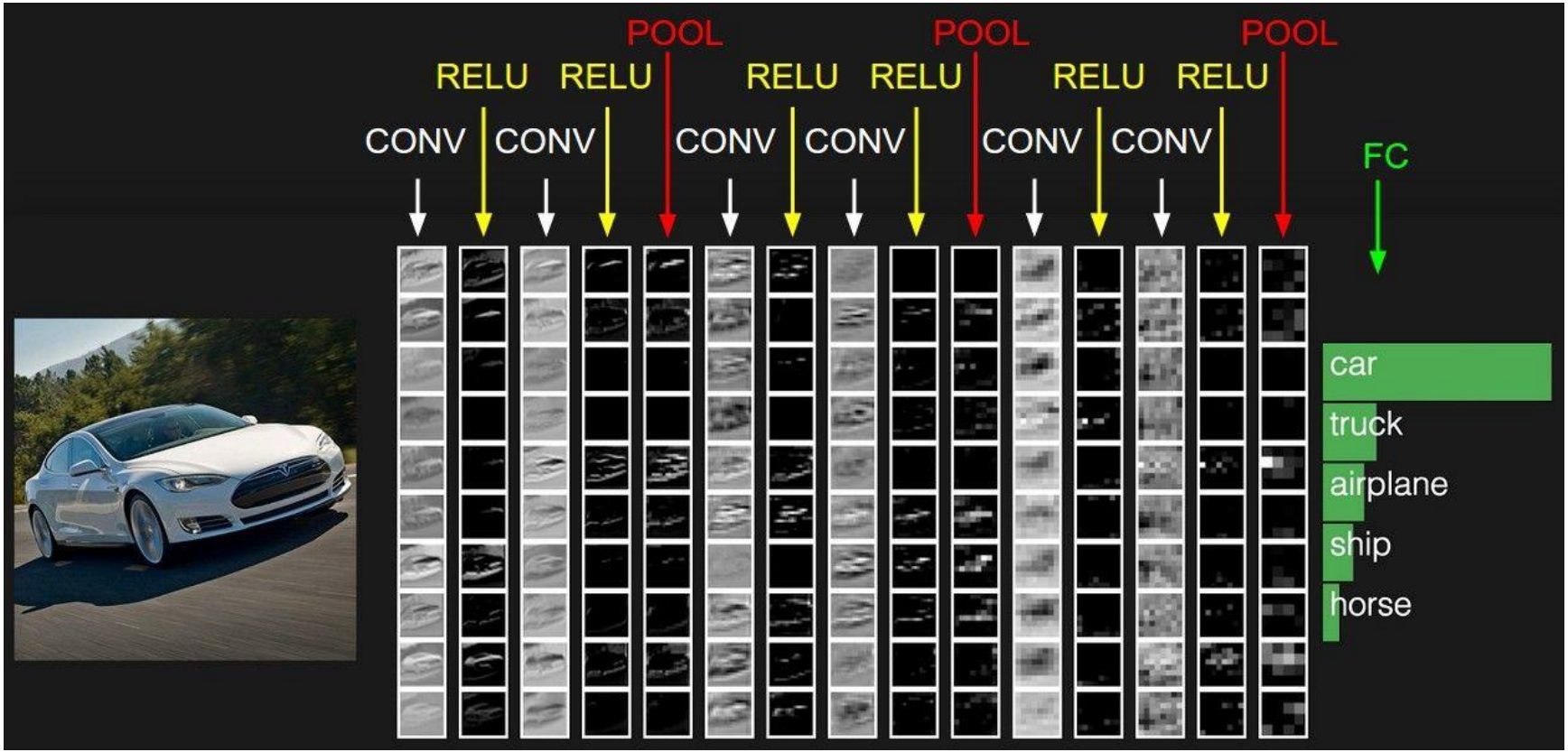
Preview:

[From recent Yann LeCun slides]



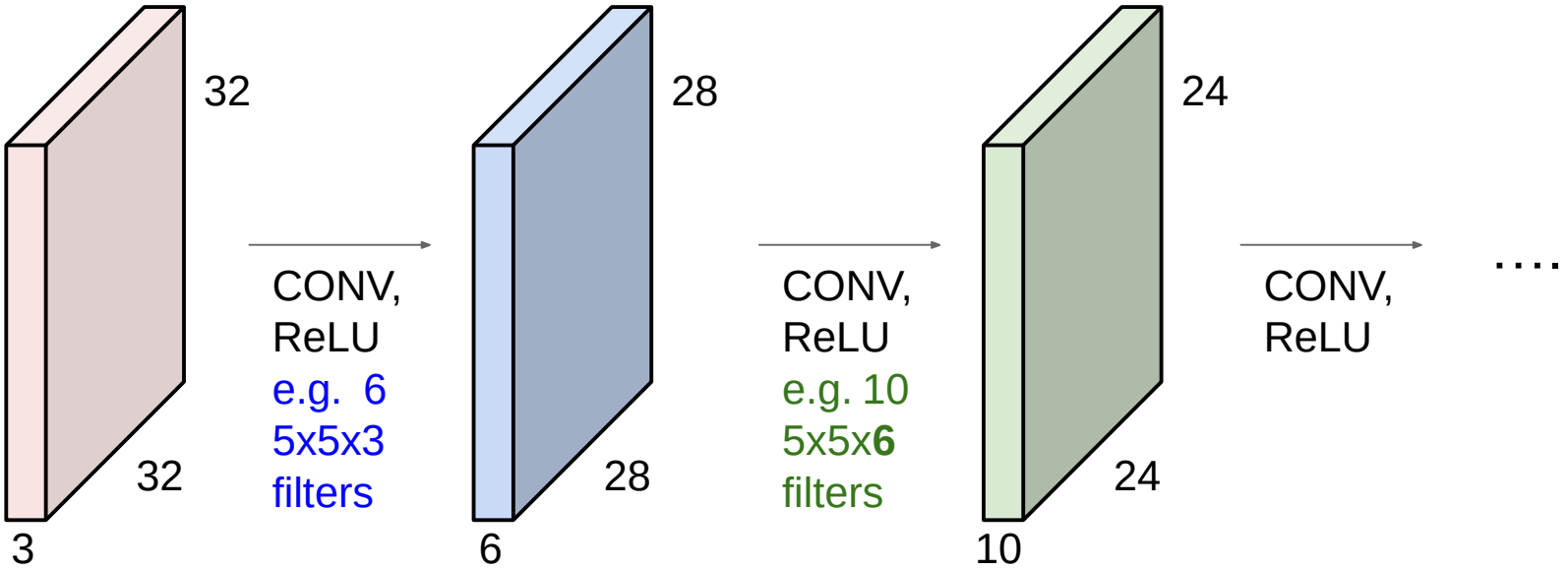
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Preview:

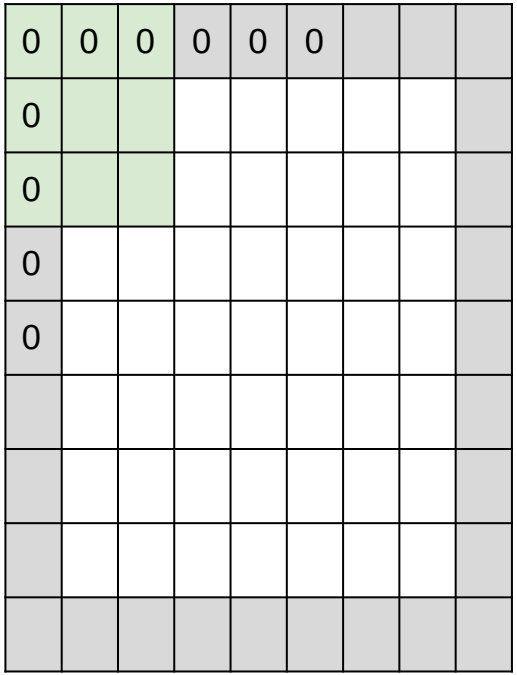

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 → 28 → 24 ...). Shrinking too fast is not good, doesn't work well.

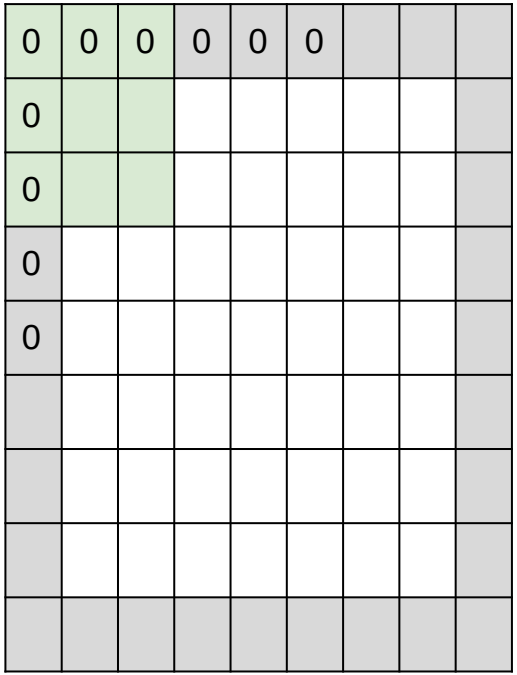


In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => **7x7 output!**

In practice: Common to zero pad the border

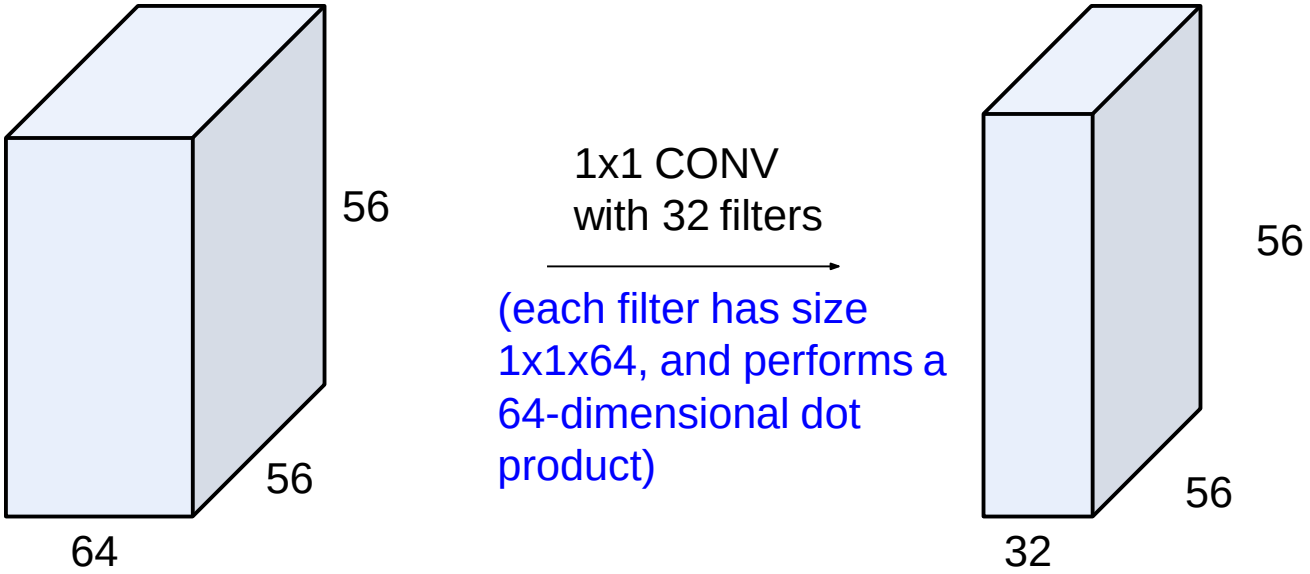


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => **7x7 output!**

In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

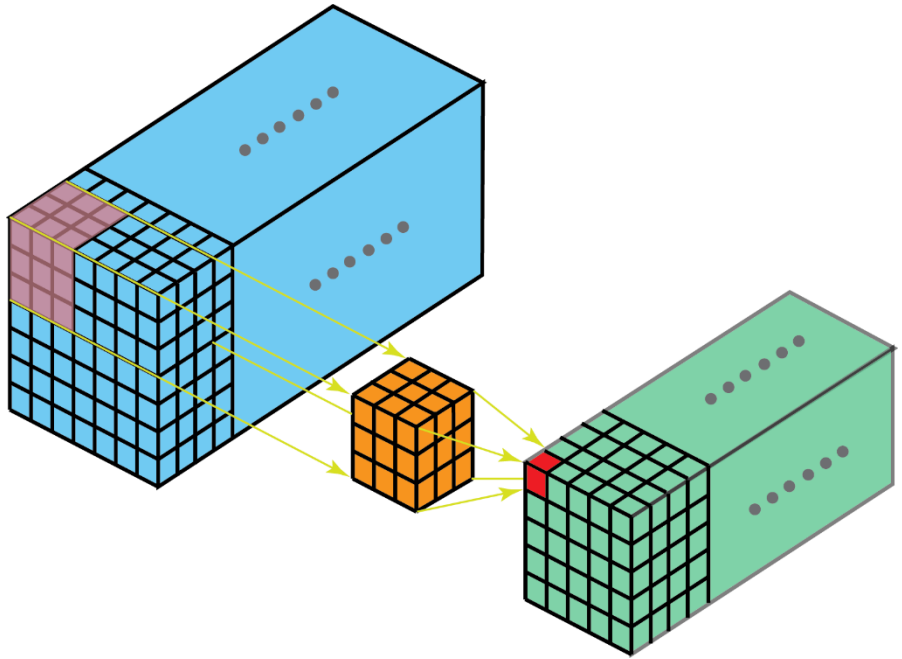
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

1x1 convolution layers

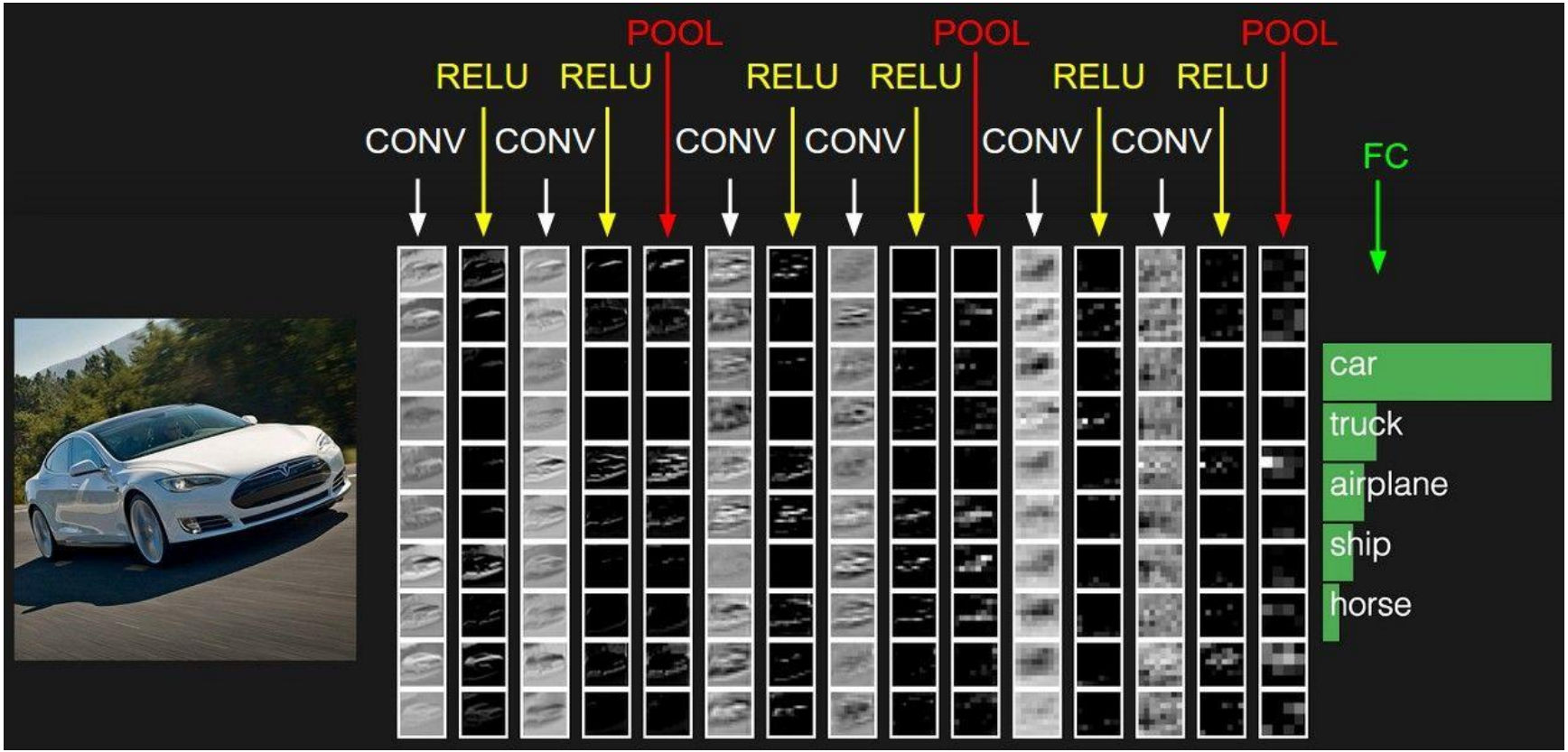


3D Convolution

- A 3D filter can move in all 3-direction (height, width, channel of the image).
- At each position, the element-wise multiplication and addition provide one number.
- Since the filter slides through a 3D space, the output is 3D too.

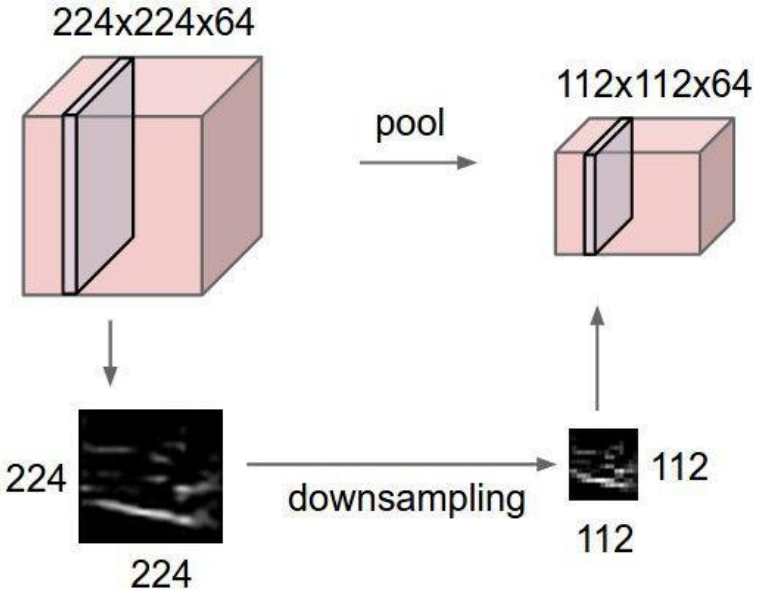


two more layers to go: POOL/FC

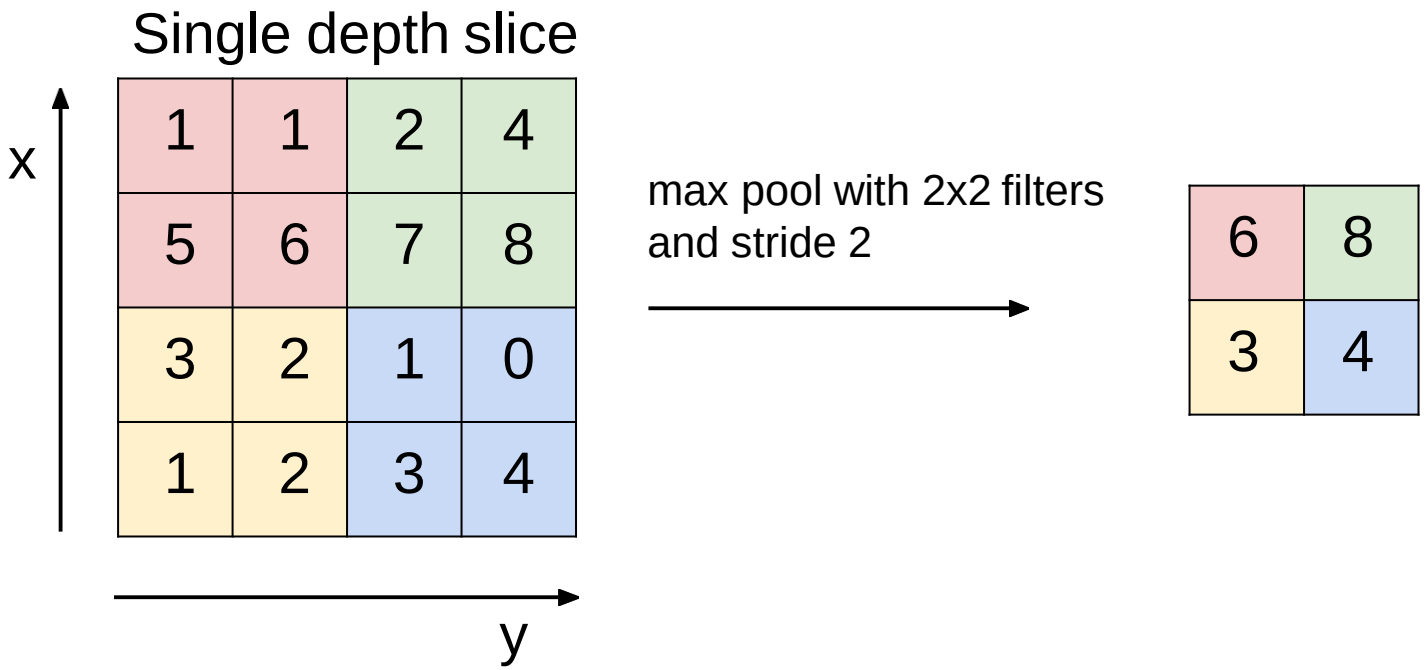


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

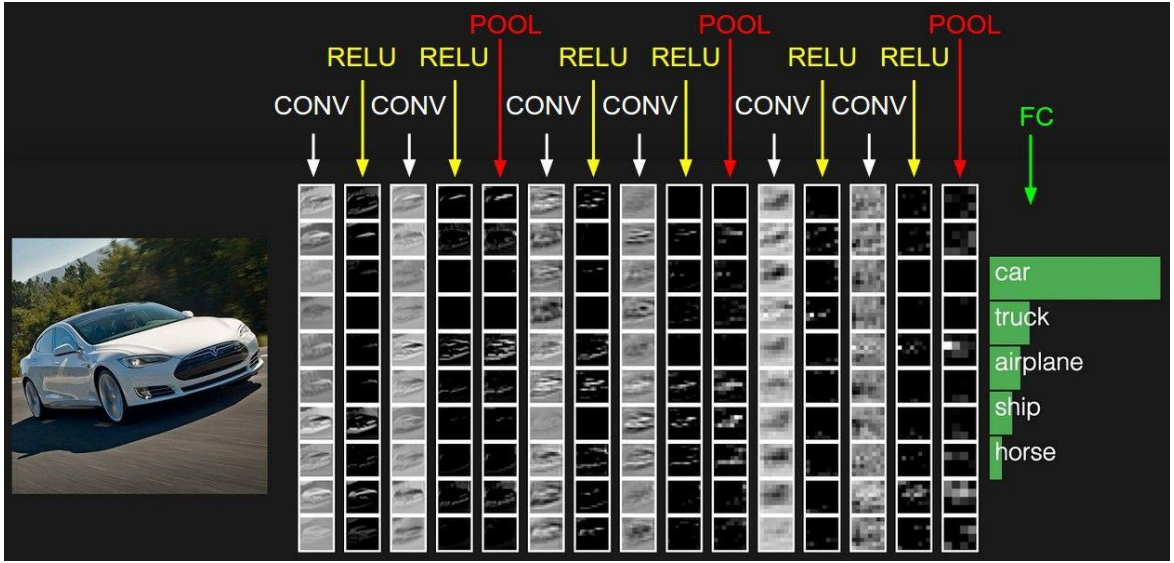


MAX Pooling

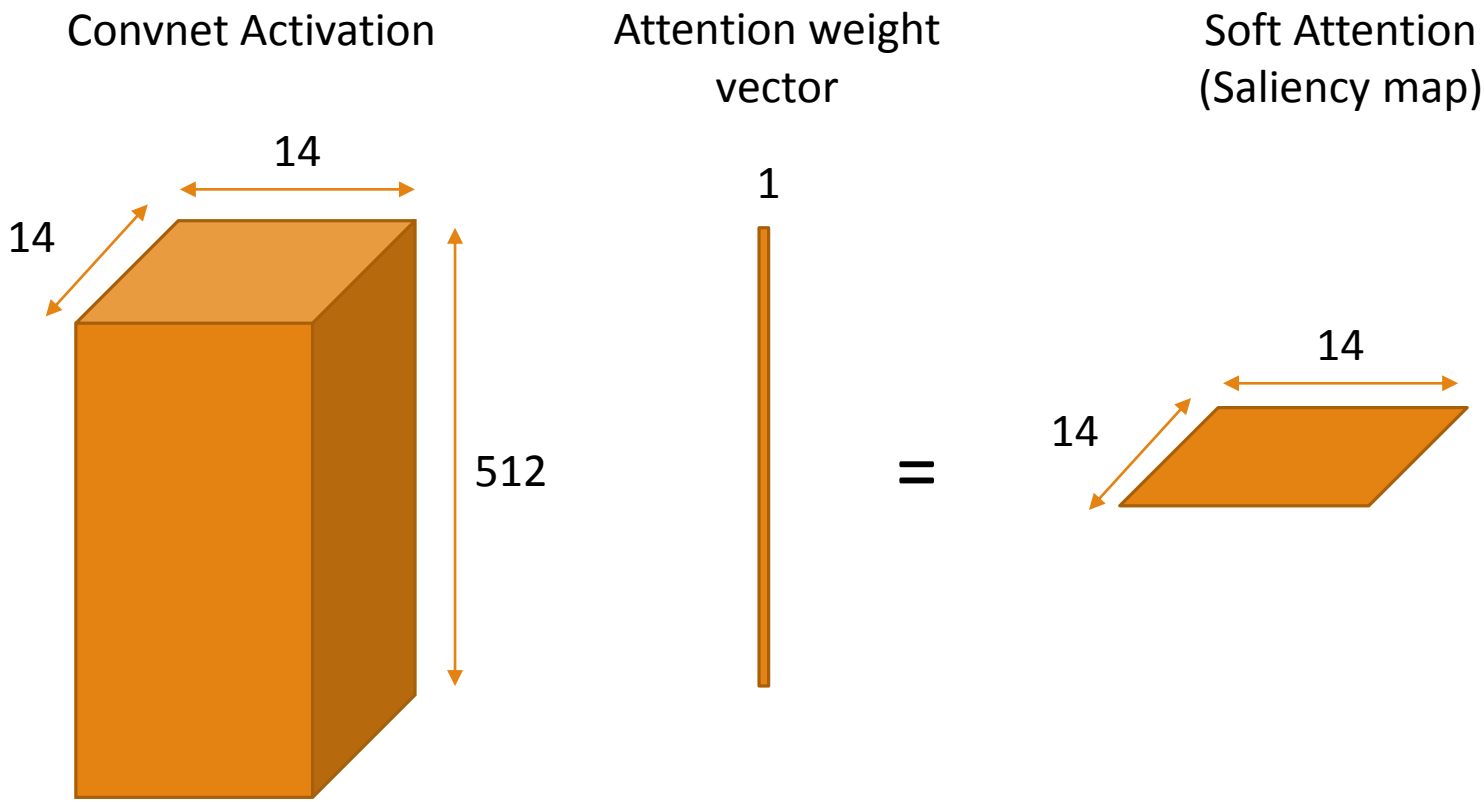

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Fully Connected Layer (FC layer)

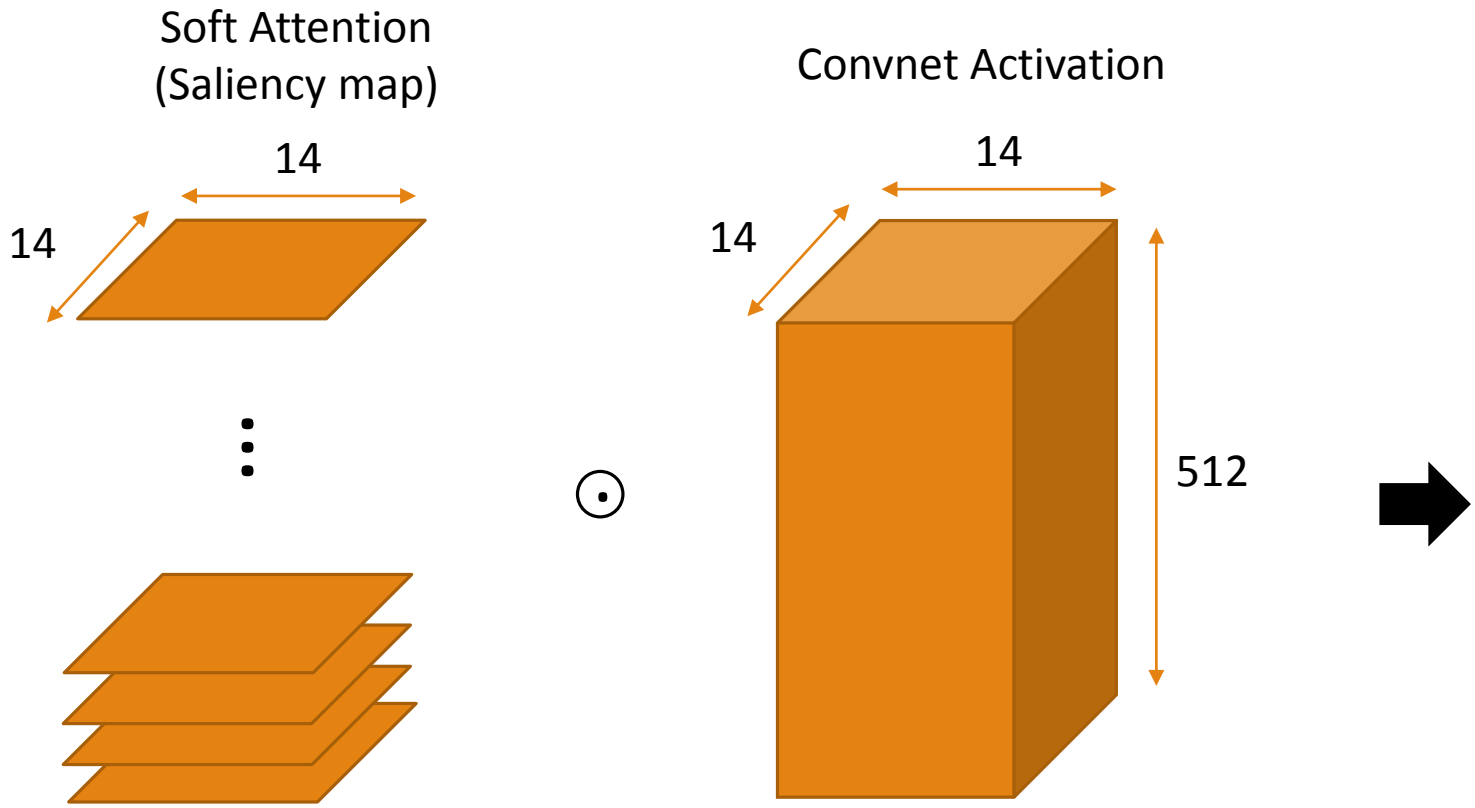
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



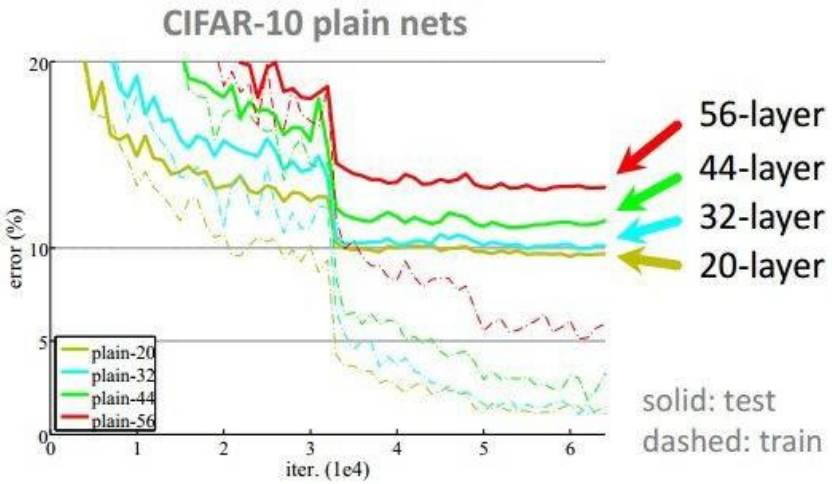
Soft attention

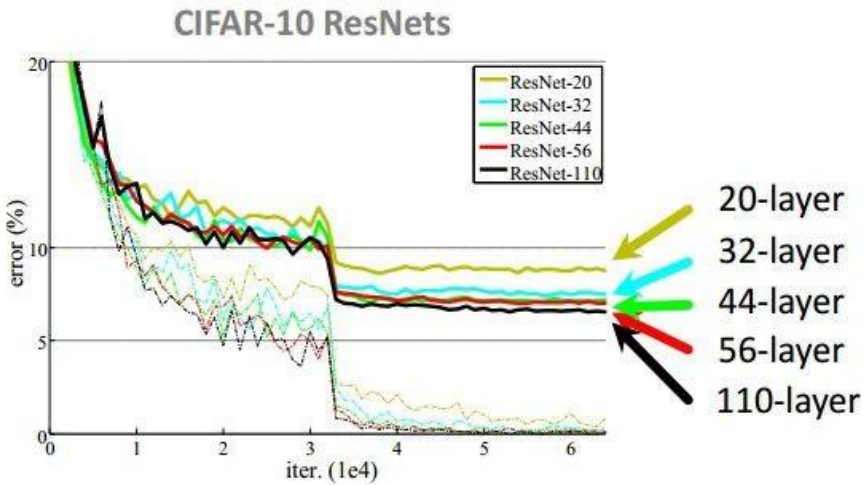


Soft attention



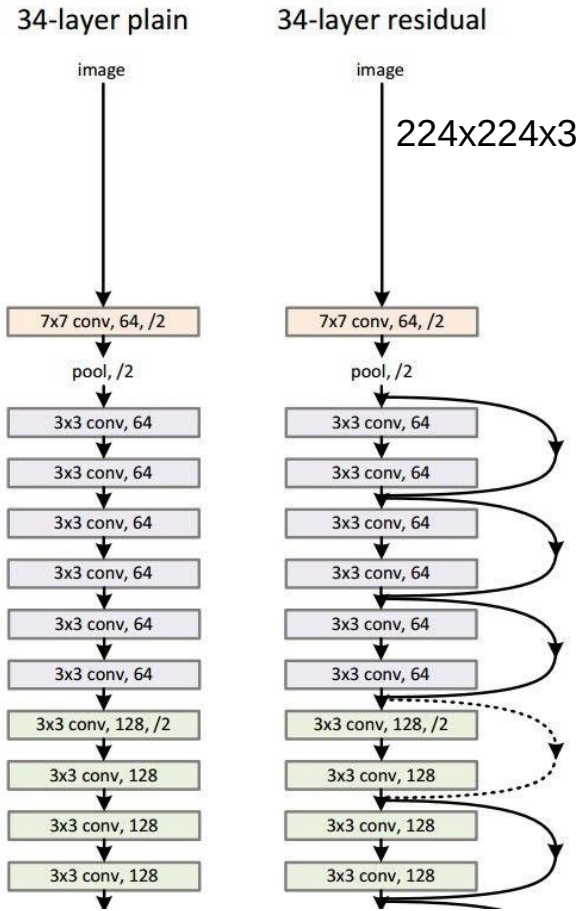
Residual Networks





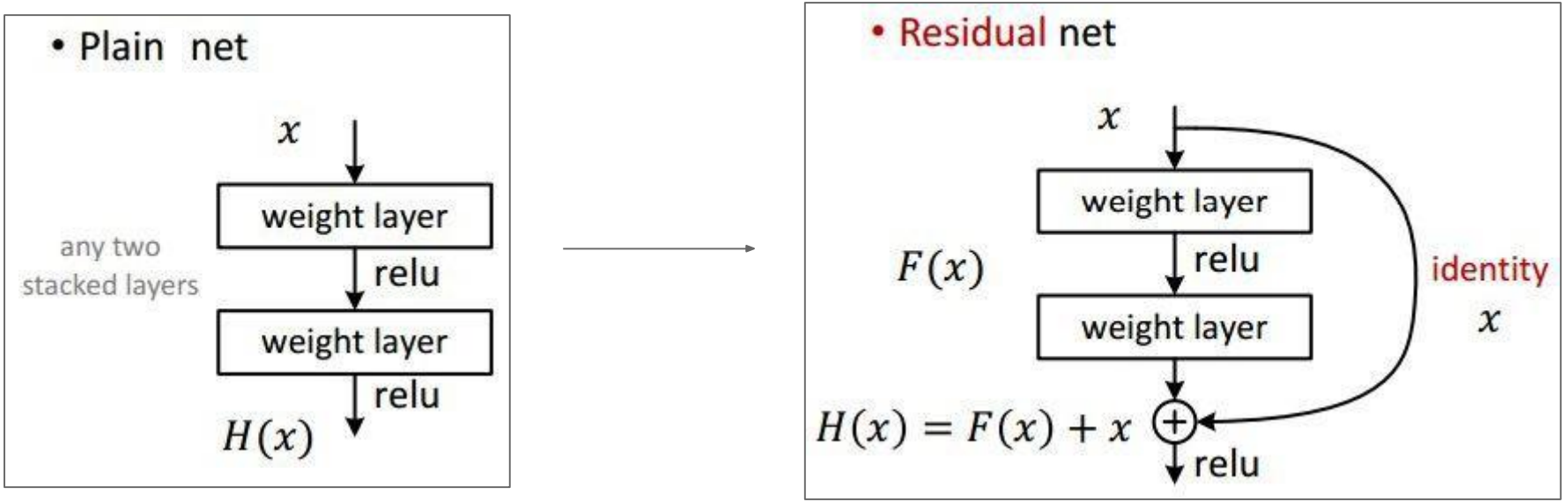
[He at al., 2015]

Residual Networks



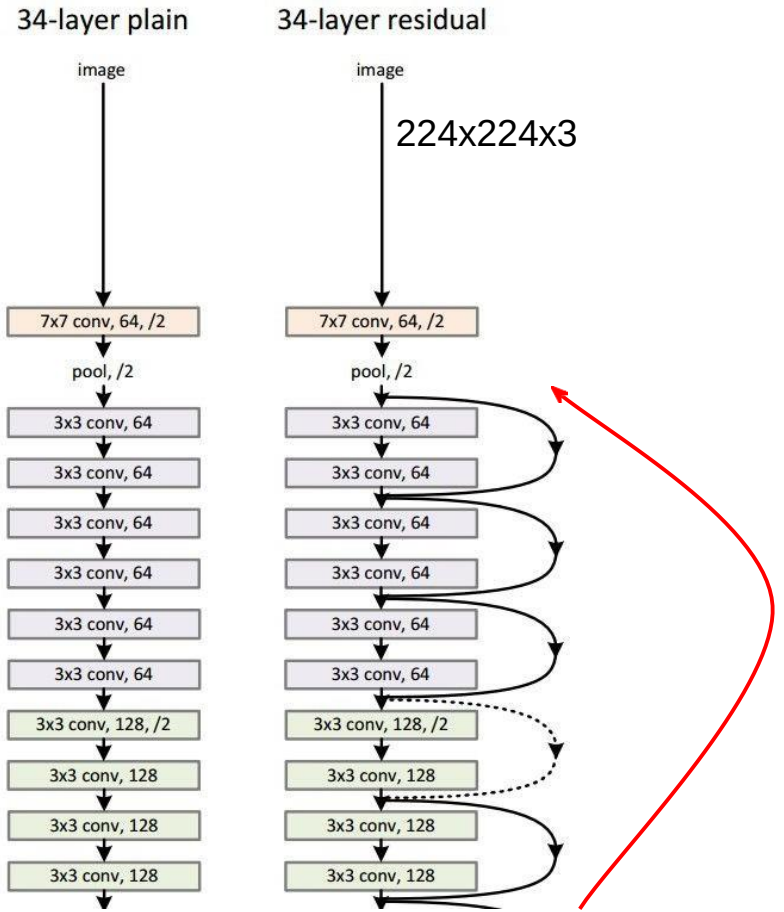
[He at al., 2015]

Residual Networks



[He at al., 2015]

Residual Networks



During back-prop, gradient is flows through layers without vanishing

[He at al., 2015]

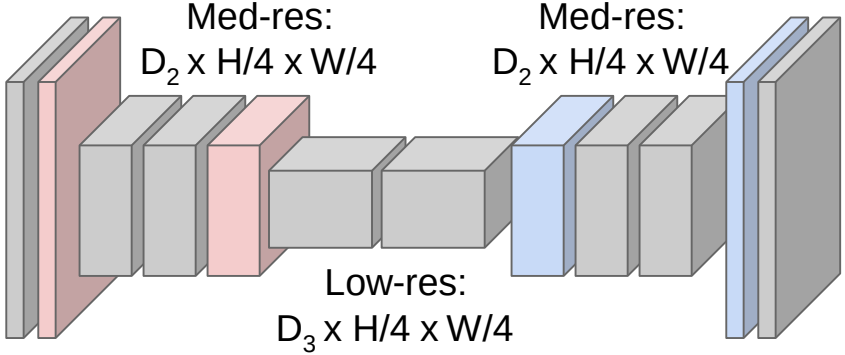
Image to Image Transformation: Fully Convolutional

Downsampling:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
???

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

In-Network upsampling: “Max Unpooling”

Max Pooling
Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

5	6
7	8

Output: 2 x 2

Rest of the network

Max Unpooling
Use positions from pooling layer

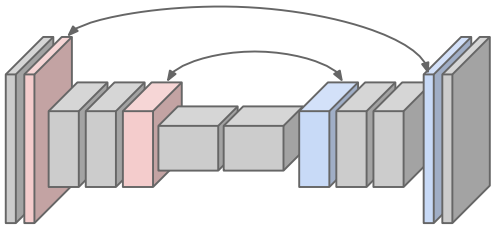
1	2
3	4

Input: 2 x 2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

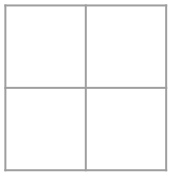
Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

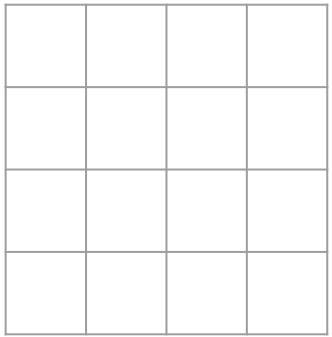


Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



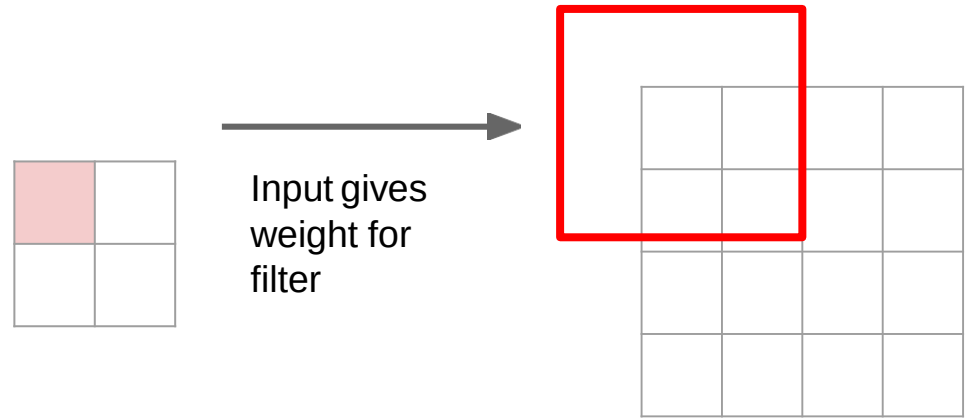
Input: 2 x 2



Output: 4 x 4

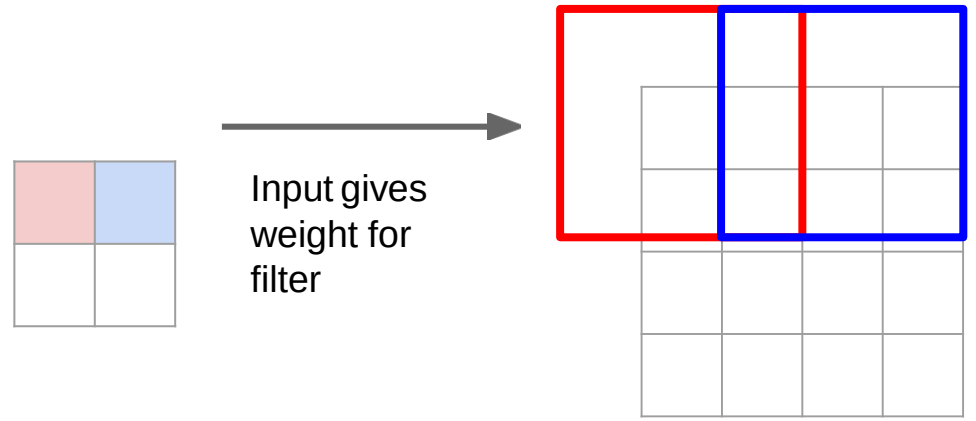
Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



Learnable Upsampling: Transpose Convolution

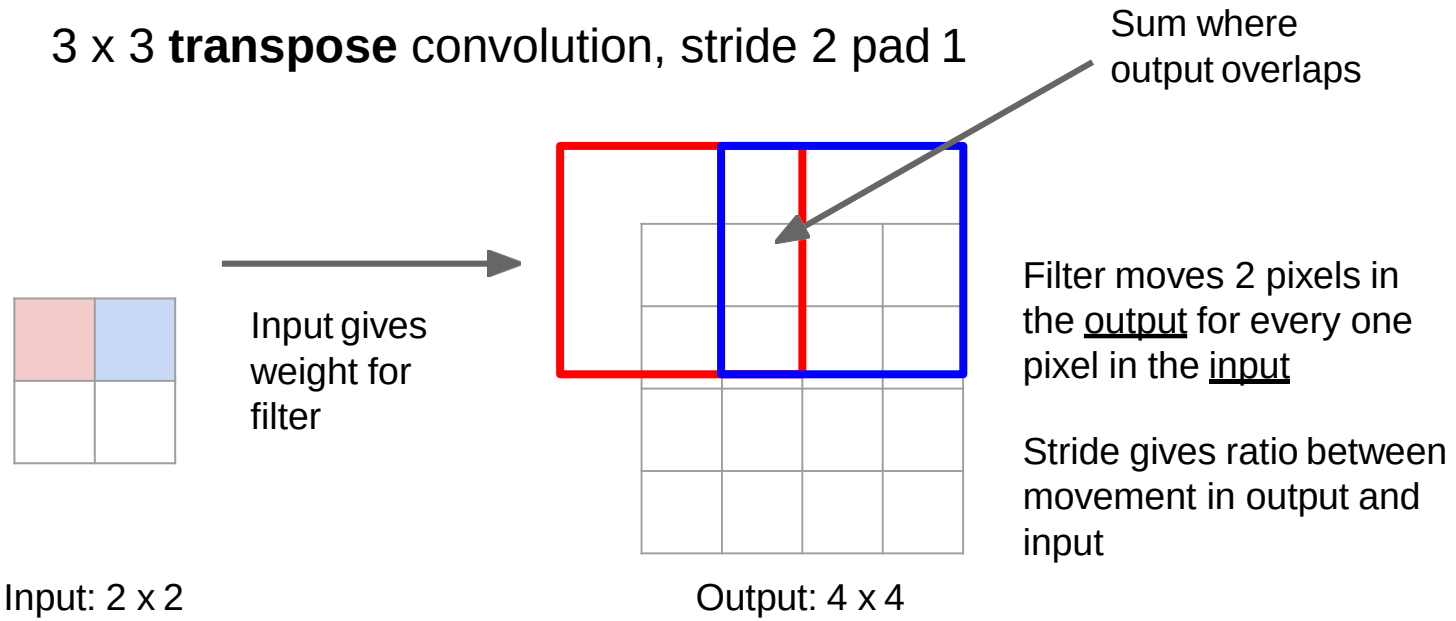
3 x 3 **transpose** convolution, stride 2 pad 1



Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

Learnable Upsampling: Transpose Convolution

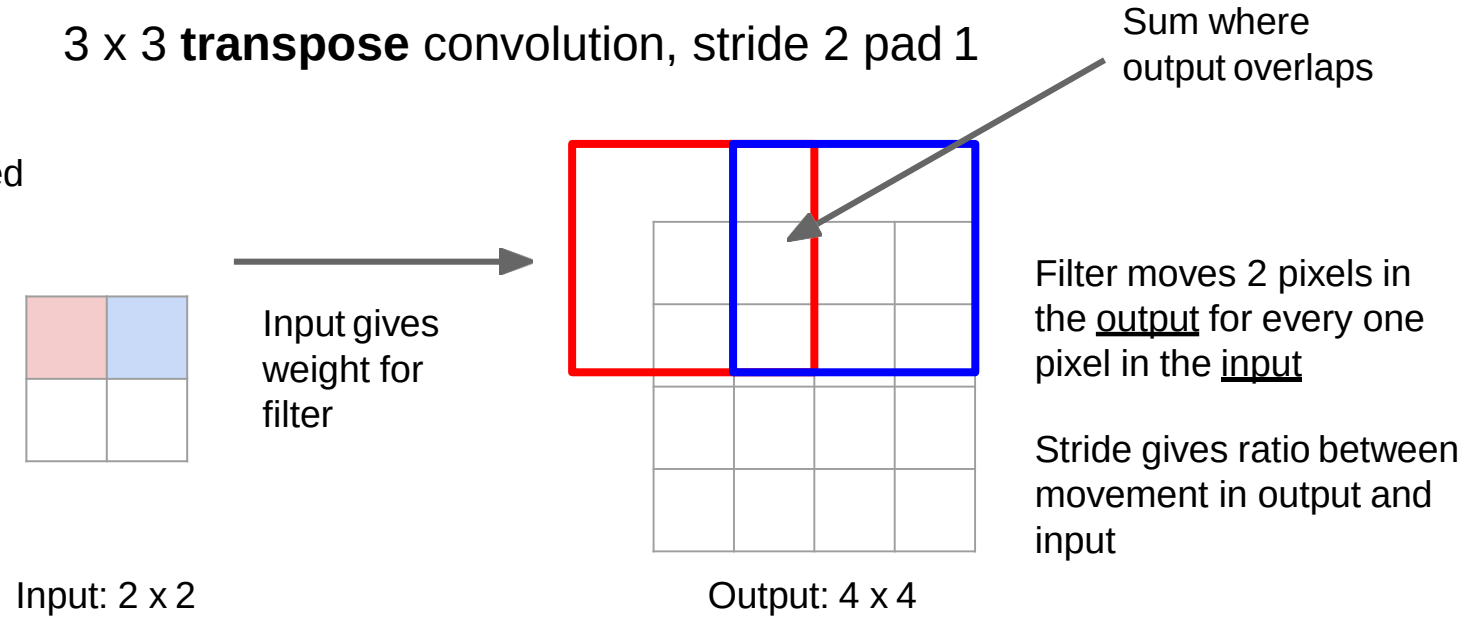


Learnable Upsampling: Transpose Convolution

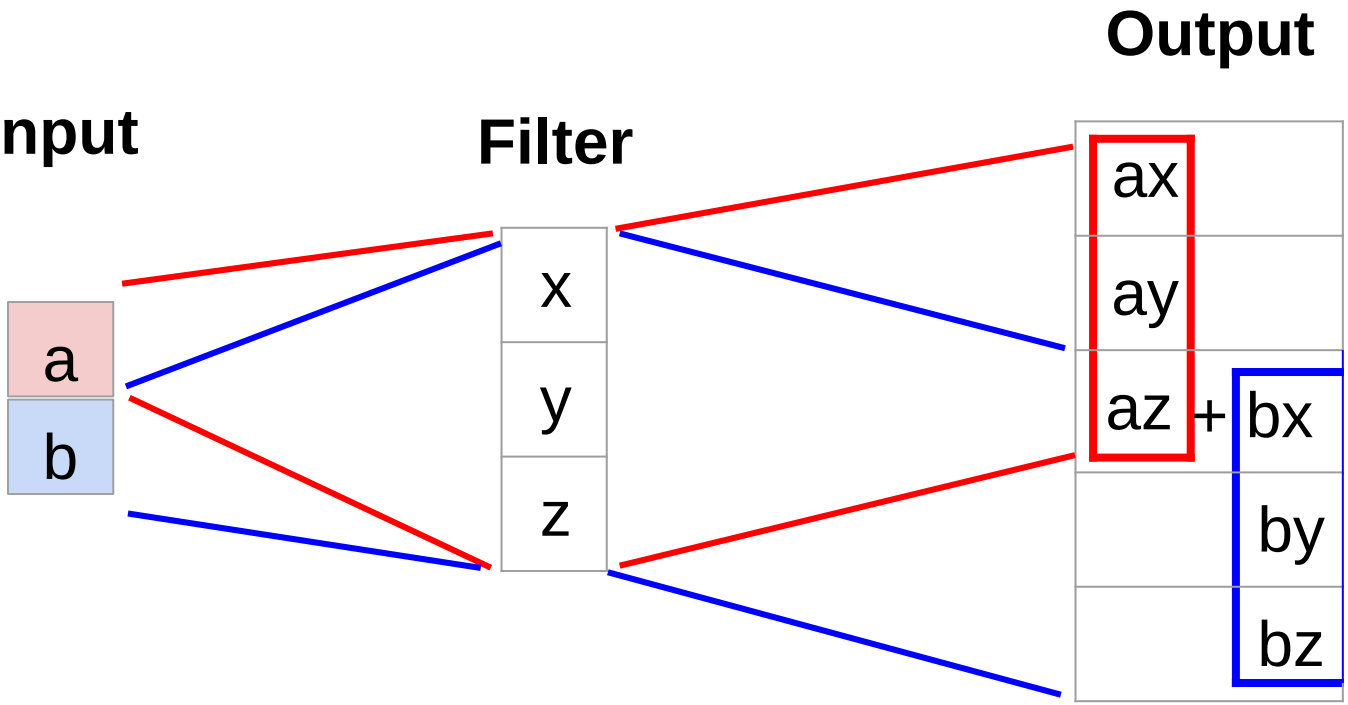
Other names:

- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

3 x 3 **transpose** convolution, stride 2 pad 1



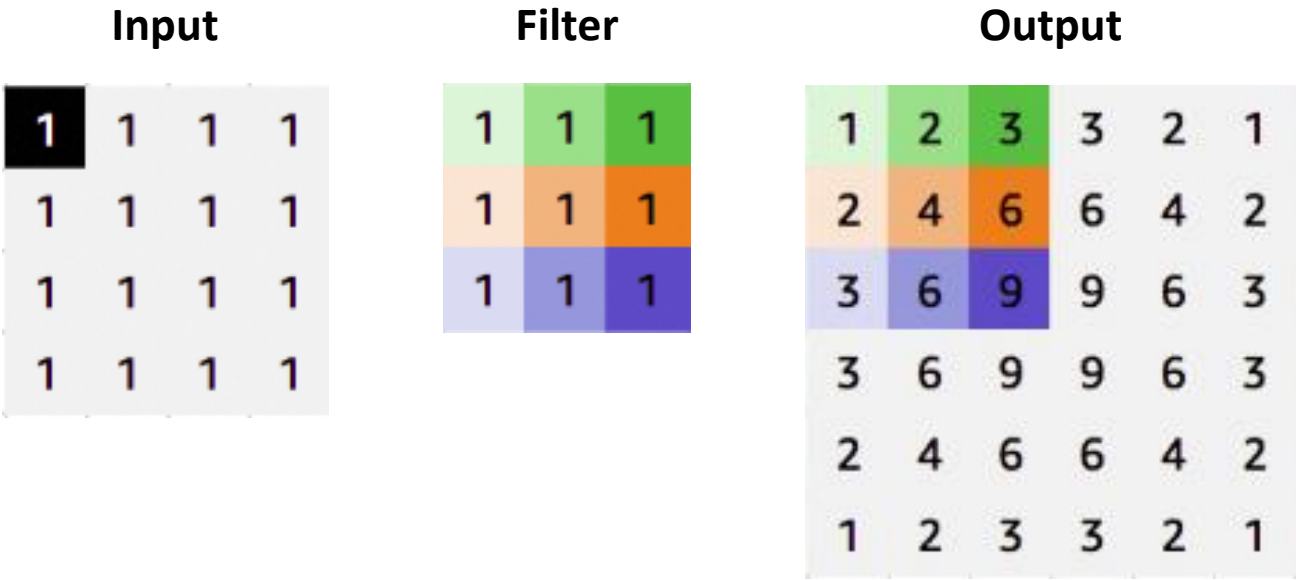
Learnable Upsampling: 1D Example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

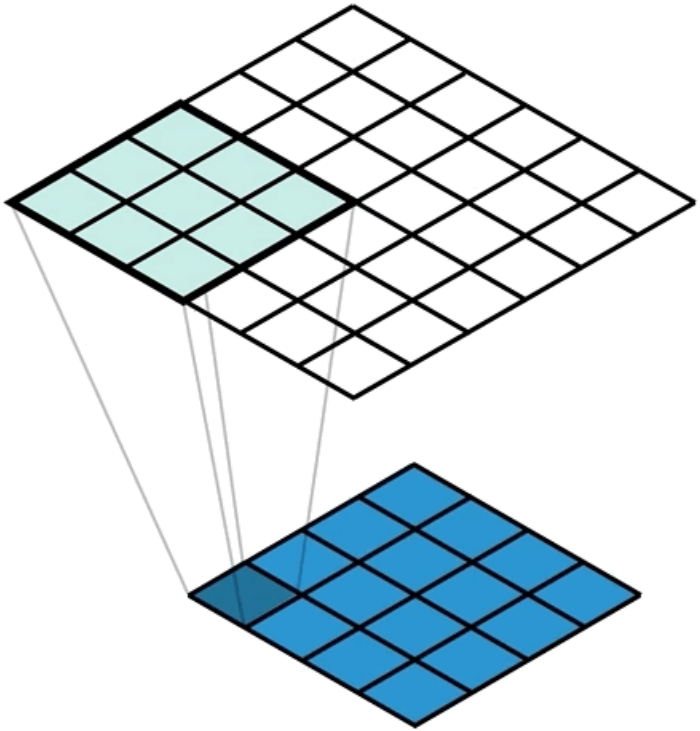
Transpose Convolution: 2D Example



- Response accumulation in the edges of the output are less than the center pixels
- Often, this is not a big problem since filter weights learn how to adjust for such issues

https://medium.com

Transpose Convolution: 2D Example



A **Transpose Convolution** with 3x3 kernel (stride 1) applied to a 4x4 input to give a 6x6 output.

https://medium.com

Recurrent Neural Networks

RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

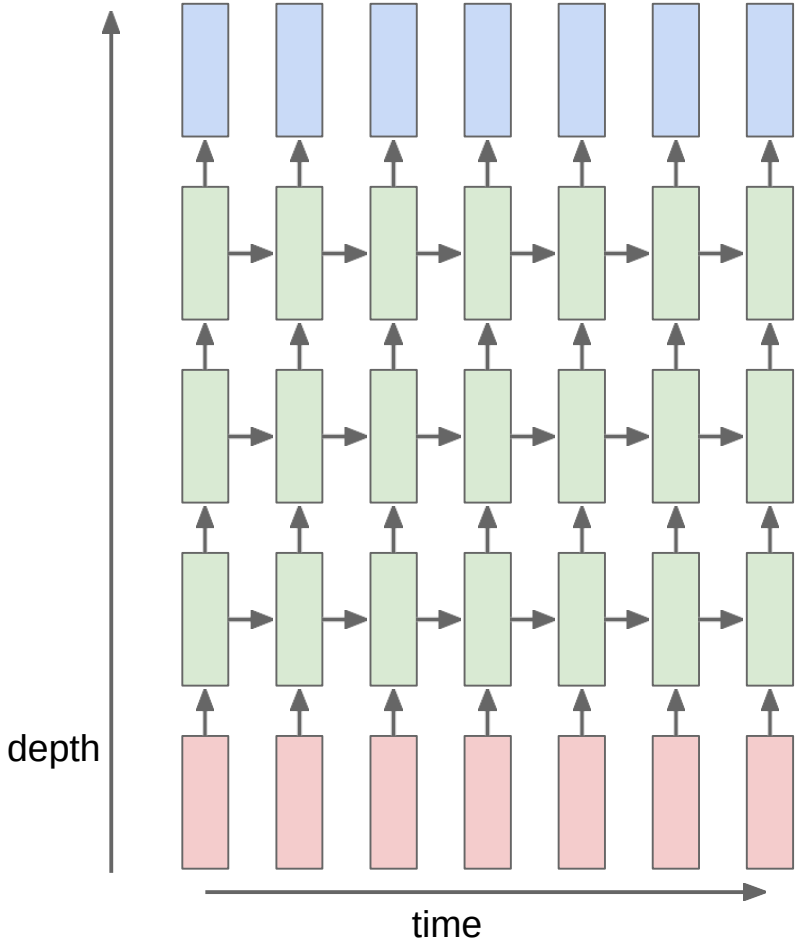
$h \in \mathbb{R}^n$ $\quad$ $W^l \ \ [n \times 2n]$

A generalization of RNN. At l=1:

- $h_t^{l-1} = x_t$
- $W^l = [W_{xh}\ W_{hh}]$

It is equivalent to:

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



RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$ $\quad W^l \ [n \times 2n]$

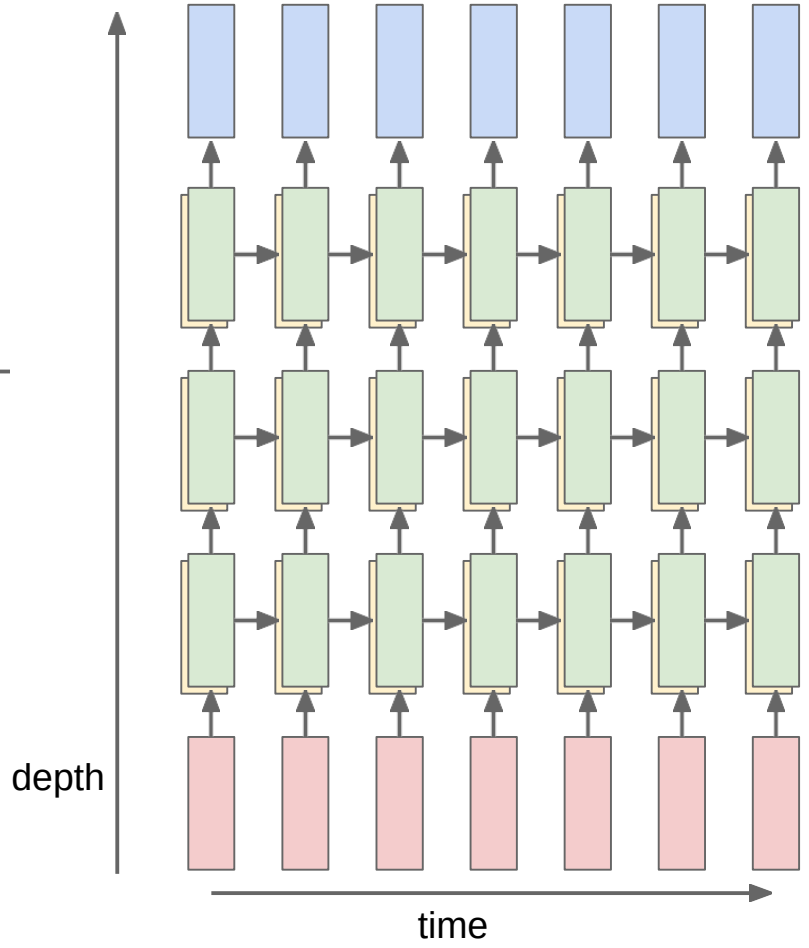
LSTM:

$W^l \ [4n \times 2n]$

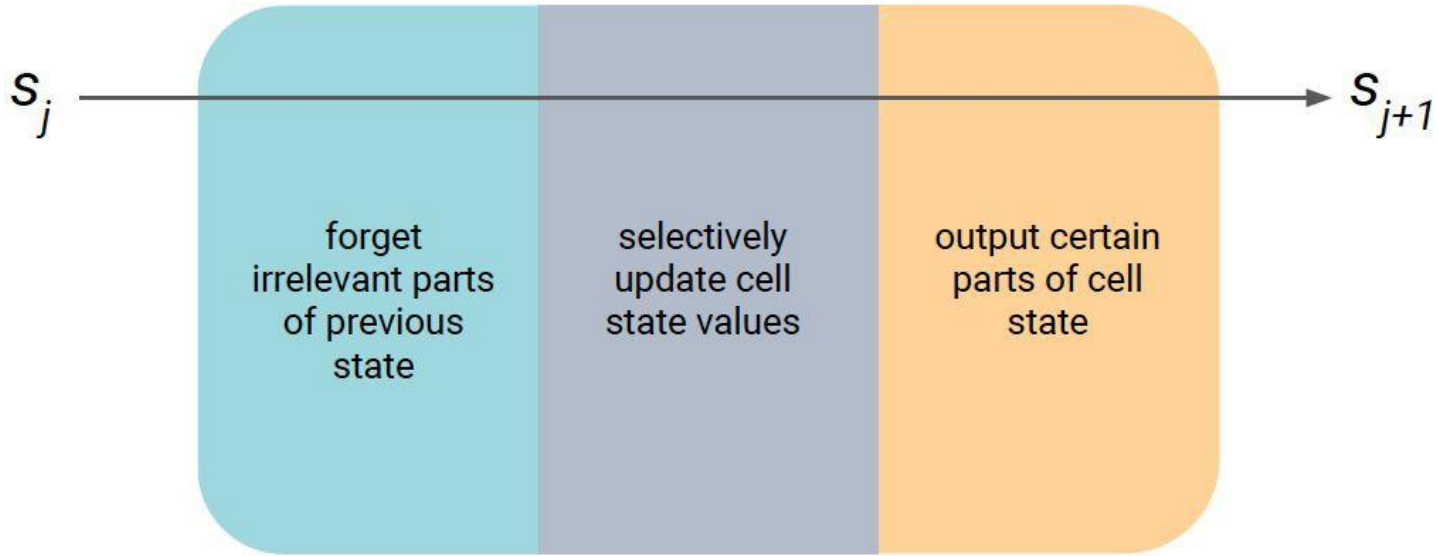
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

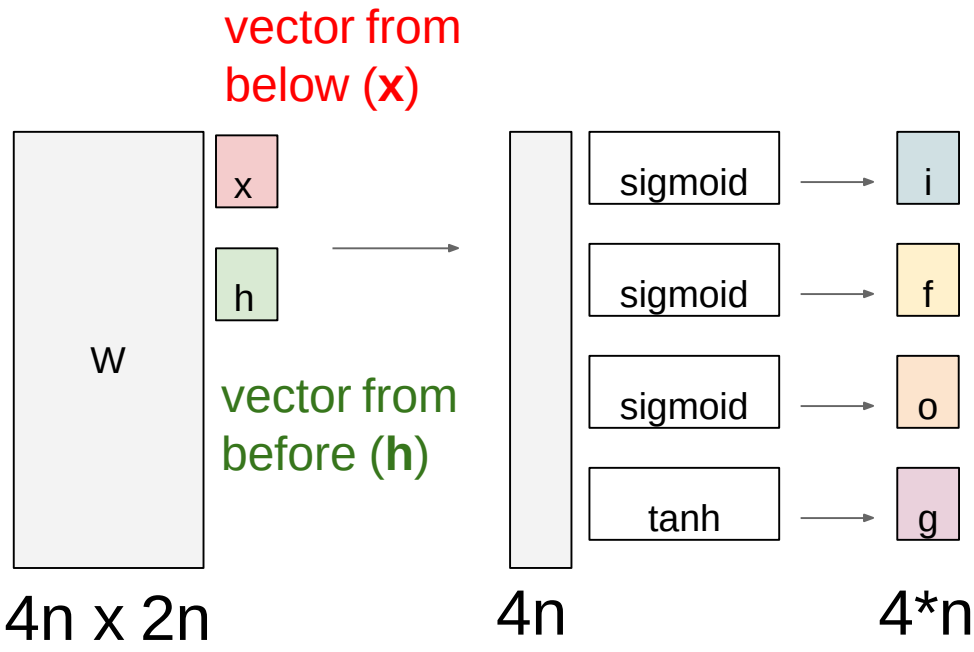


LSTM - main idea



Slide adapted from MIT 6.S191 (IAP 2017), by Harini Suresh

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]



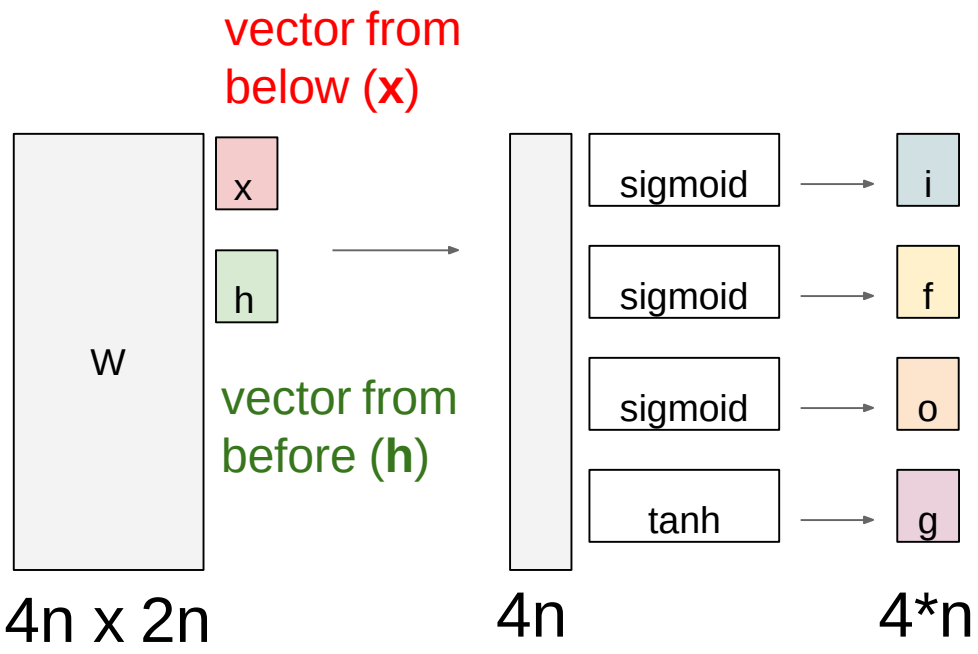
- *c: cell state*
- *h: hidden state (cell output)*
- *i: input gate, weight of acquiring new information*
- *f*: forget gate, weight of remembering old information
- *g: transformed input ([-1,+1])*
- *o: output gate, decides values to be activated based on current memory*

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]



f decides *the degree of preservation for cell state,* by scaling it with a number in [0,1]

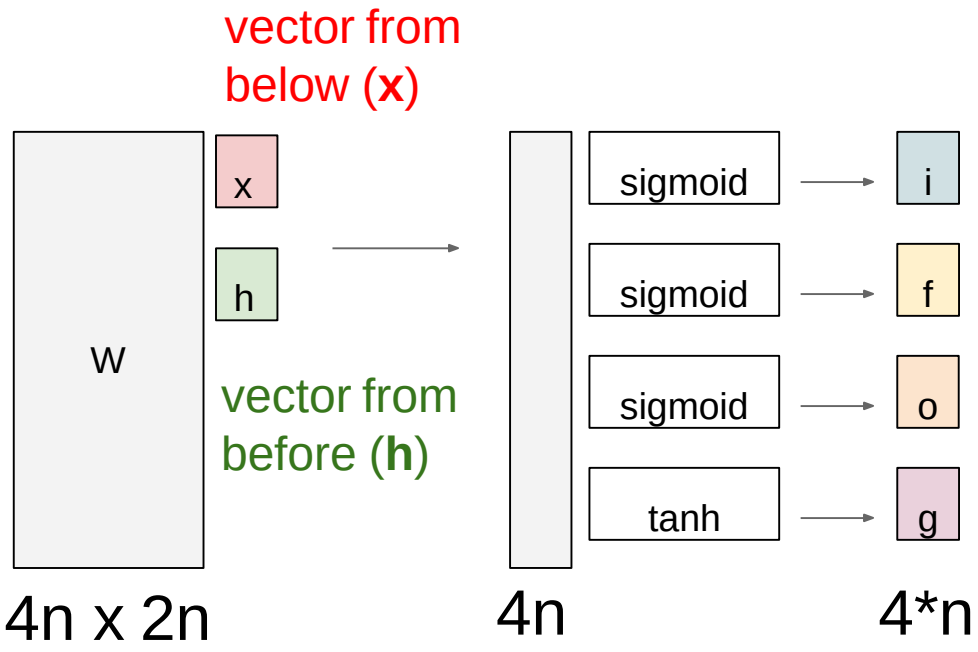
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

g is a transformation of input / hidden state

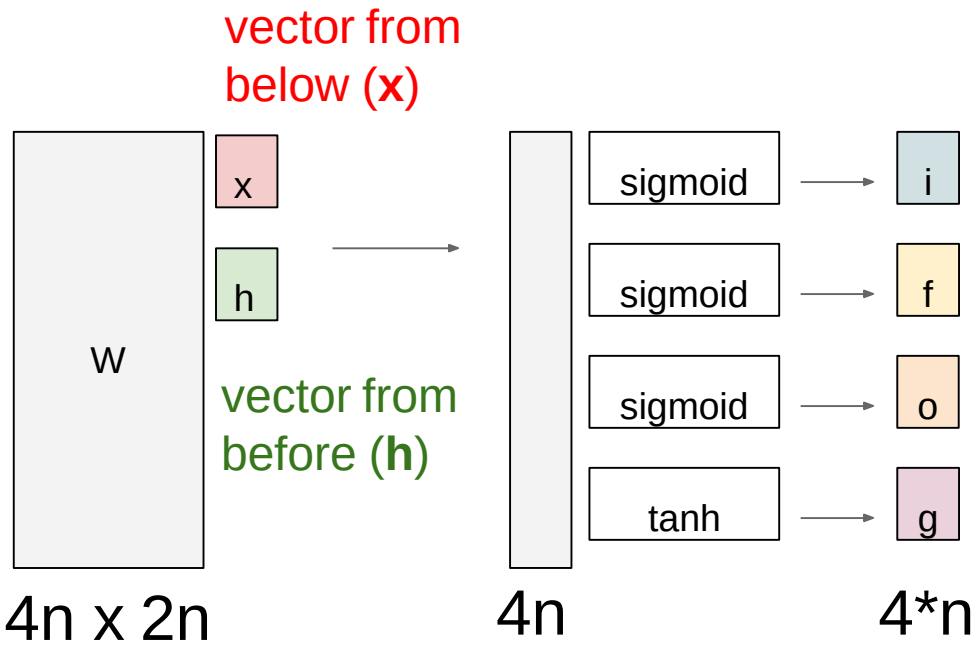


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]



Add *g* into the *cell state*, weighted by *i* *(weight of acquiring new information)*

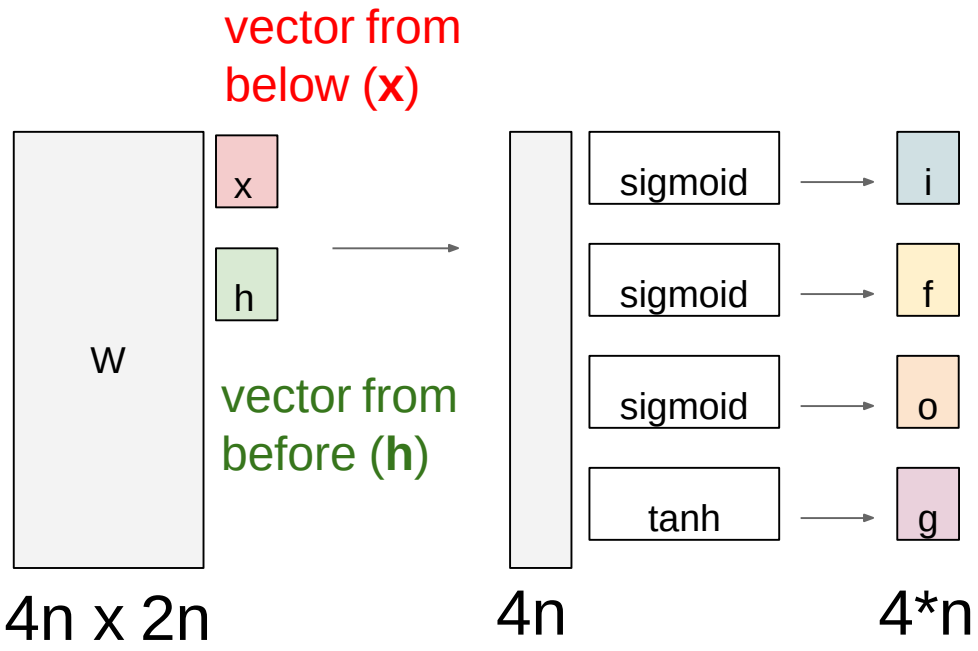
*Alternative interpretation: i*g decouples the "influence of g" and "g itself".*

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]



New hidden state is a scaled version of tanh(cell state).

o: output gate, decides values to be activated based on current memory

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

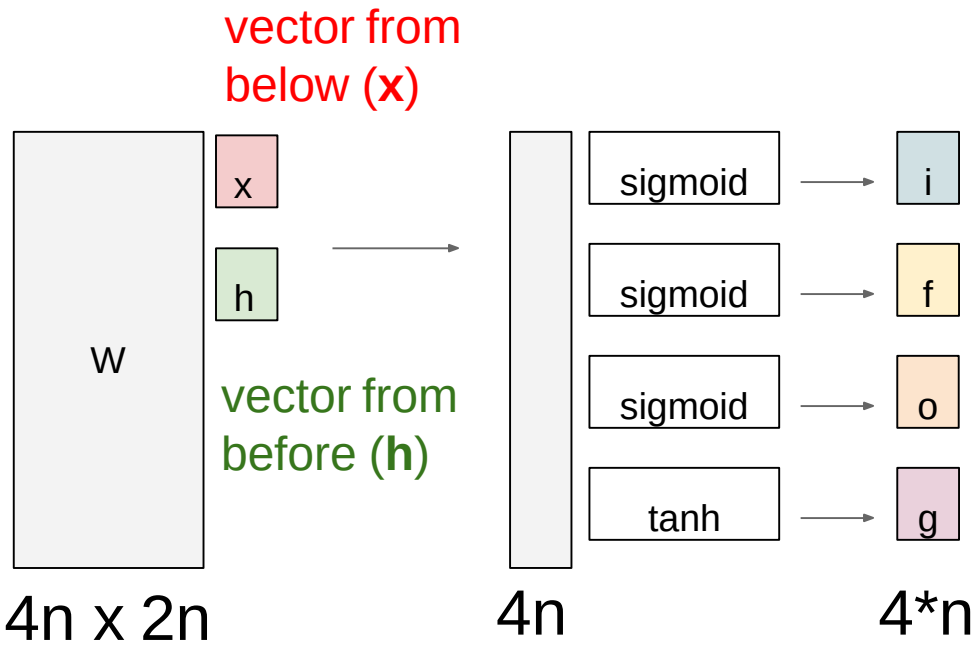
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

Q: Why tanh?

A: Not very crucial, sometimes not used



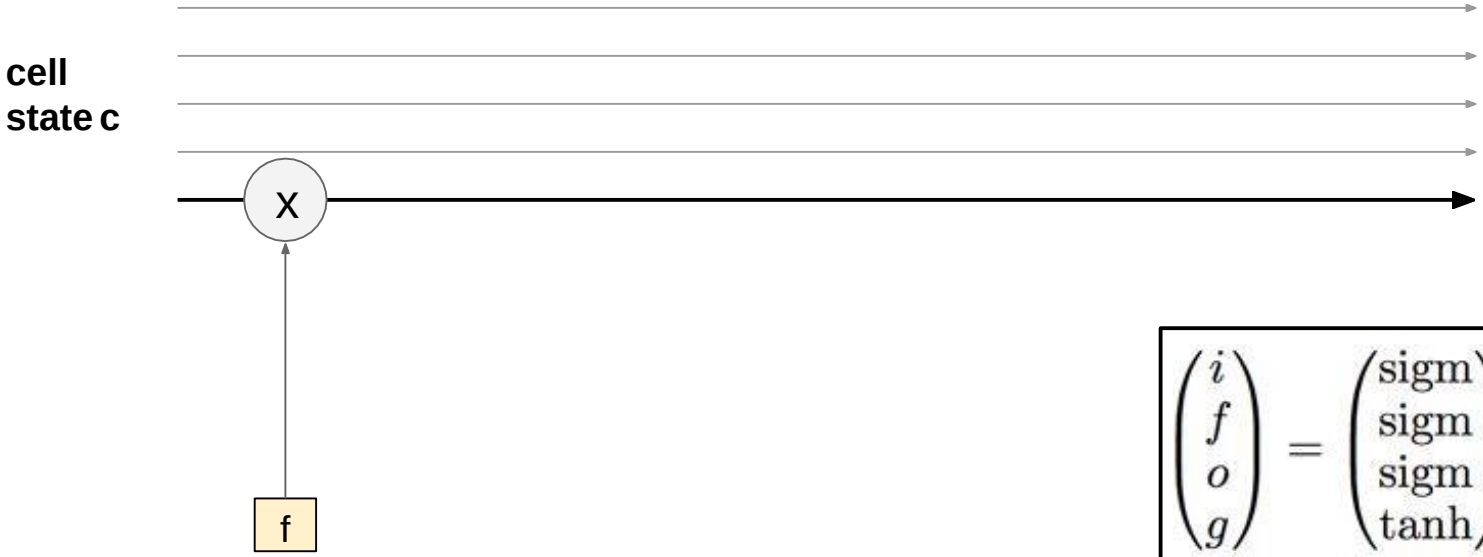
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

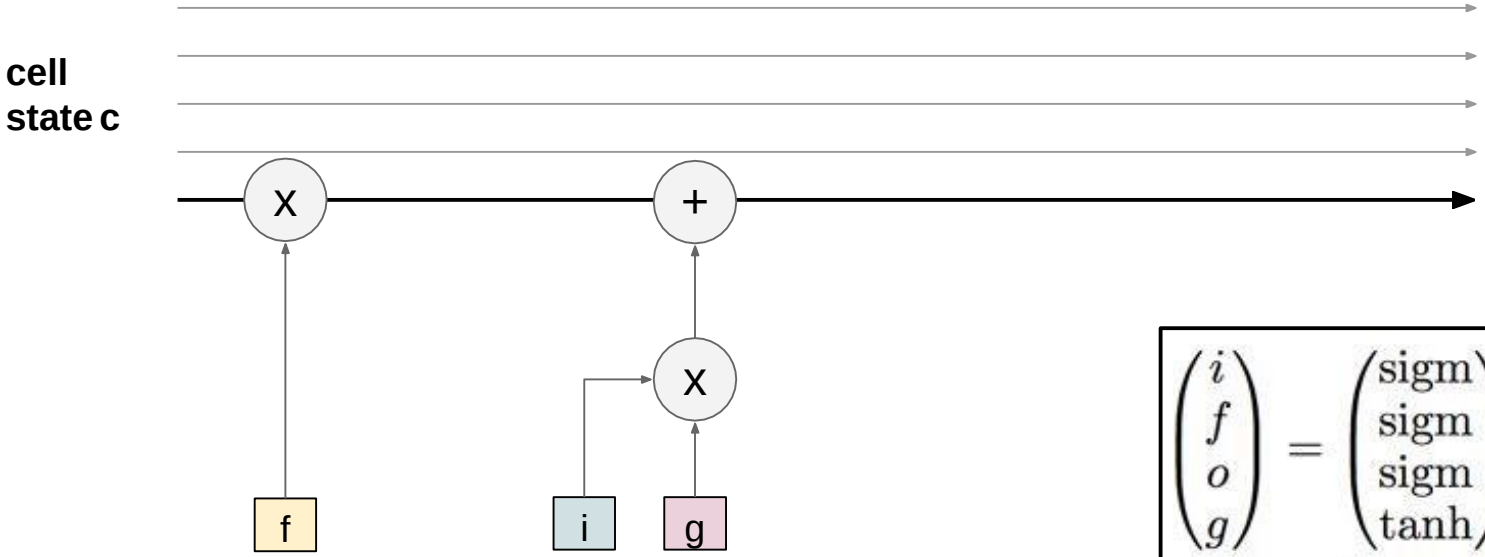


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

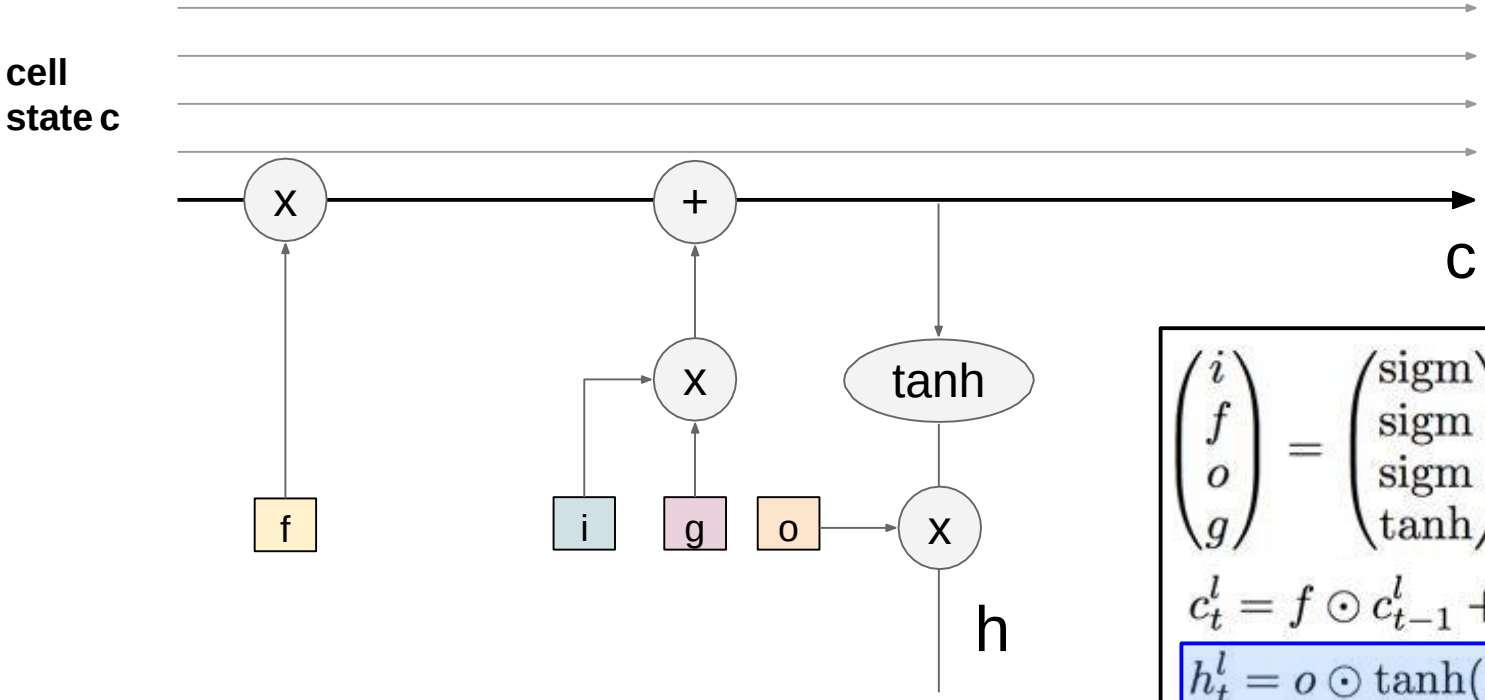


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

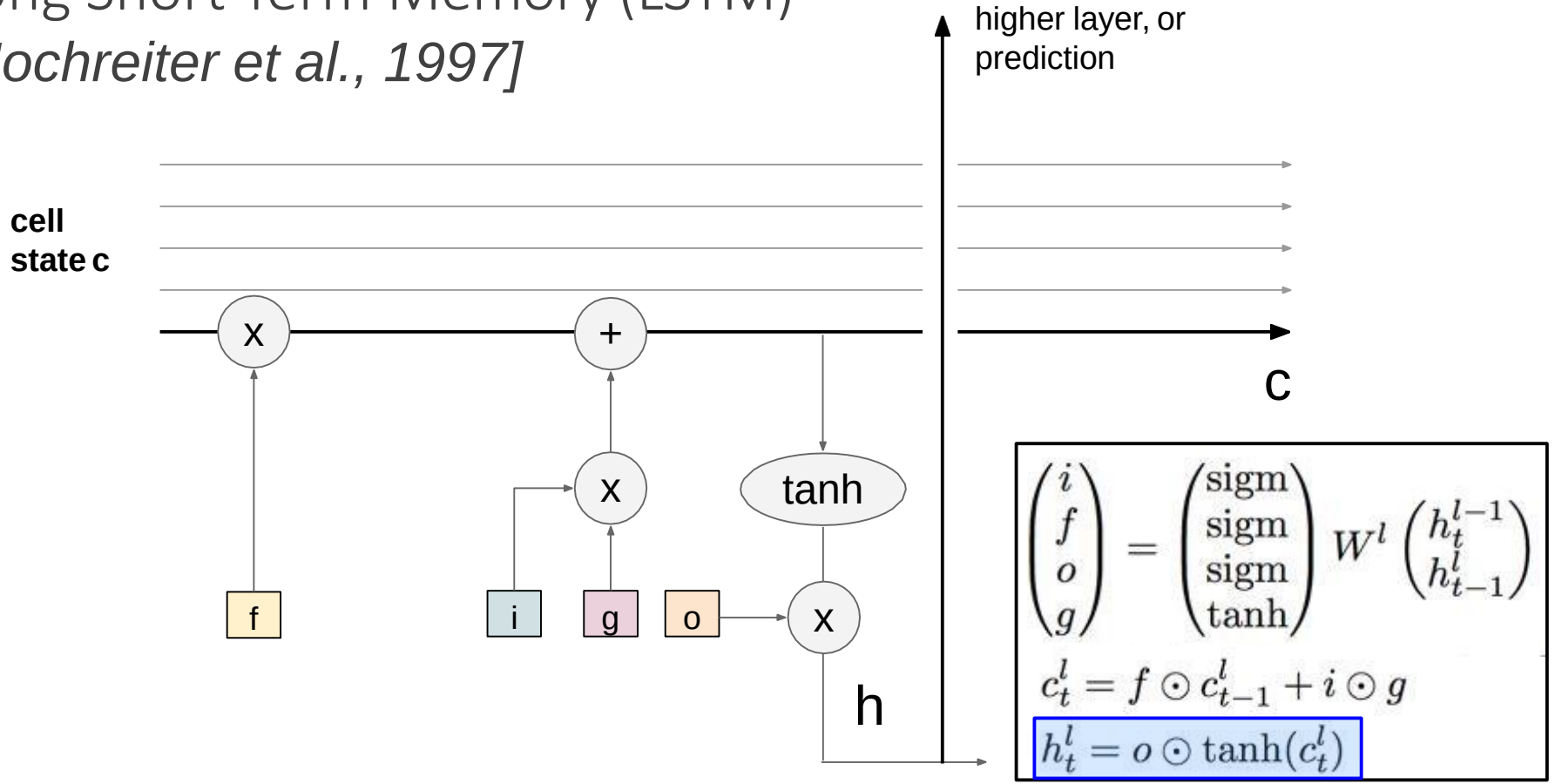


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

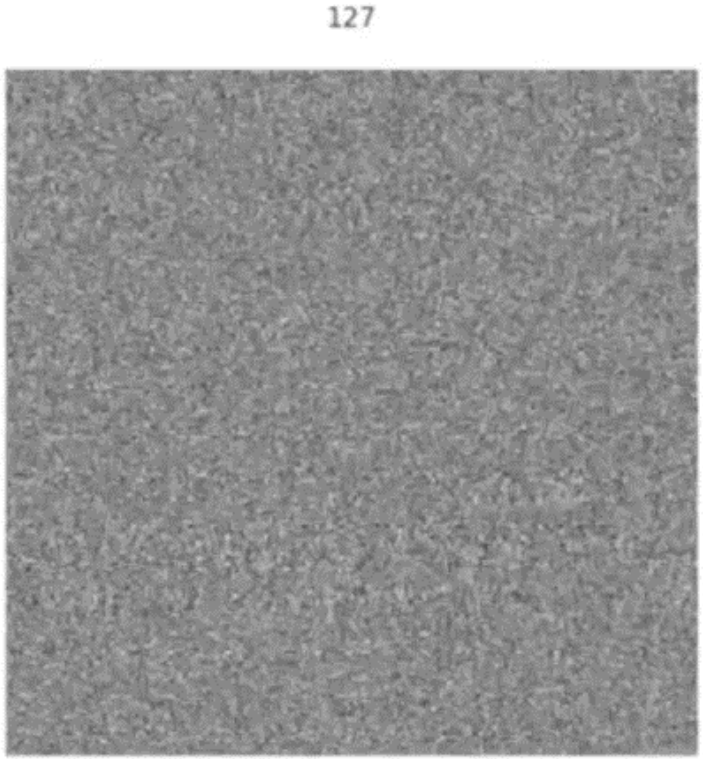
$$h_t^l = o \odot \tanh(c_t^l)$$

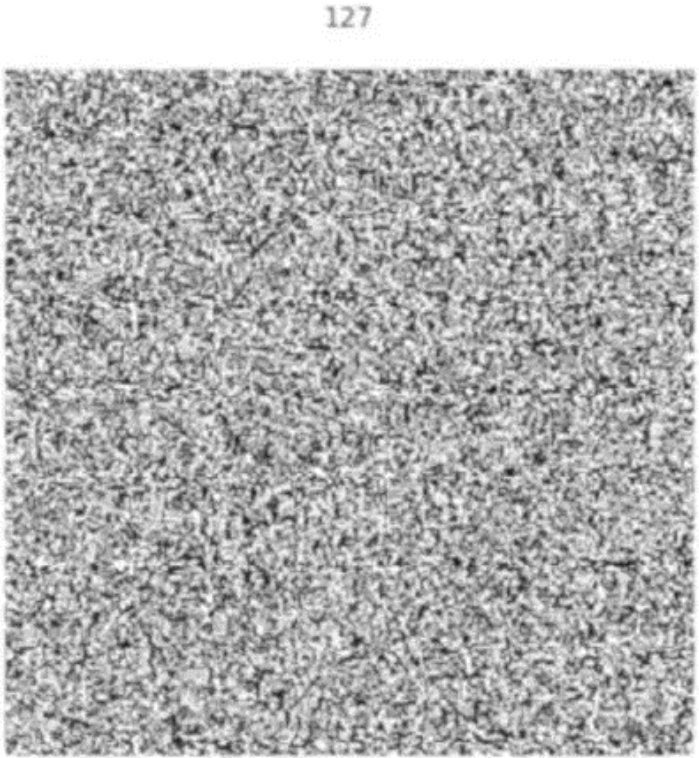
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]



Understanding gradient flow dynamics

Backprop signal



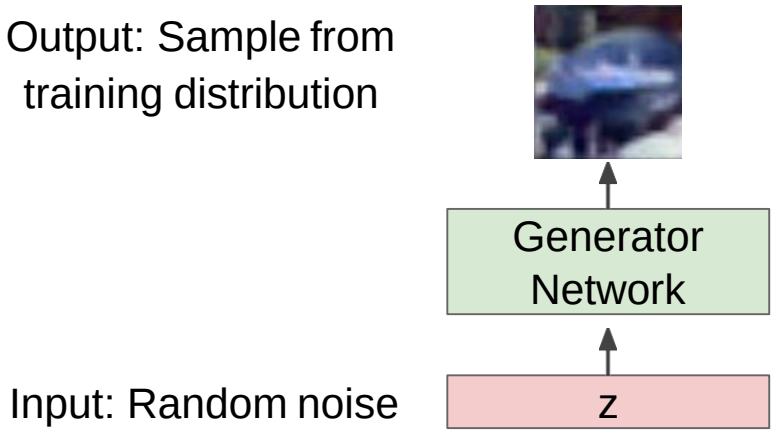


Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

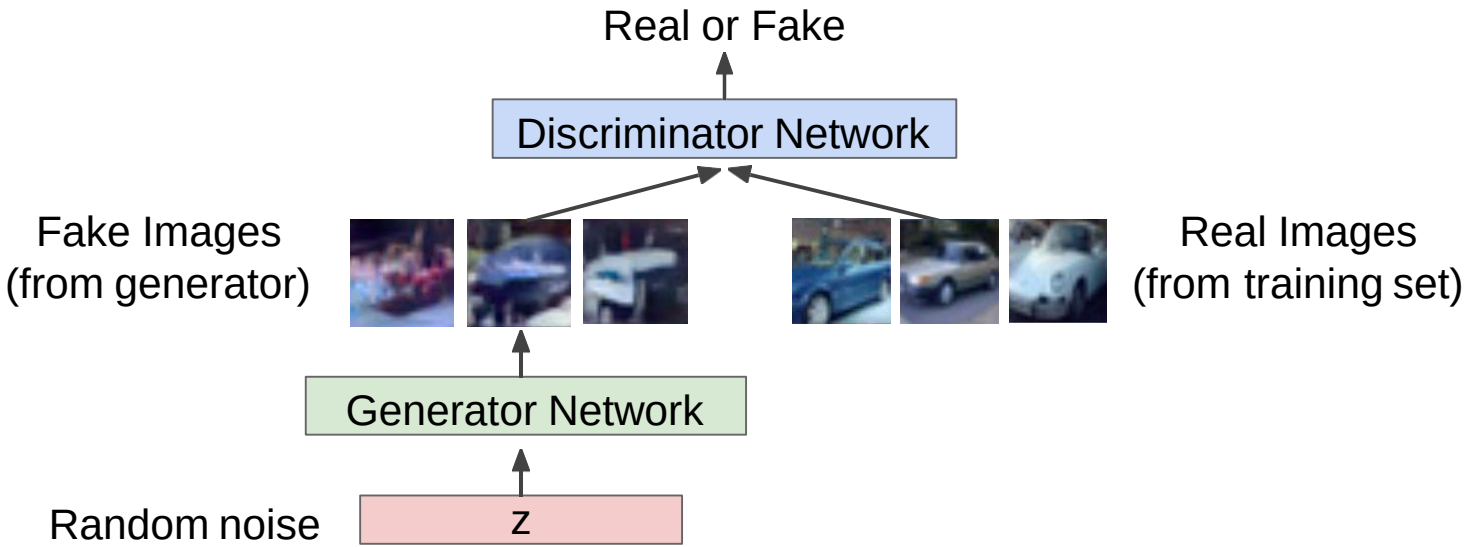
Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.



Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

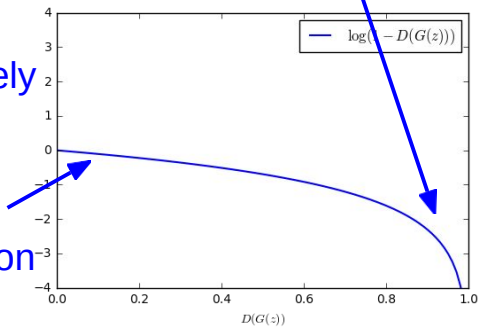
2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

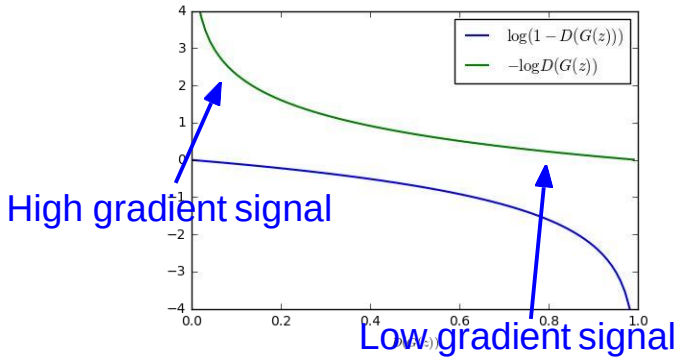
1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

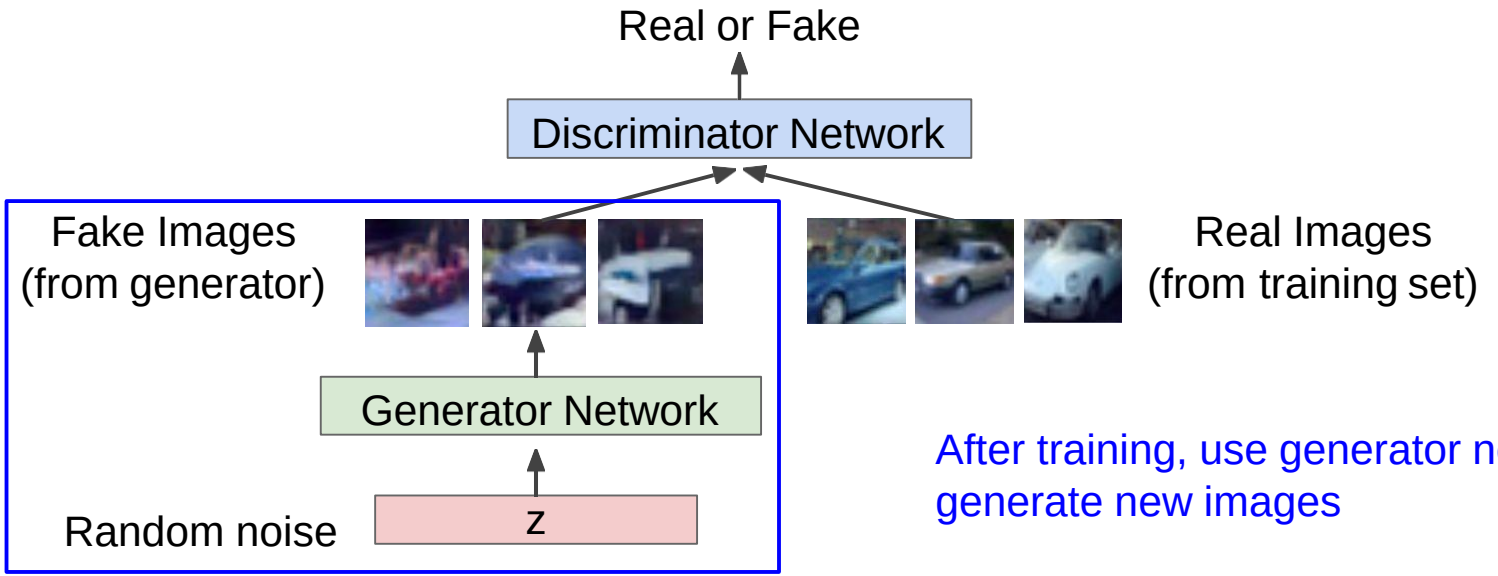
Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

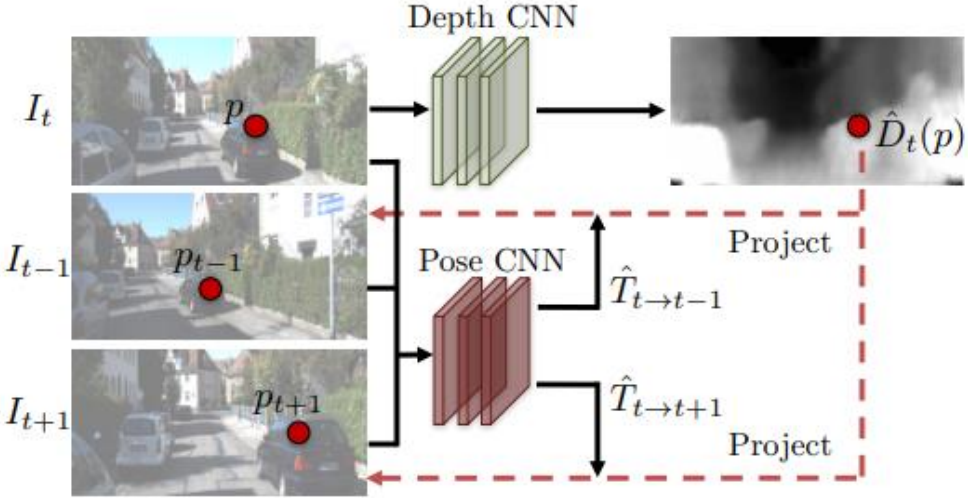
Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



After training, use generator network to generate new images

Case Studies

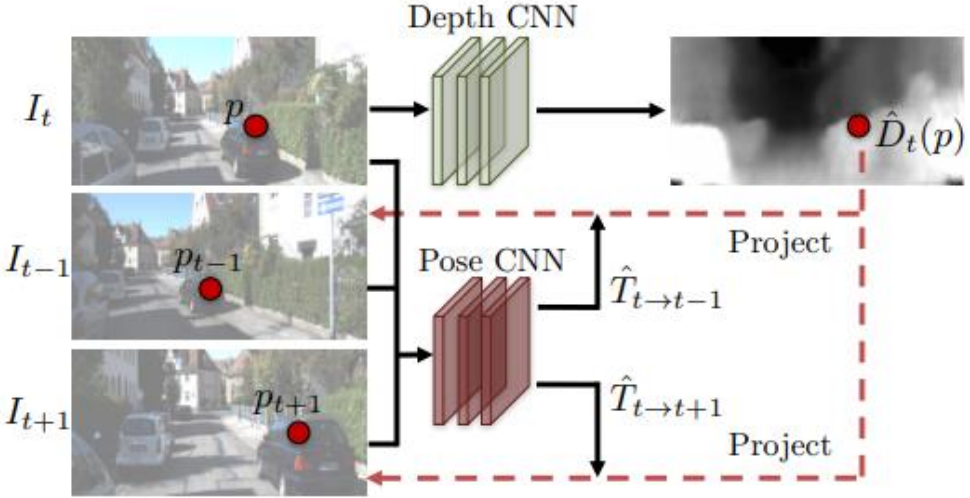
Depth Estimation



Overview of the supervision pipeline based on view synthesis. The depth network takes only the target view as input, and outputs a per-pixel depth map $\hat{D}_t$. The pose network takes both the target view (I_t) and the nearby/source views (e.g., I_{t-1} and I_{t+1}) as input, and outputs the relative camera poses ($\hat{T}_{t\to t-1}, \hat{T}_{t\to t+1}$). The outputs of both networks are then used to inverse warp the source views (see Sec. 3.2) to reconstruct the target view, and the photometric reconstruction loss is used for training the CNNs. By utilizing view synthesis as supervision, we are able to train the entire framework in an unsupervised manner from videos.

Zhou, Brown, Snavely, Lowe. Unsupervised learning of depth and ego-motion from video. IEEE CVPR, 2017.

Depth Estimation



$$p_s \sim K\hat{T}_{t\to s}\hat{D}_t(p_t)K^{-1}p_t$$

projected coordinates onto the source view

camera intrinsics matrix

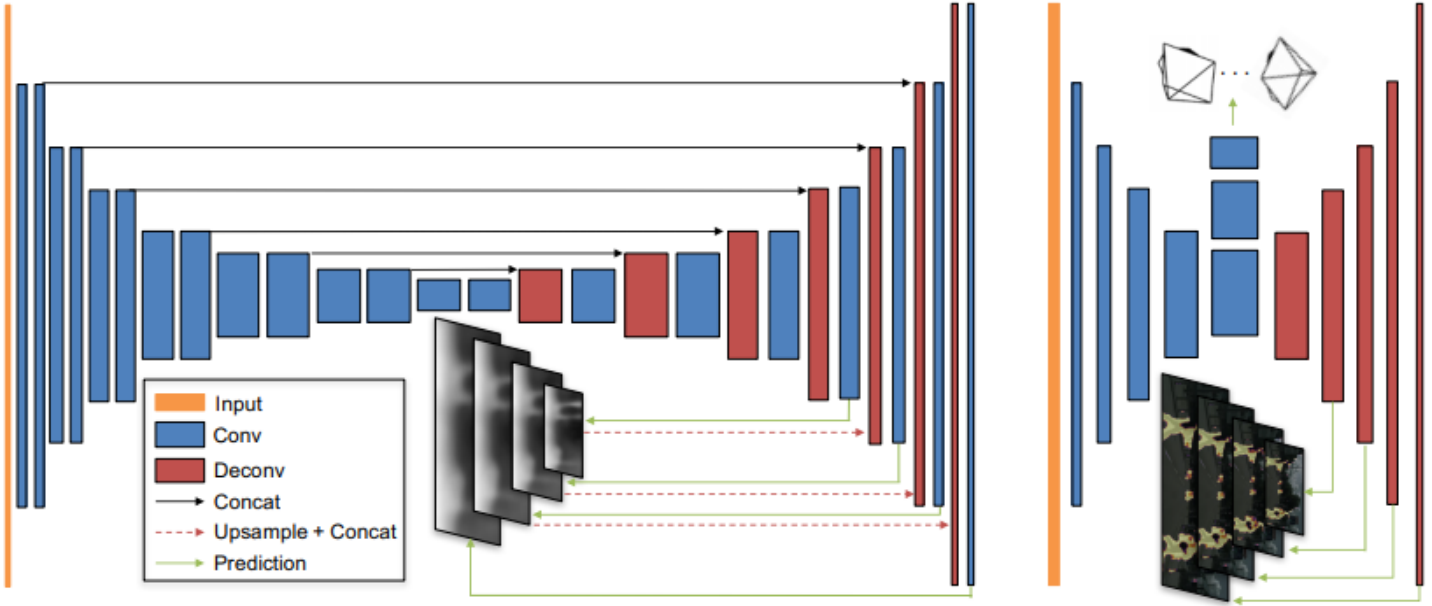
relative pose

depth

homogeneous coordinates of a pixel

Zhou, Brown, Snavely, Lowe. Unsupervised learning of depth and ego-motion from video. IEEE CVPR, 2017.

Depth Estimation

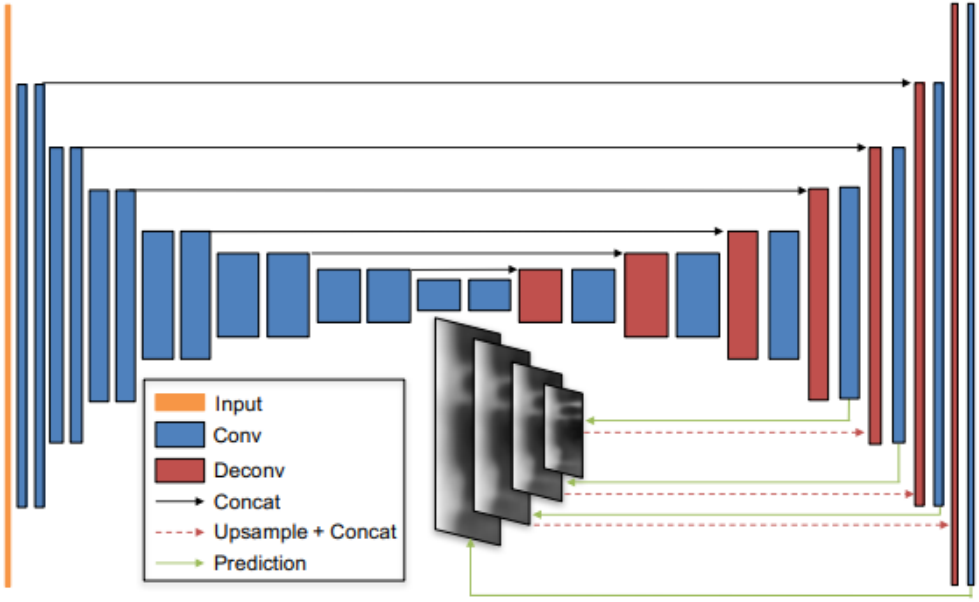


(a) Single-view depth network (b) Pose/explainability network

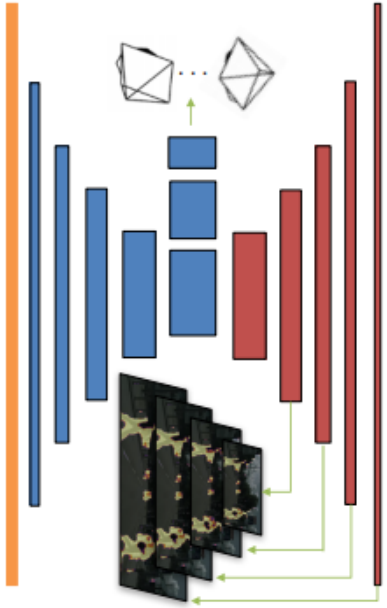
Network architecture for our depth/pose/explainability prediction modules. The width and height of each rectangular block indicates the output channels and the spatial dimension of the feature map at the corresponding layer respectively, and each reduction/increase in size indicates a change by the factor of 2. (a) For single-view depth, we adopt the DispNet [35] architecture with multi-scale side predictions. The kernel size is 3 for all the layers except for the first 4 conv layers with 7, 7, 5, 5, respectively. The number of output channels for the first conv layer is 32. (b) The pose and explainabilty networks share the first few conv layers, and then branch out to predict 6-DoF relative pose and multi-scale explainability masks, respectively. The number of output channels for the first conv layer is 16, and the kernel size is 3 for all the layers except for the first two conv and the last two deconv/prediction layers where we use 7, 5, 5, 7, respectively. See Section 3.5 for more details.

Zhou, Brown, Snavely, Lowe. Unsupervised learning of depth and ego-motion from video.
IEEE CVPR, 2017.

Depth Estimation



(a) Single-view depth network



(b) Pose/explainability network

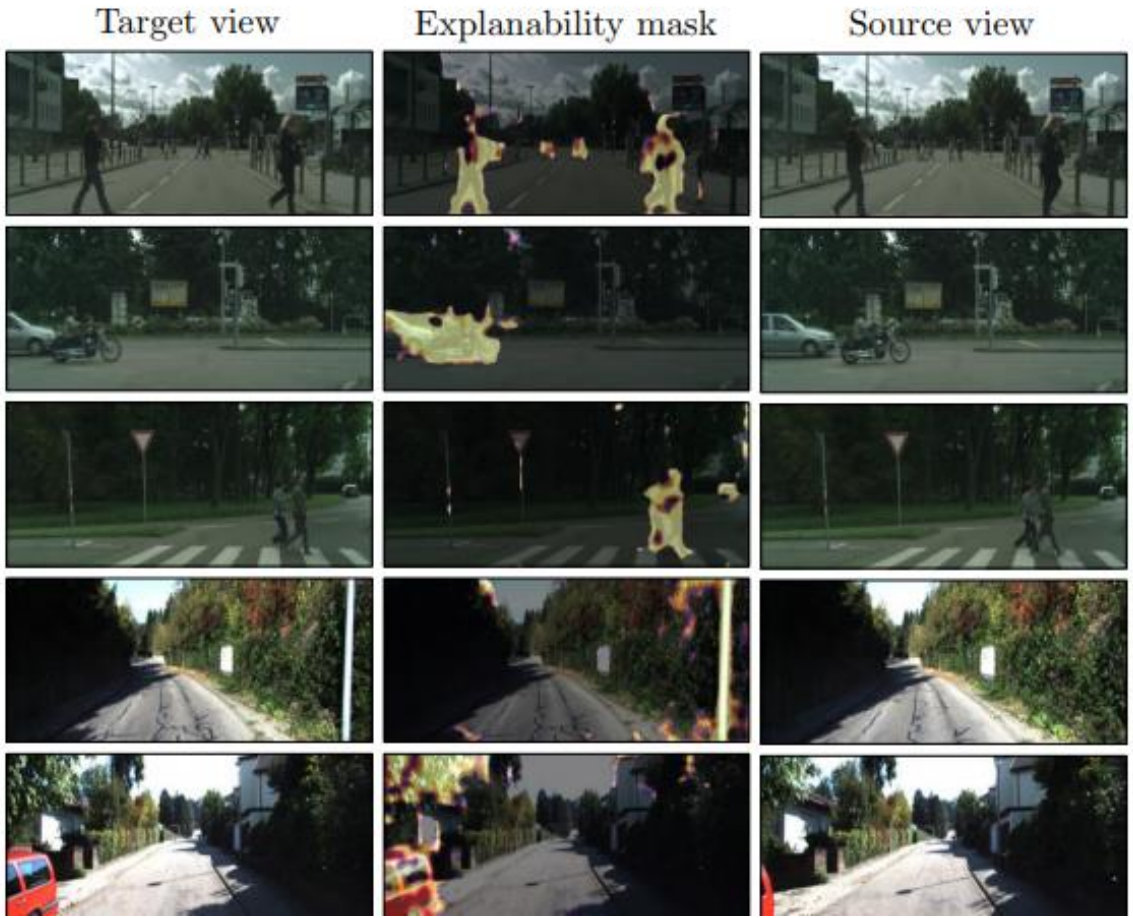
reliability of estimates per pixel

$$\mathcal{L}_{vs} = \sum_{<I_1,\ldots,I_N>\in\mathcal{S}} \sum_{p} \hat{E}_s(p)|I_t(p) - \hat{I}_s(p)|$$

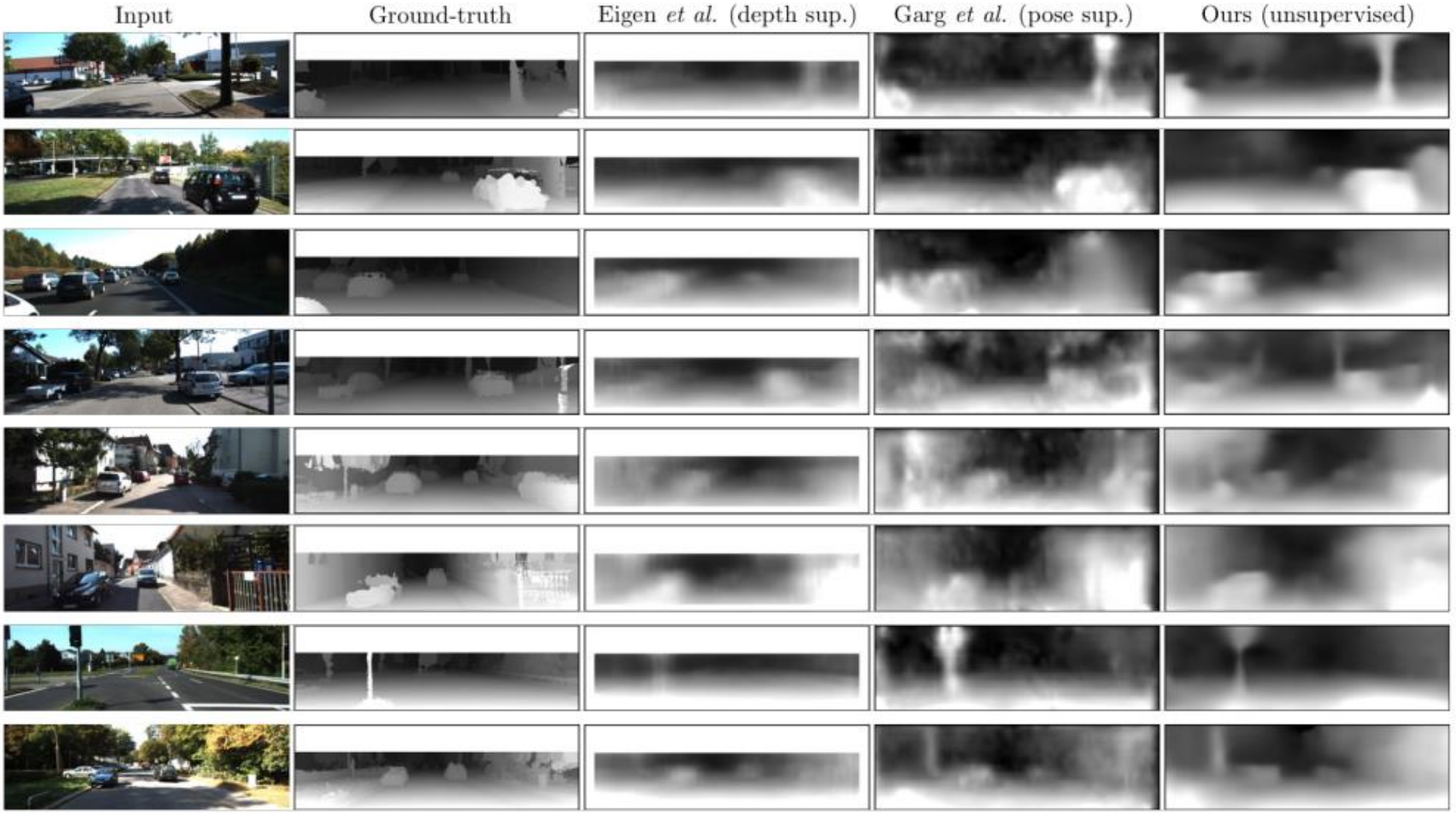
Zhou, Brown, Snavely, Lowe. Unsupervised learning of depth and ego-motion from video. IEEE CVPR, 2017.

Depth Estimation

Sample visualizations of the explainability masks. Highlighted pixels are predicted to be unexplainable by the network due to motion (rows 1–3), occlusion/visibility (rows 4–5), etc.



Zhou, Brown, Snavely, Lowe. Unsupervised learning of depth and ego-motion from video.
IEEE CVPR, 2017.



Zhou, Brown, Snavely, Lowe. Unsupervised learning of depth and ego-motion from video.
IEEE CVPR, 2017.

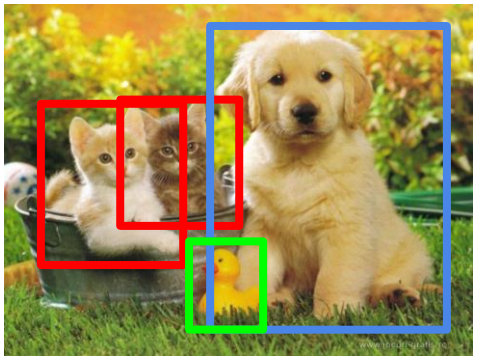
Object Detection

Classification

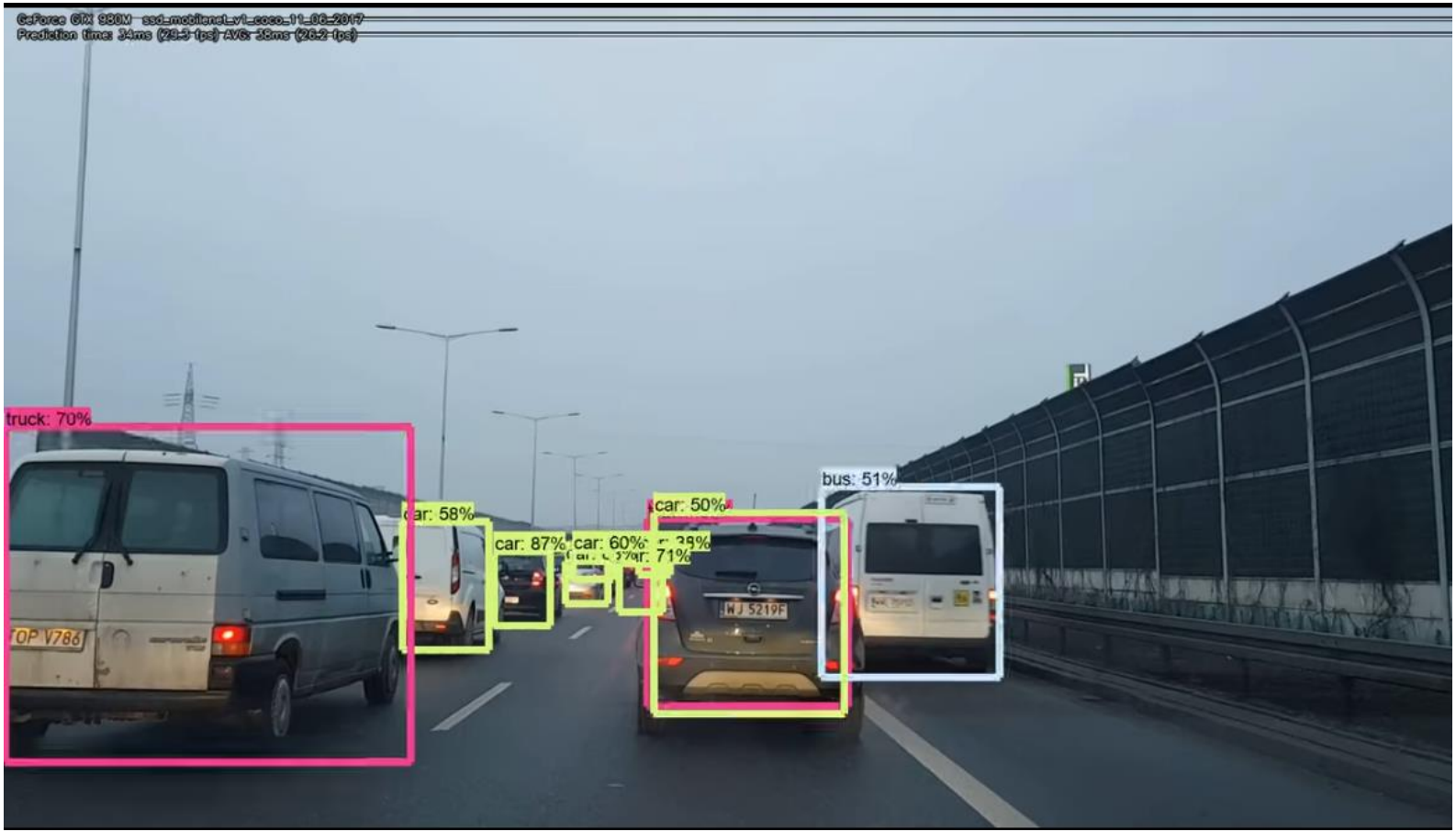
Classification + Localization

Object Detection

Instance Segmentation

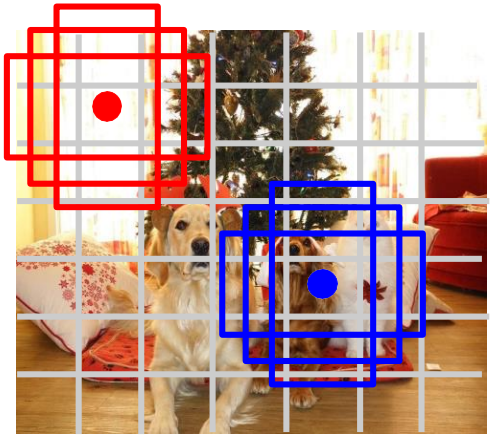


Object Detection



Single-Stage Object Detectors: YOLO

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:

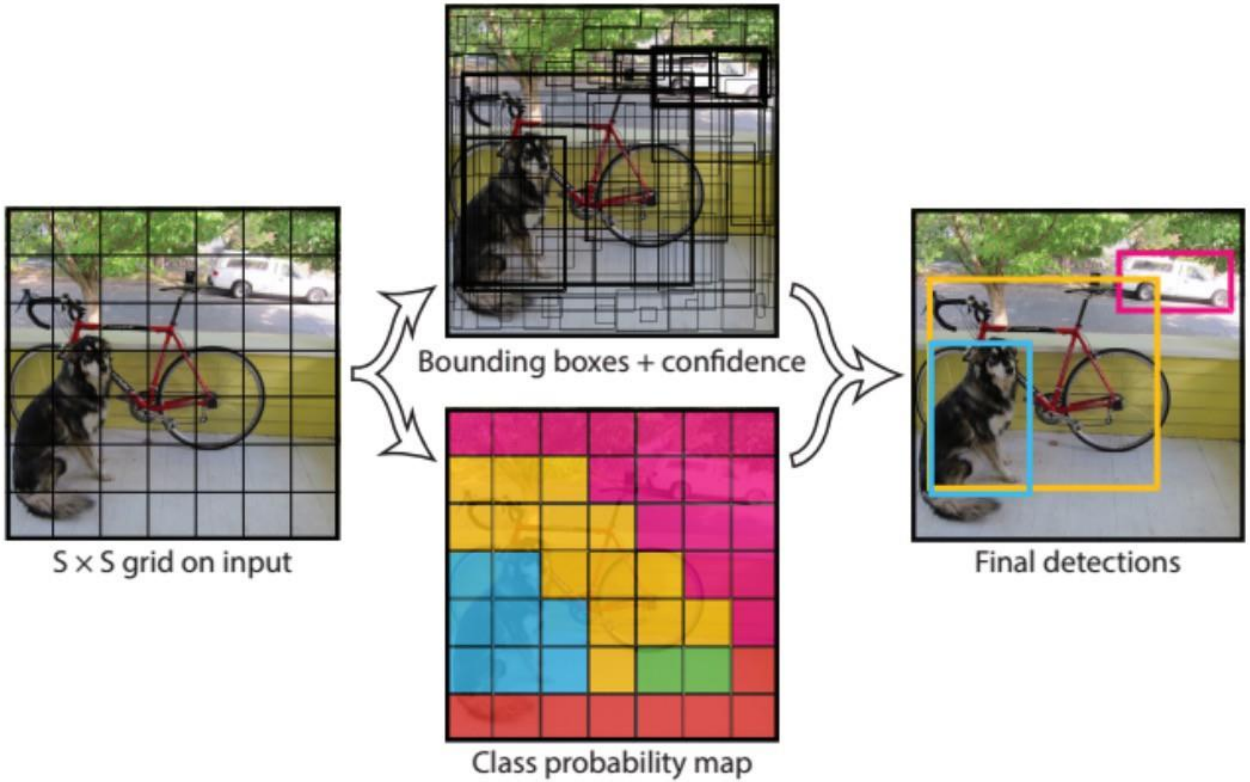
- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like Region Proposal Networks, but category-specific!

Output:
7 x 7 x (5 * B + C)

# of boxes:
7 x 7 x B

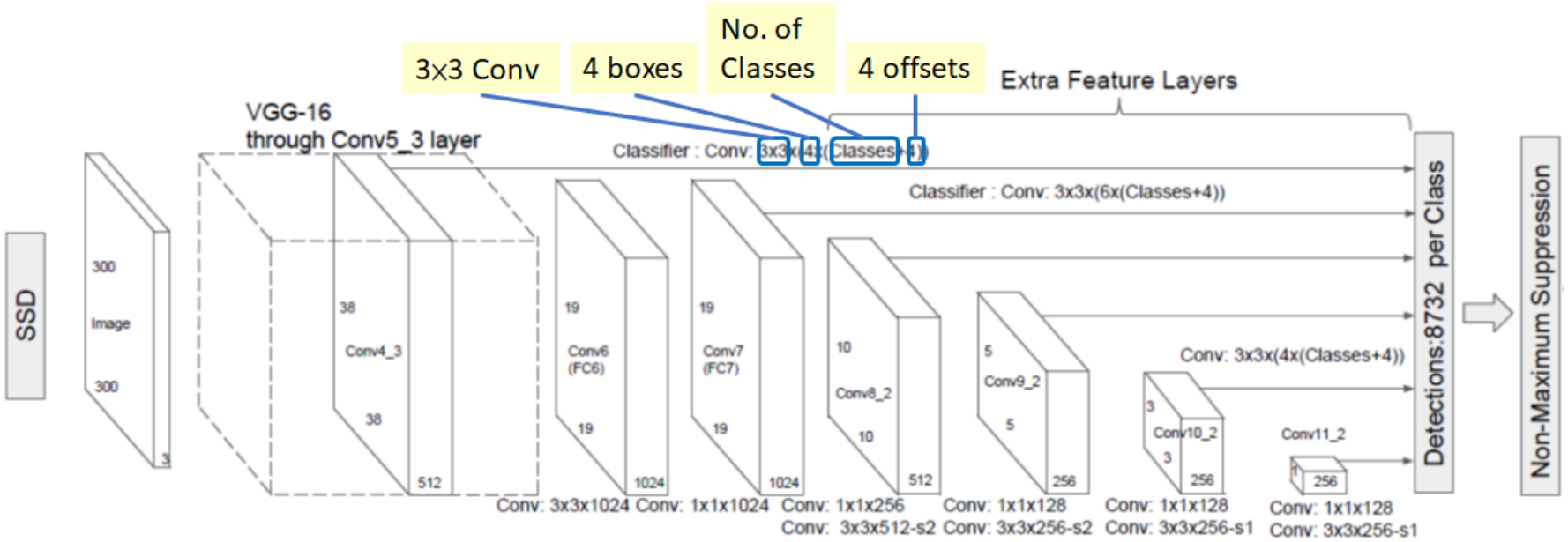
Redmon et al, “You Only Look Once: Unified, Real-Time Object Detection”, CVPR 2016

Single-Stage Object Detectors: YOLO



Redmon et al, “You Only Look Once: Unified, Real-Time Object Detection”, CVPR 2016

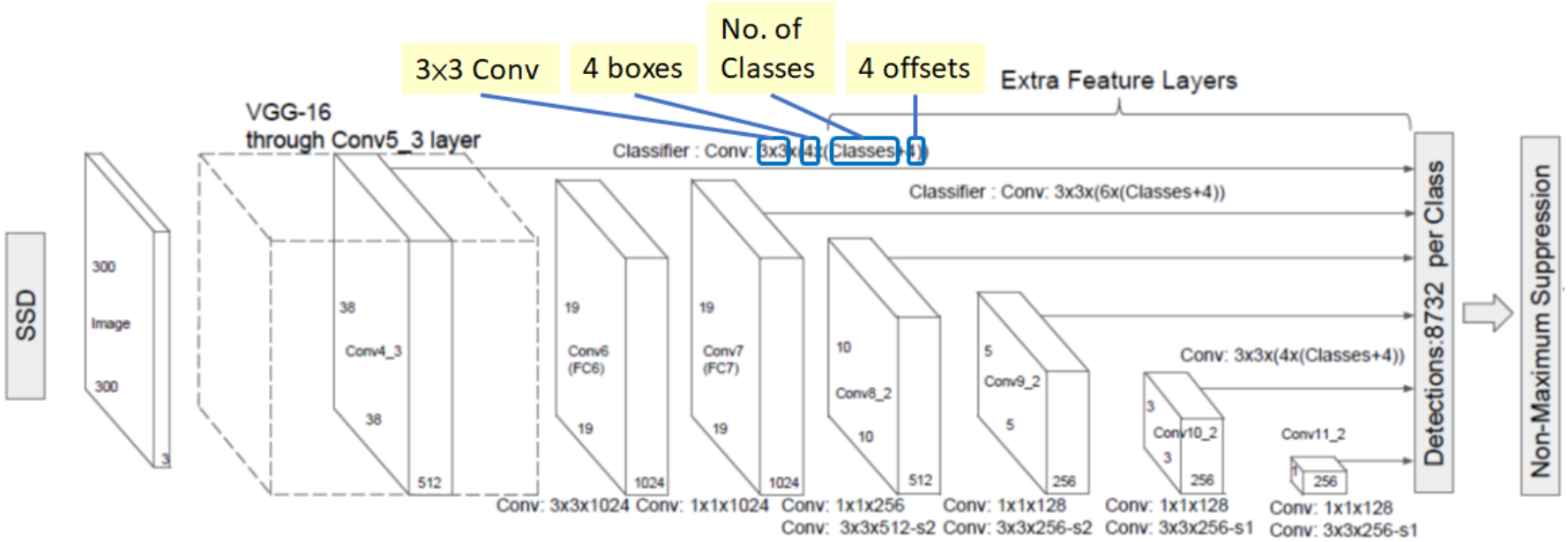
Object Detection: Single Shot Detector (SSD)



Liu et al, “SSD: Single-Shot MultiBox Detector”, ECCV 2016

https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11

Object Detection: Single Shot Detector (SSD)



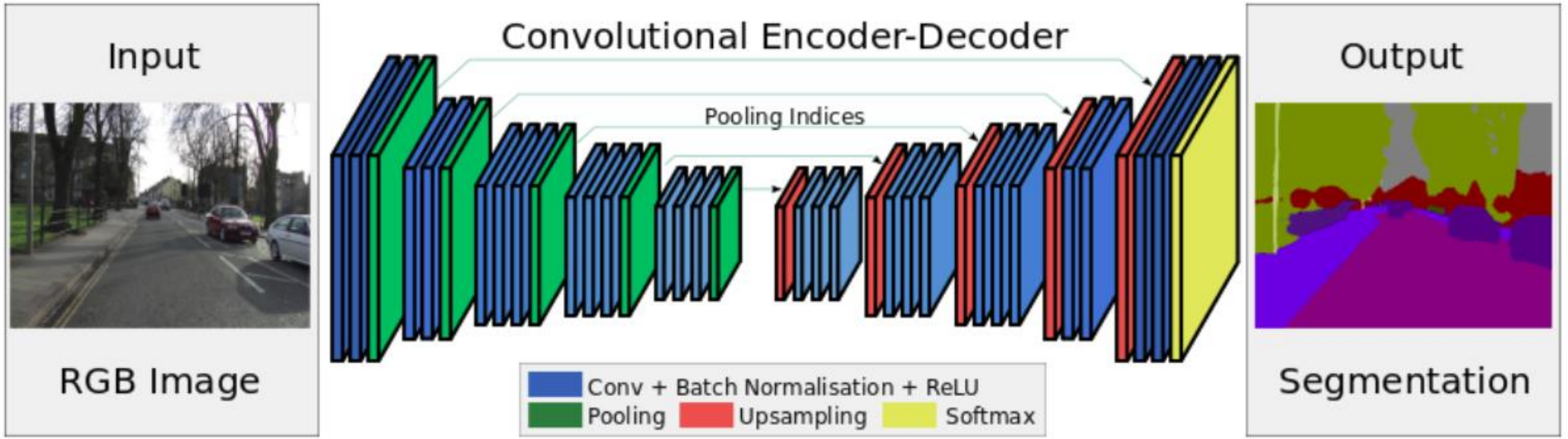
- Conv4_3: 38×38×4 = 5776 boxes (4 boxes for each location)
- Conv7: 19×19×6 = 2166 boxes (6 boxes for each location)
- Conv8_2: 10×10×6 = 600 boxes (6 boxes for each location)
- Conv9_2: 5×5×6 = 150 boxes (6 boxes for each location)
- Conv10_2: 3×3×4 = 36 boxes (4 boxes for each location)
- Conv11_2: 1×1×4 = 4 boxes (4 boxes for each location)
- In total **SSD has 8732 bounding boxes which is much more than that of YOLO**.

Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11

Instance Segmentation

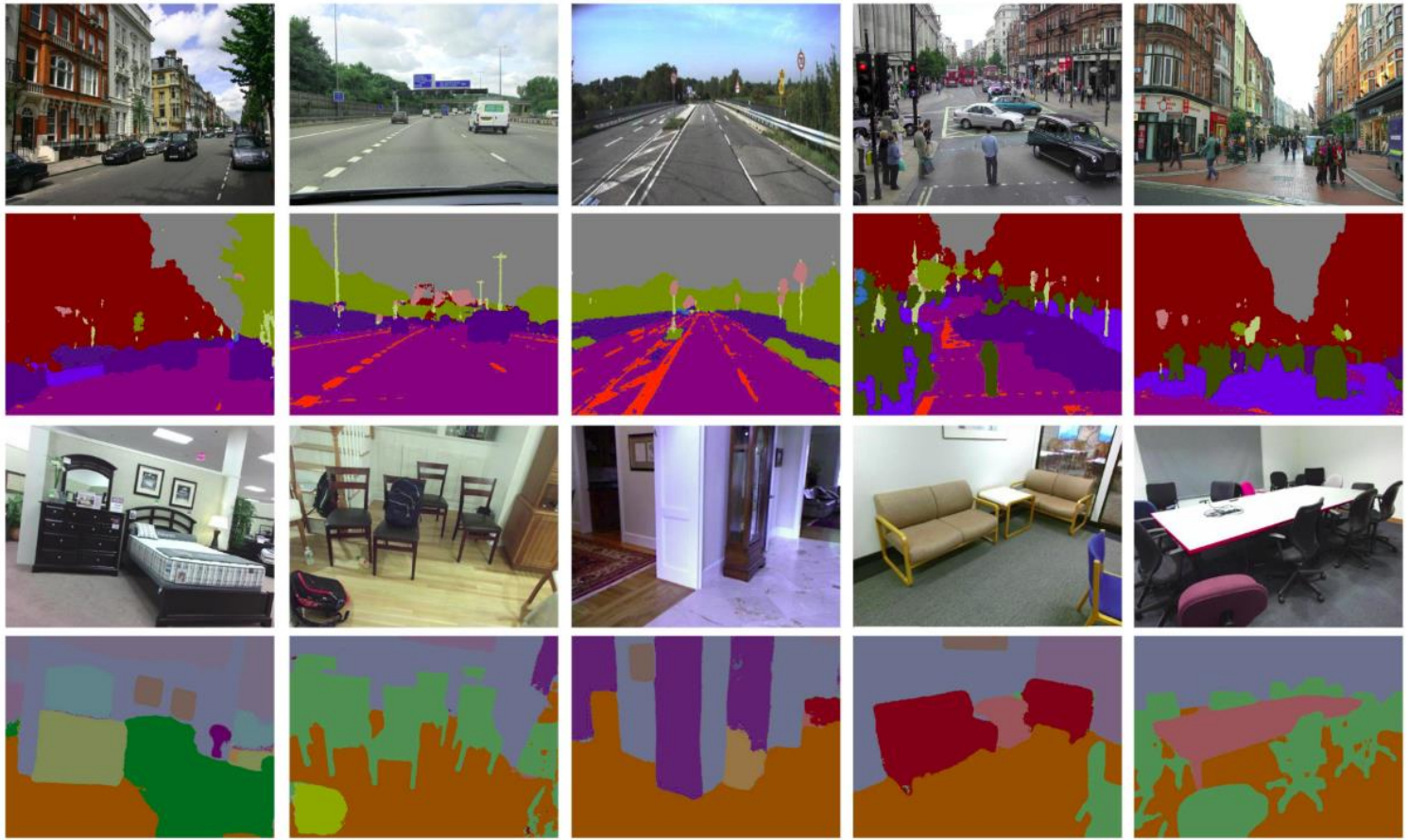
Classification	Semantic Segmentation	Object Detection	**Instance Segmentation**
			
CAT	GRASS, CAT, TREE, SKY	DOG, DOG, CAT	**DOG, DOG, CAT**

Instance Segmentation: SegNet



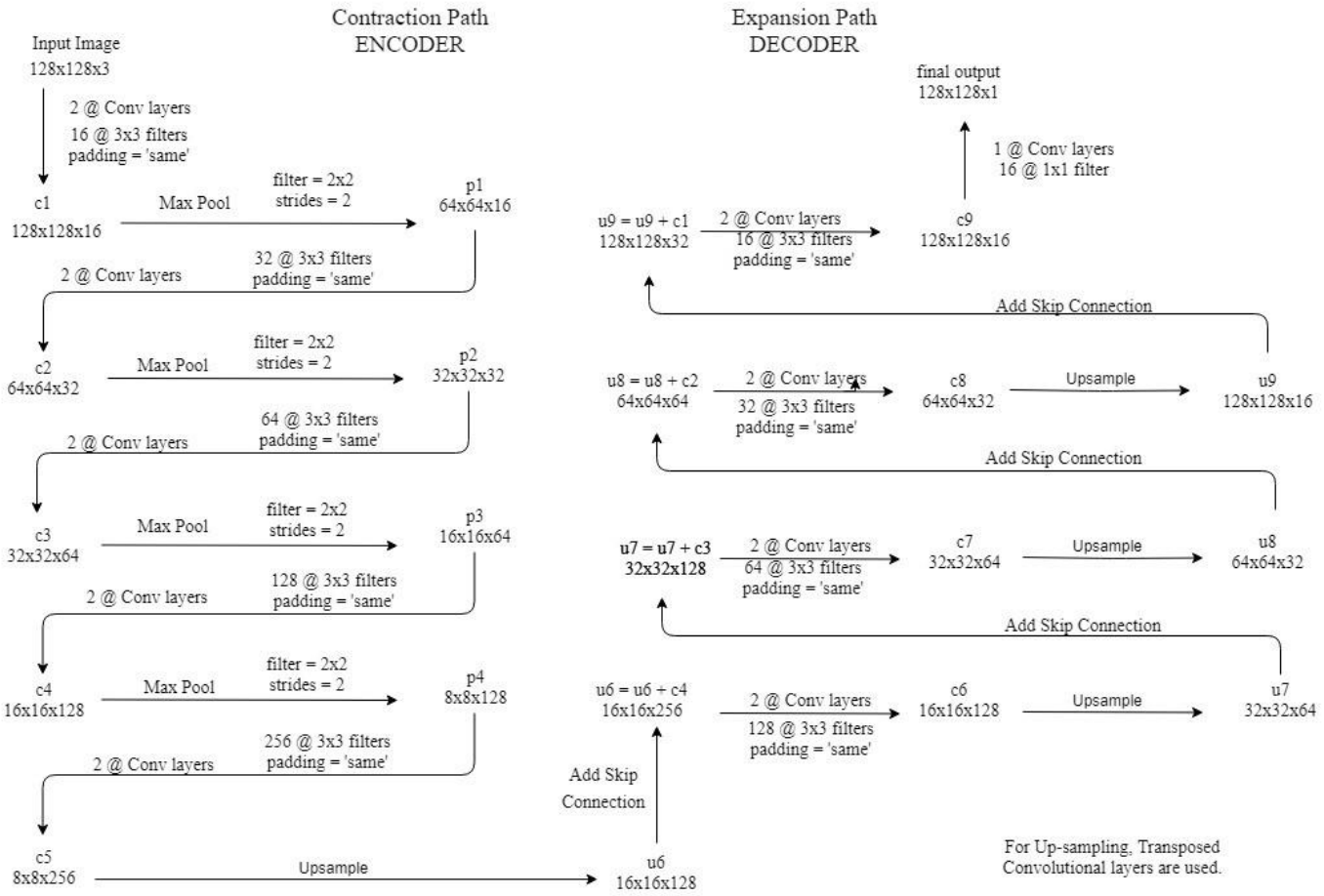
Badrinarayanan, Kendall, Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017*

Instance Segmentation: SegNet



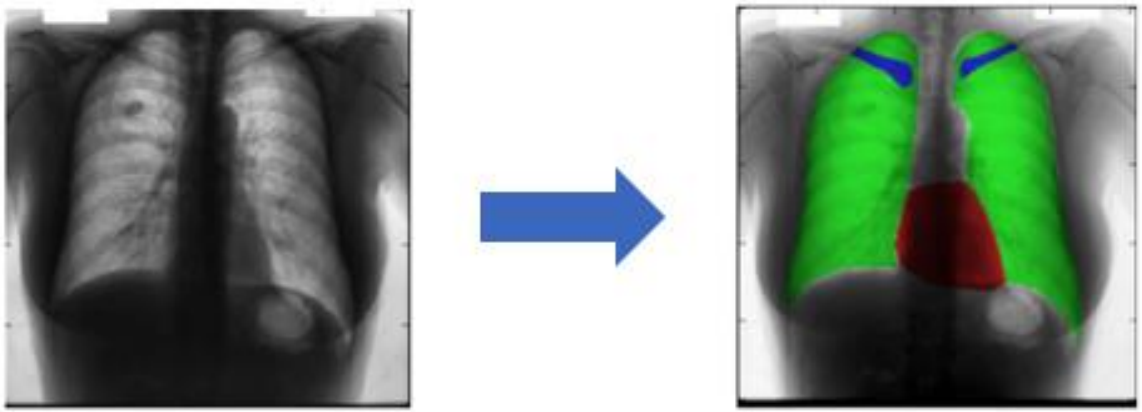
Badrinarayanan, Kendall, Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017*

Instance Segmentation: UNet



Ronneberger, Fischer, Brox. "U-net: Convolutional networks for biomedical image segmentation." International Conference on Medical Image Computing and Computer-assisted Intervention, 2015

Other Applications: Biomedical Image Diagnosis

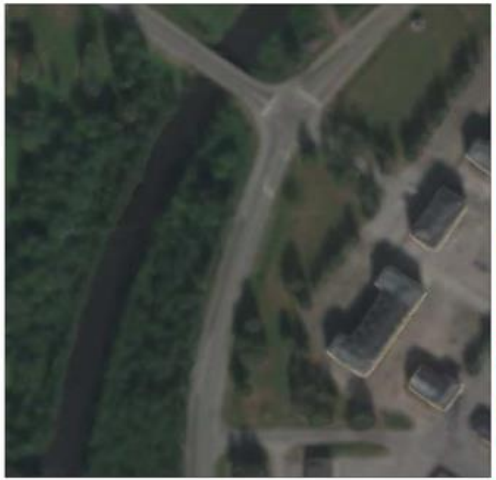


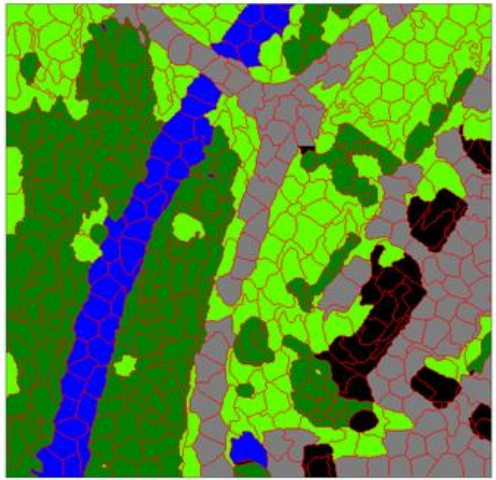
Input Image

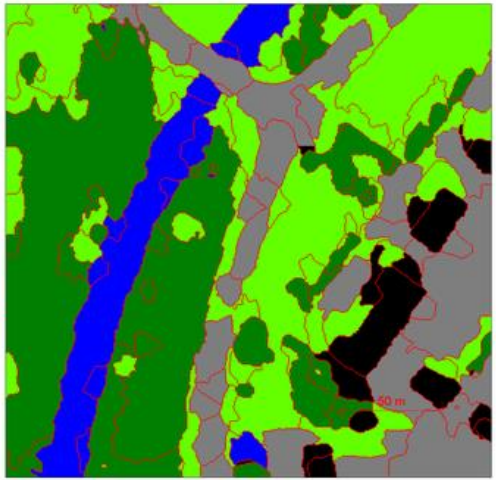
Segmented Image

Source: https://arxiv.org/abs/1701.08816

Other Applications: Geo Sensing

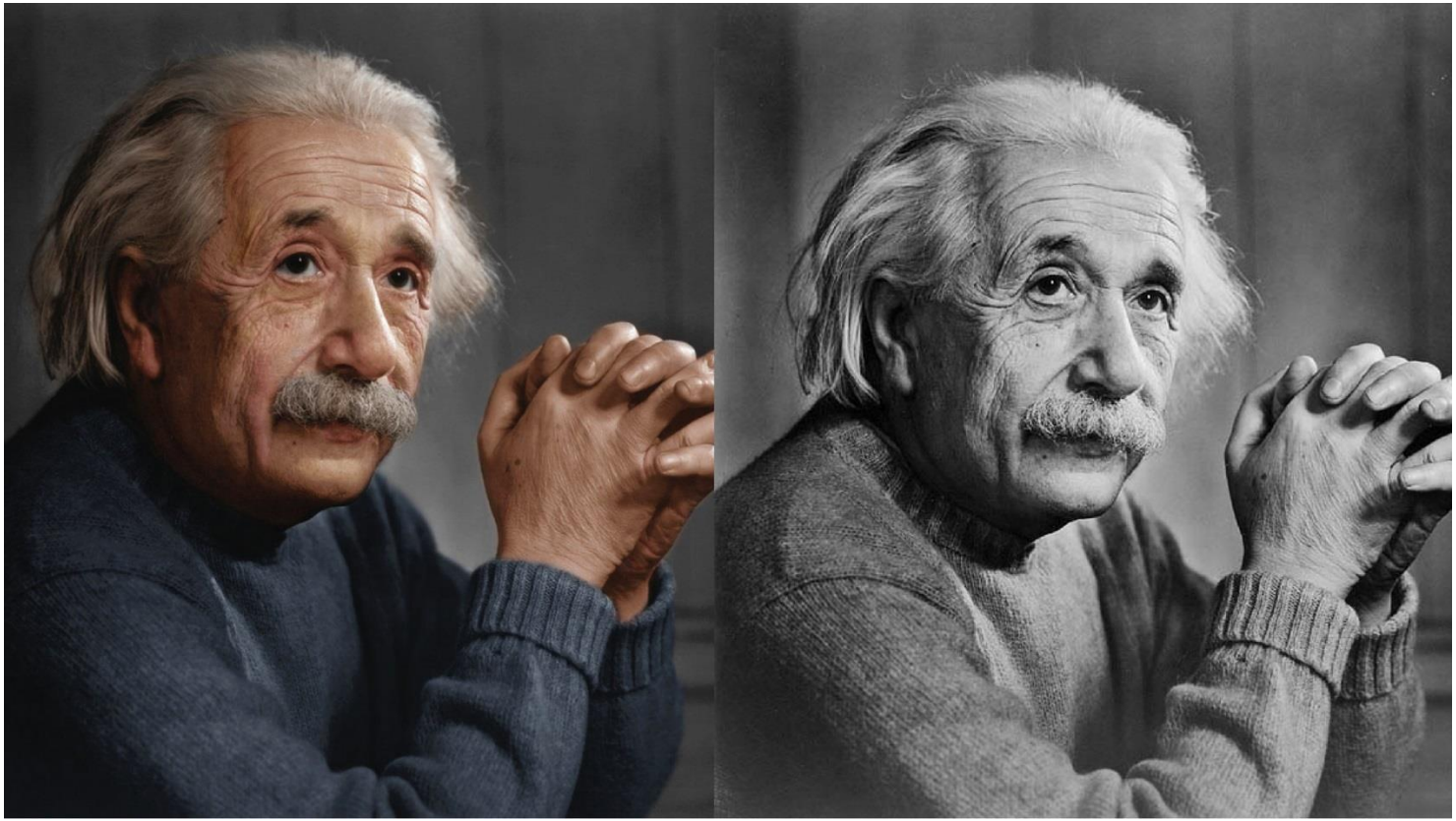






Other Applications: Super Resolution

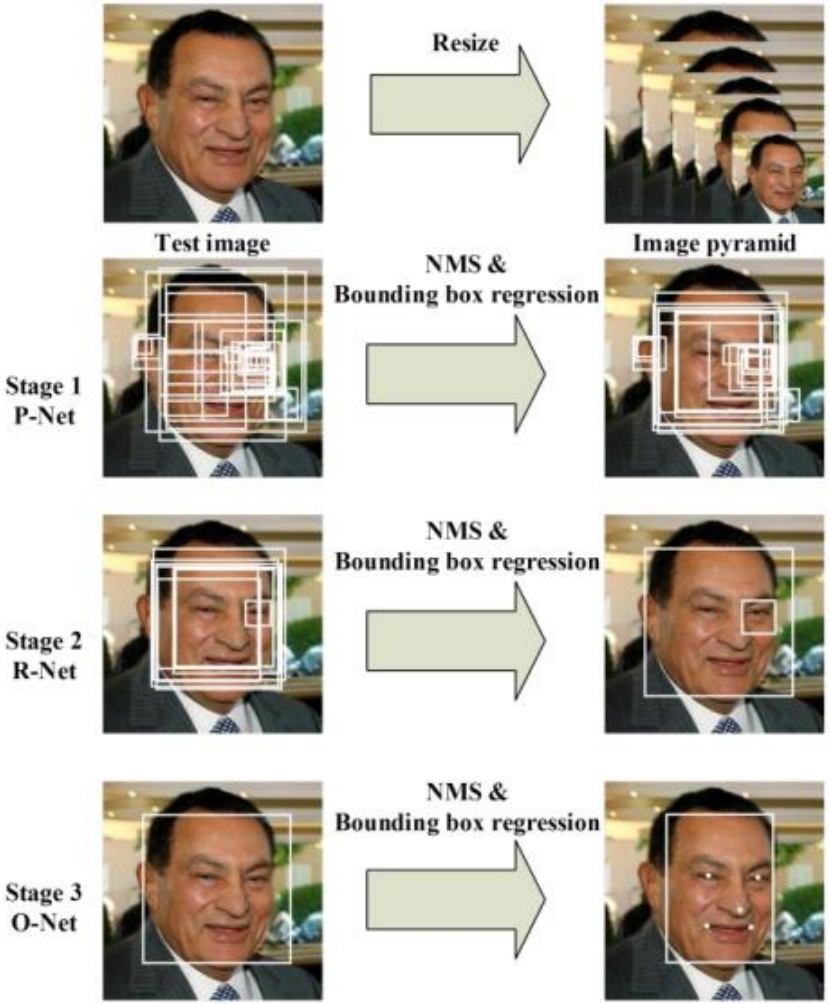
Other Applications: Image Colorization



Other Applications: Rain/Fog Removal

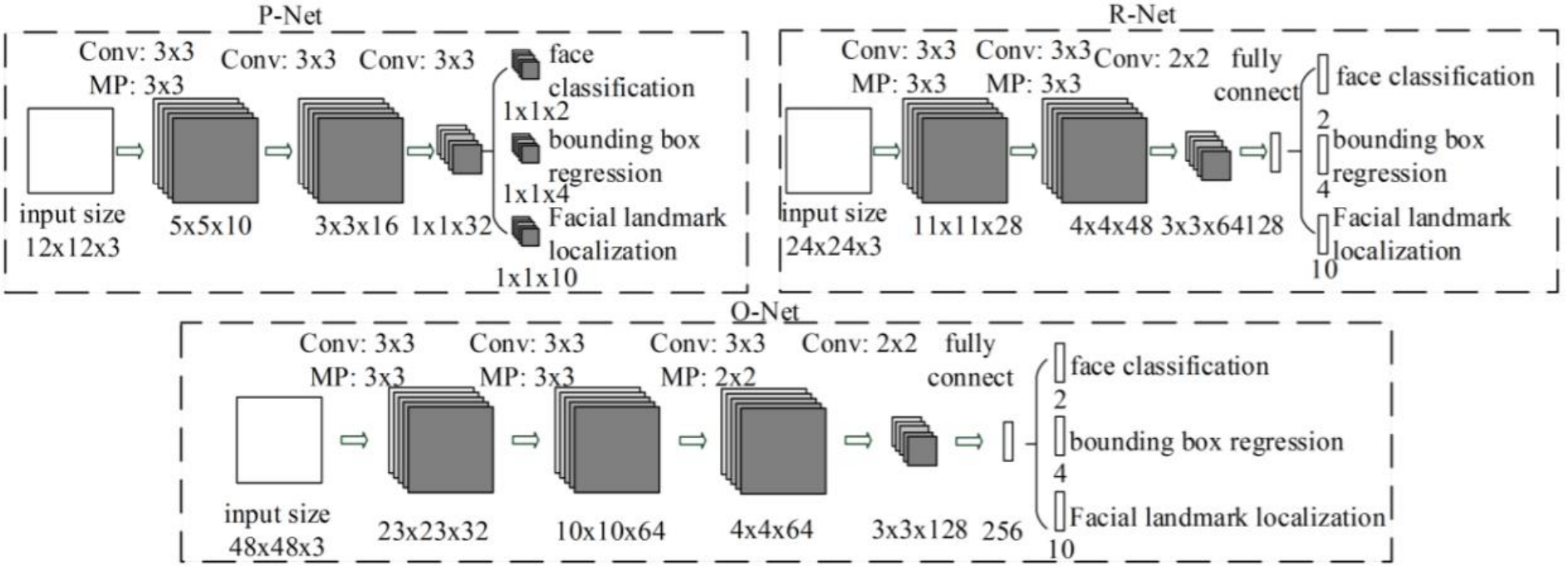
Face Detection

1. A fully convolutional network (Proposal Network: P-Net) is used to obtain proposed regions and their bounding box regression vectors. Apply non-maxima suppression (NMS) to merge highly overlapped regions.
2. All proposed regions will be fed to another CNN (Refine Network: R-Net), which will reject a large number of false candidates, performs another calibration with bounding box regression and also NMS candidate merge.
3. Through Output Network (O-Net), face detections are further refined.



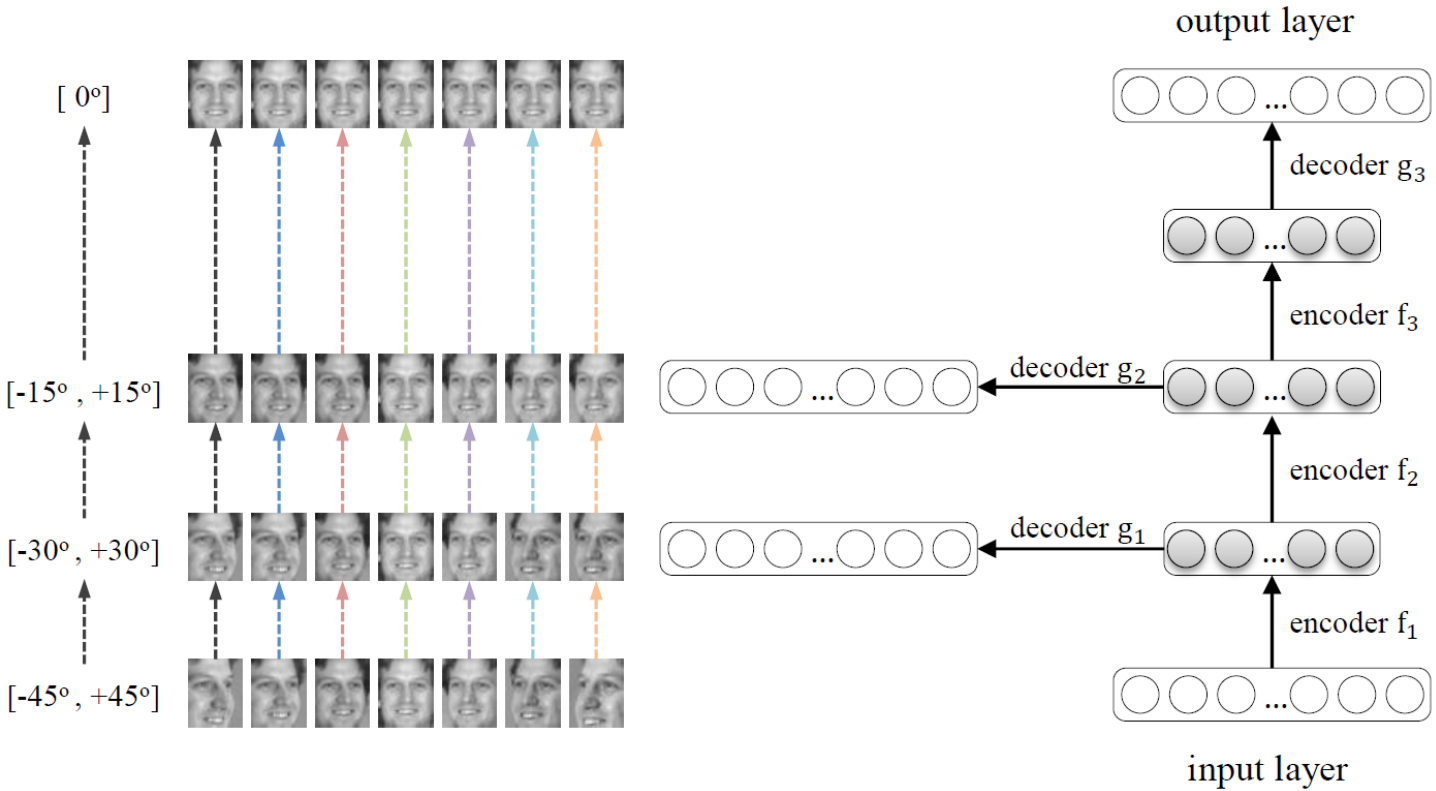
Zhang, Zhang, Li, Qiao. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks.
IEEE Signal Processing Letters, 2016.

Face Detection



Zhang, Zhang, Li, Qiao. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks.
IEEE Signal Processing Letters, 2016.

Head Pose Normalization

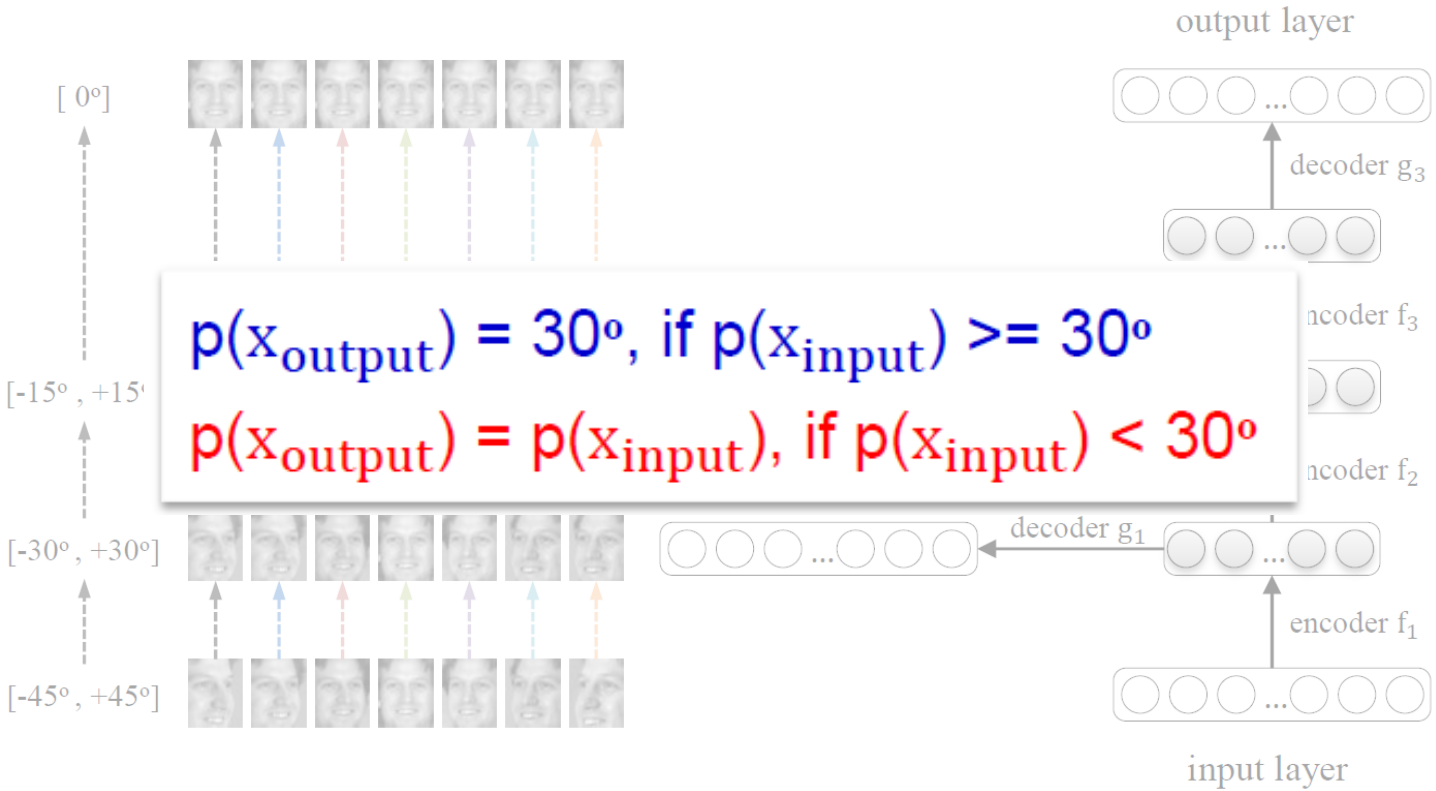


Kan, Shan, Chen. Stacked Progressive Auto-Encoders (SPAE) for Face Recognition Across Poses. IEEE CVPR 2014.

Head Pose Normalization

- **Aim:** Predicting view from one pose to another
- **Direct model:** e.g. Stacked de-noising Auto-Encoder
  - Regard non-fontal view as corrupted version of frontal view
  - But, it fails
    - Complex non-linear model
    - Easily overfit to “Small” data
- **Proposed idea:**
  - Pose changes smoothly (stage-wise non-linear)
  - Normalize the pose progressively

Head Pose Normalization

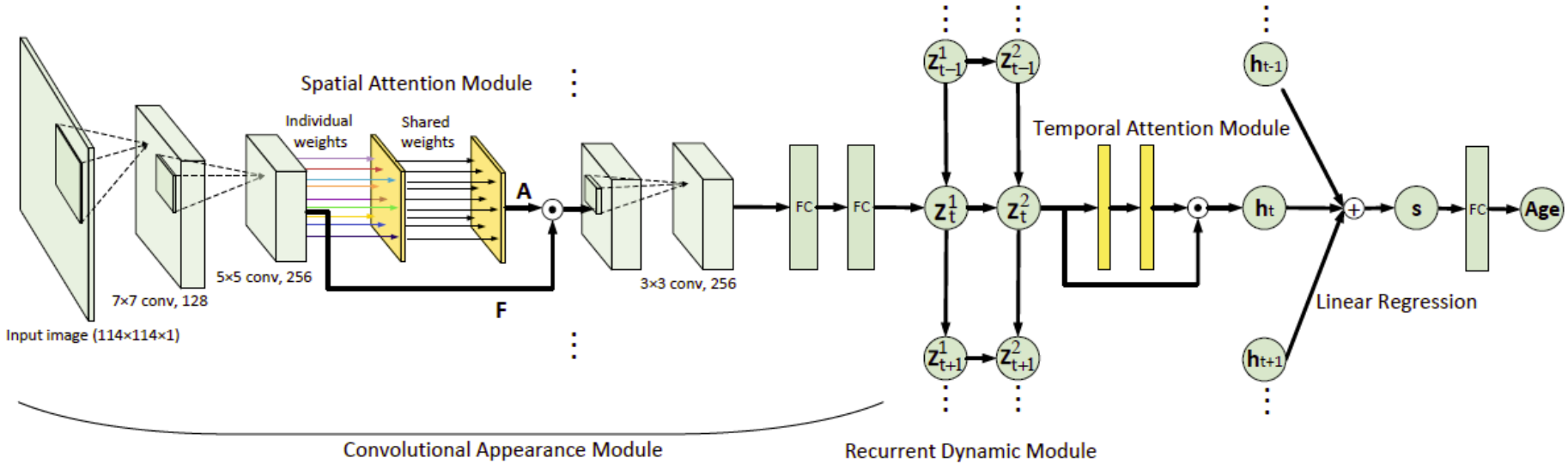


Kan, Shan, Chen. Stacked Progressive Auto-Encoders (SPAE) for Face Recognition Across Poses. IEEE CVPR 2014.

Head Pose Normalization

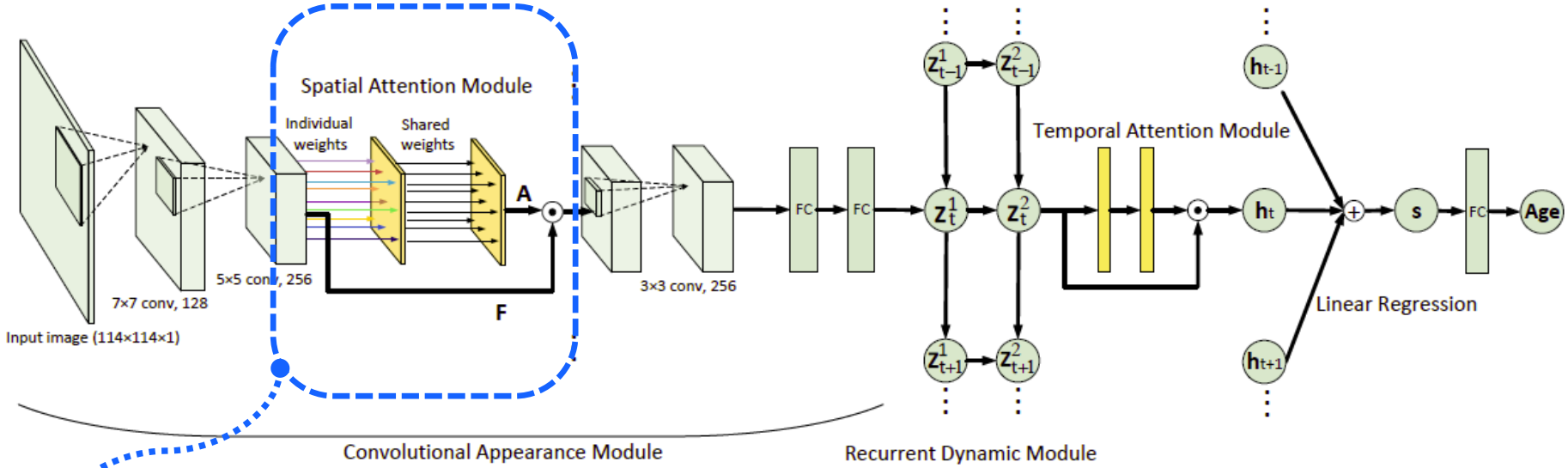
Kan, Shan, Chen. Stacked Progressive Auto-Encoders (SPAE) for Face Recognition Across Poses.
IEEE CVPR 2014.

Age Estimation



Pei, Dibeklioğlu, Baltrusaitis, Tax. Attended End-to-end Architecture for Age Estimation from Facial Expression Videos. IEEE Transactions on Image Processing, 2020.

Age Estimation

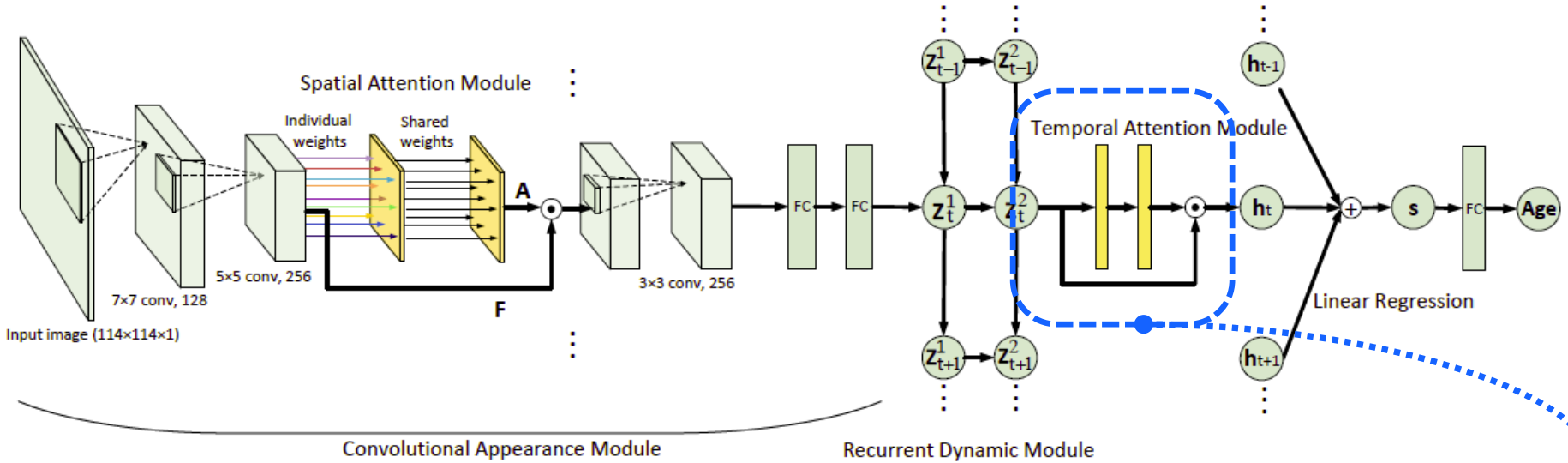


$$\mathbf{A}_{ij} = \sigma(\mathbf{u}^\top \tanh(\mathbf{W}_{ij}\mathbf{F}_{ij} + \mathbf{b}_{ij}) + c)$$

$$\mathbf{I} = \mathbf{F} \odot \mathbf{A}$$

Pei, Dibeklioğlu, Baltrusaitis, Tax. Attended End-to-end Architecture for Age Estimation from Facial Expression Videos.
IEEE Transactions on Image Processing, 2020.

Age Estimation

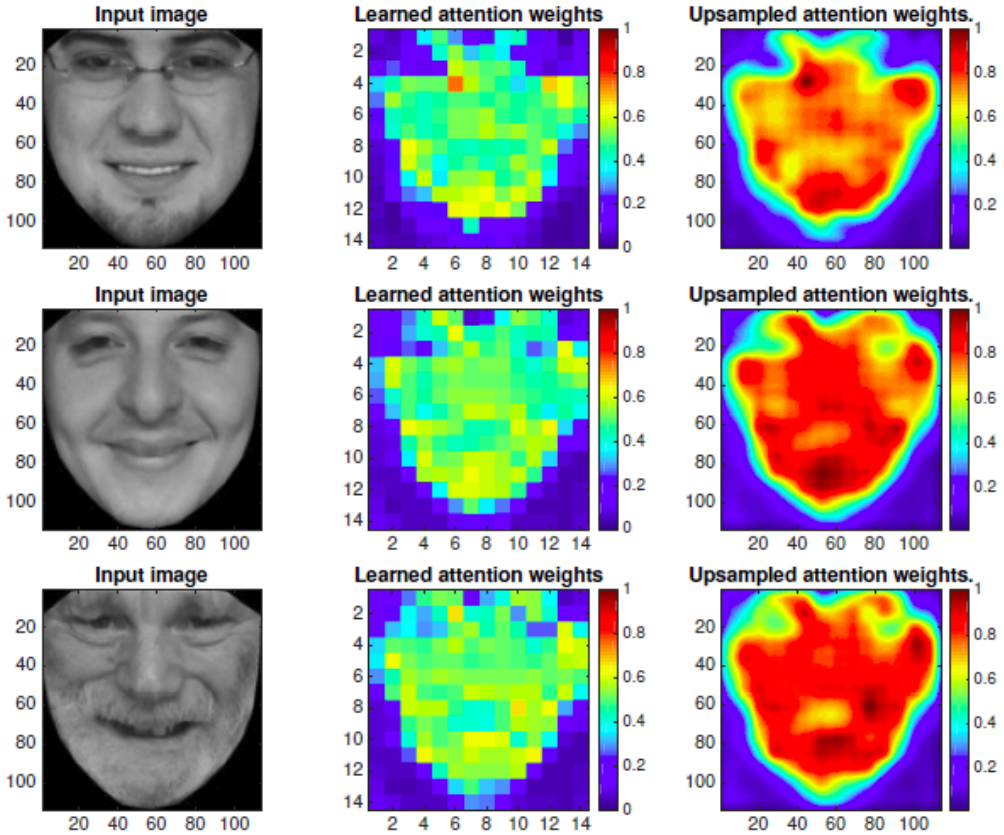


$$e_t = \sigma(\mathbf{v}^\top \tanh(\mathbf{M}\mathbf{z}_t + \mathbf{b}) + c)$$

$$o_t = \frac{e_t}{\sum_{t'=1}^{T} e_{t'}} \qquad h_t = o_t\, z_t$$

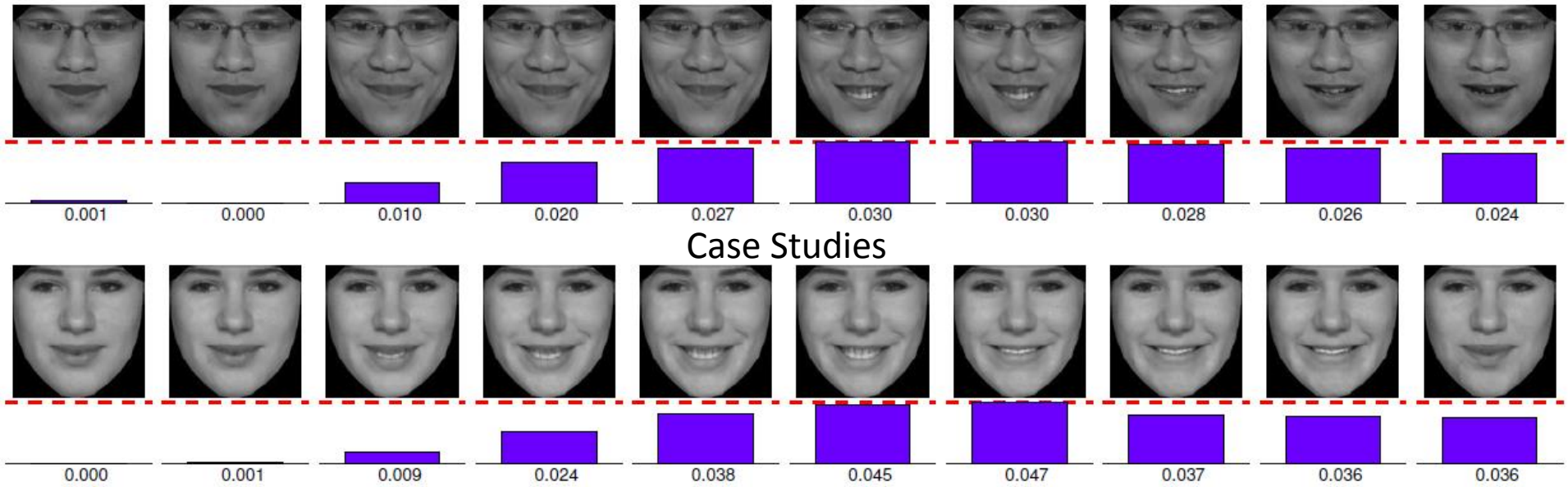
Pei, Dibeklioğlu, Baltrusaitis, Tax. Attended End-to-end Architecture for Age Estimation from Facial Expression Videos. IEEE Transactions on Image Processing, 2020.

Age Estimation: Visualization of Spatial Attention Weights



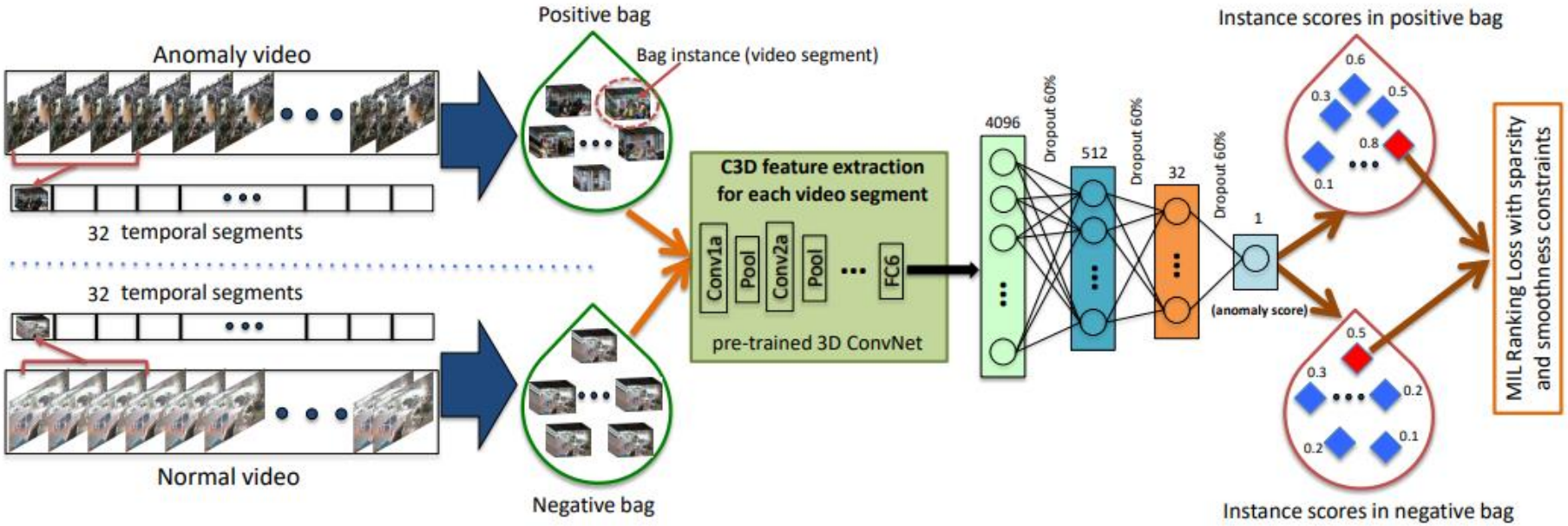
Pei, Dibeklioğlu, Baltrusaitis, Tax. Attended End-to-end Architecture for Age Estimation from Facial Expression Videos. IEEE Transactions on Image Processing, 2020.

Age Estimation: Visualization of Temporal Attention Weights



Pei, Dibeklioğlu, Baltrusaitis, Tax. Attended End-to-end Architecture for Age Estimation from Facial Expression Videos. IEEE Transactions on Image Processing, 2020.

Anomaly Detection



Sultani, Chen, Shah. Real-world Anomaly Detection in Surveillance Videos.
IEEE CVPR, 2018.

Anomaly Detection

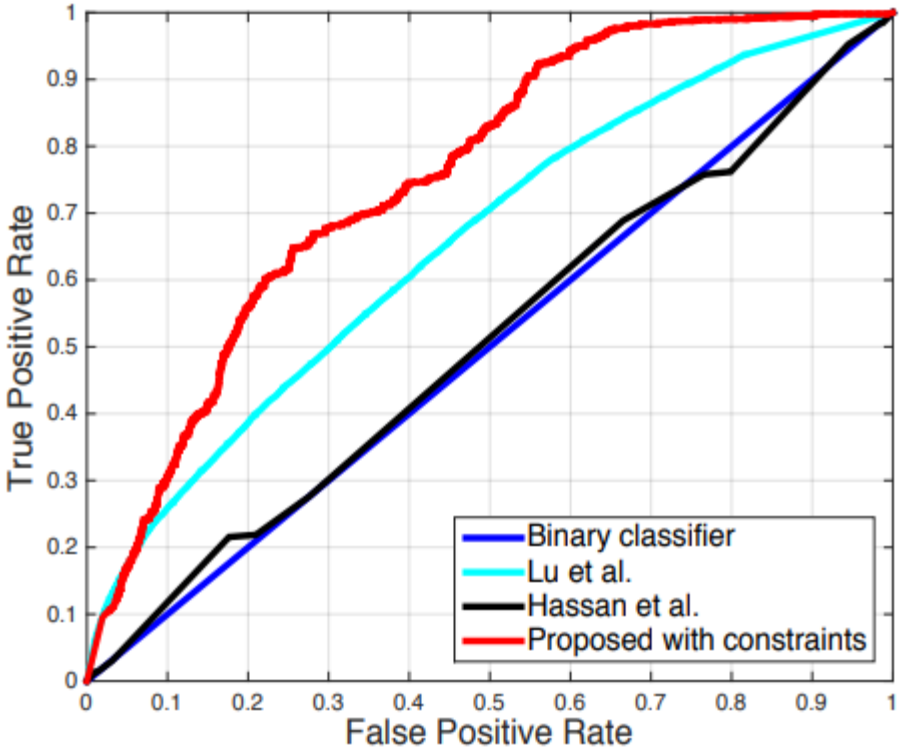
$$l(\mathcal{B}_a, \mathcal{B}_n) = \max(0, 1 - \max_{i \in \mathcal{B}_a} f(\mathcal{V}_a^i) + \max_{i \in \mathcal{B}_n} f(\mathcal{V}_n^i))$$

$$+\lambda_1 \overbrace{\sum_i^{(n-1)} (f(\mathcal{V}_a^i) - f(\mathcal{V}_a^{i+1}))^2}^{①} + \lambda_2 \overbrace{\sum_i^{n} f(\mathcal{V}_a^i)}^{②},$$

where ① indicates the temporal smoothness term and ② represents the sparsity term.

Sultani, Chen, Shah. Real-world Anomaly Detection in Surveillance Videos.
IEEE CVPR, 2018.

Anomaly Detection



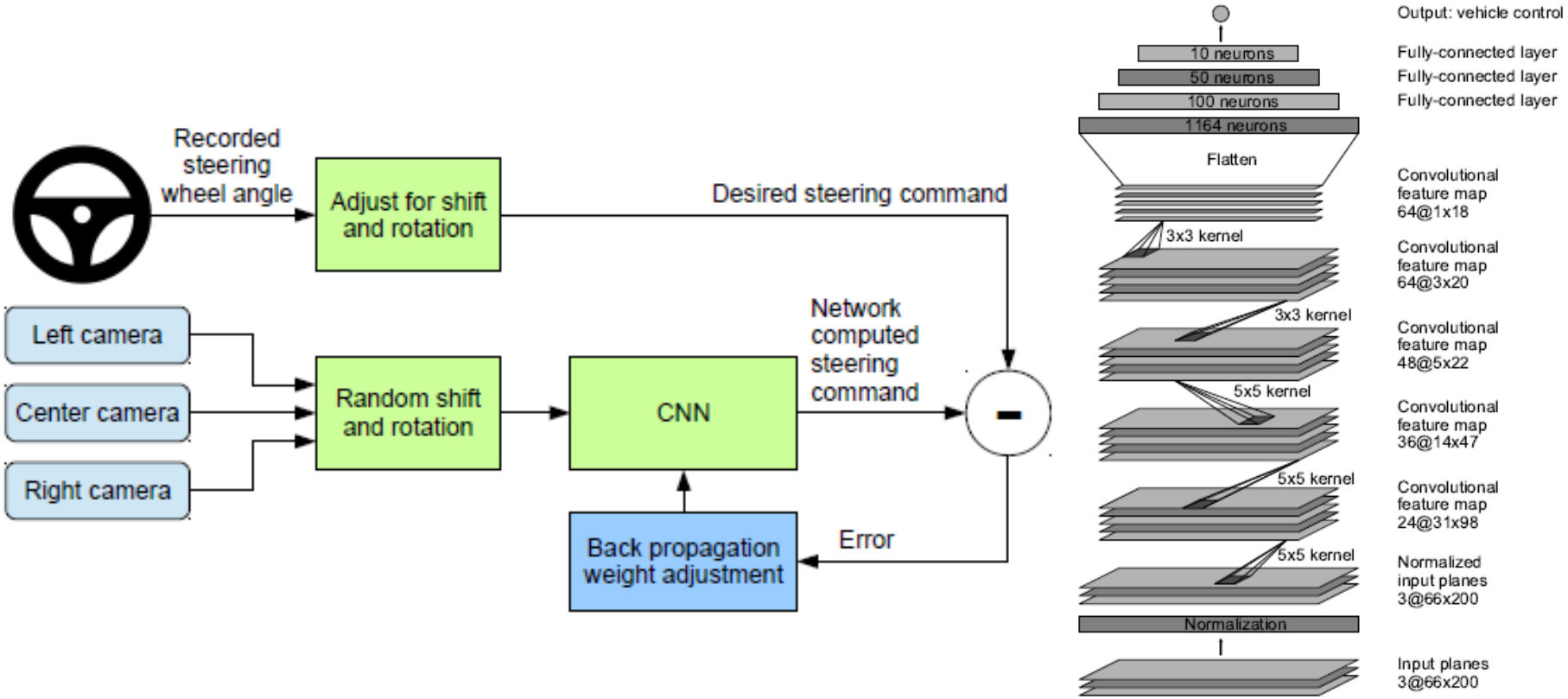
Method	AUC
Binary classifier	50.0
Hasan *et al.* [18]	50.6
Lu *et al.* [28]	65.51
Proposed w/o constraints	74.44
Proposed w constraints	**75.41**

Sultani, Chen, Shah. Real-world Anomaly Detection in Surveillance Videos.
IEEE CVPR, 2018.

Anomaly Detection: Examples of different anomalies

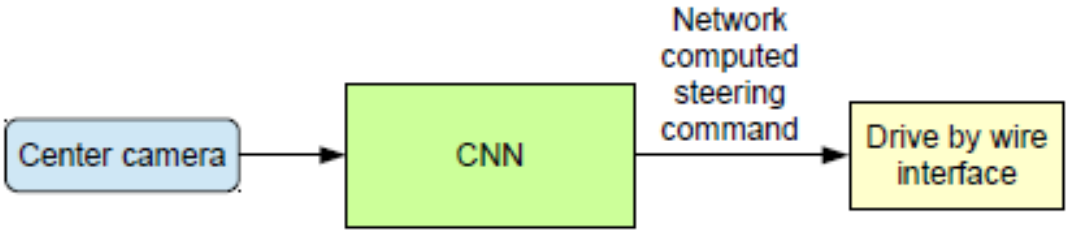
Sultani, Chen, Shah. Real-world Anomaly Detection in Surveillance Videos.
IEEE CVPR, 2018.

Autonomous Driving



Bojarski et al. End to end learning for self-driving cars. arXiv, 2016.

Autonomous Driving



Bojarski et al. End to end learning for self-driving cars. arXiv, 2016.

DAVE 2 Driving a Lincoln

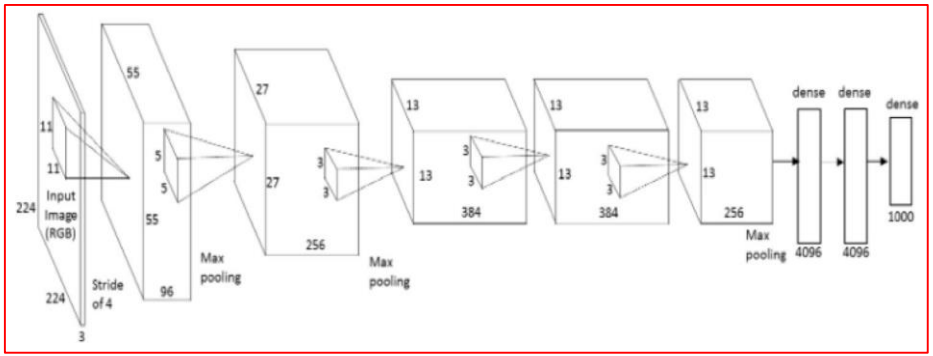
- A convolutional neural network
- Trained by human drivers
- Learns perception, path planning, and control
  "pixel in, action out"
- Front-facing camera is the only sensor

https://youtu.be/NJU9ULQUwng

NeuralStyle

[A Neural Algorithm of Artistic Style by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge, 2015]

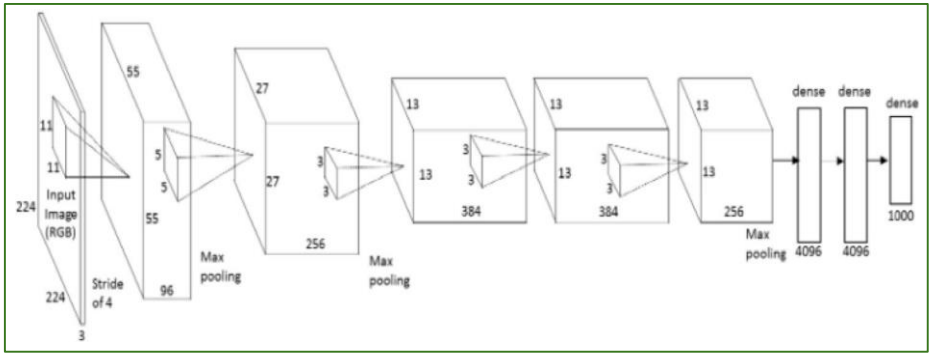
Step 1: Extract **content targets**
(ConvNet activations of all layers for the given content image)



content activations

e.g.
at CONV5_1 layer we would have a [14x14x512] array of target activations

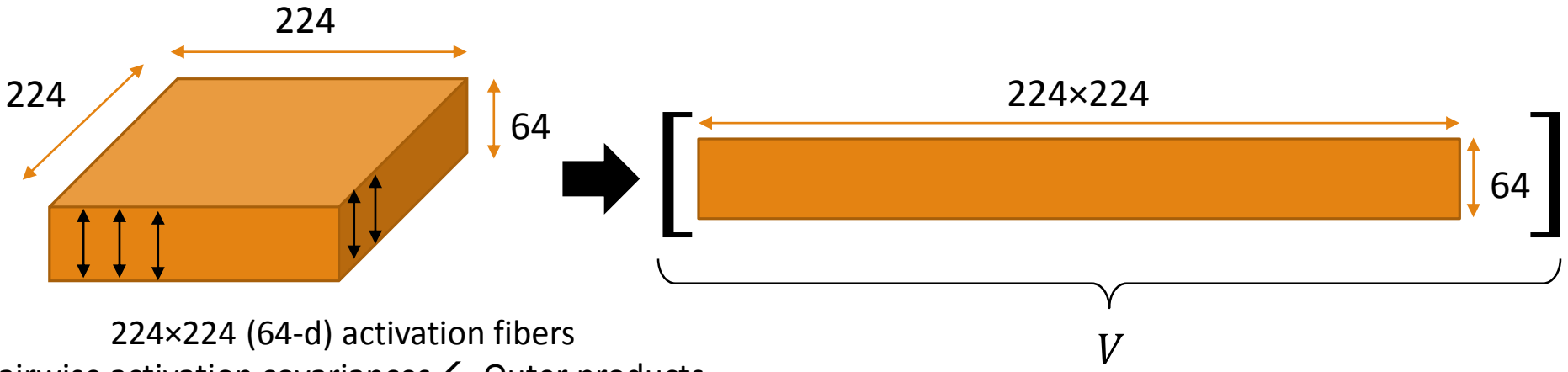
Step 2: Extract **style targets**
(Gram matrices of ConvNet activations of all layers for the given style image)



style Gram matrices

Gram matrix: $G = VV^T$

e.g.
at CONV1 layer (with [224x224x64] activations) would give a [64x64] Gram matrix of all pairwise activation covariances (summed across spatial locations)



224×224 (64-d) activation fibers

Pairwise activation covariances ← Outer products

Gram matrix: $G = VV^T$

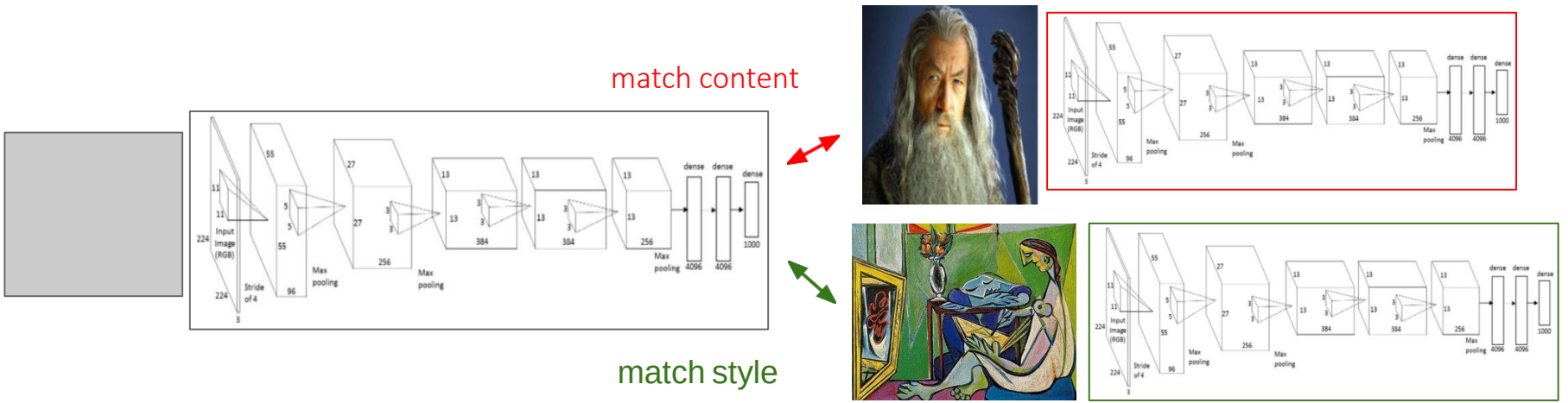
e.g.
at CONV1 layer (with [224x224x64] activations) would give a [64x64] Gram matrix of all pairwise activation covariances (summed across spatial locations)

Step 3: Optimize over image to have:

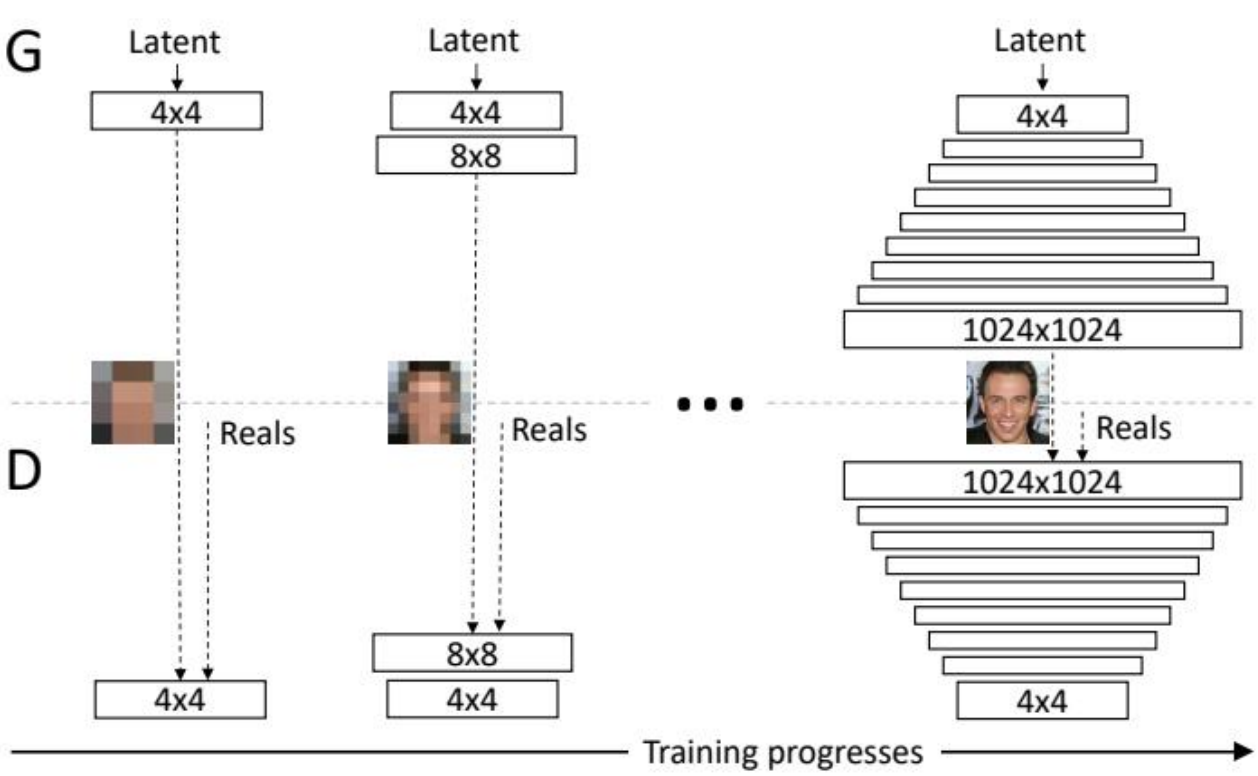
- The **content** of the content image (activations match content)
- The **style** of the style image (Gram matrices of activations match style)

$$\mathcal{L}_{total}(p, a, x) = \alpha \mathcal{L}_{content}(p, x) + \beta \mathcal{L}_{style}(a, x)$$

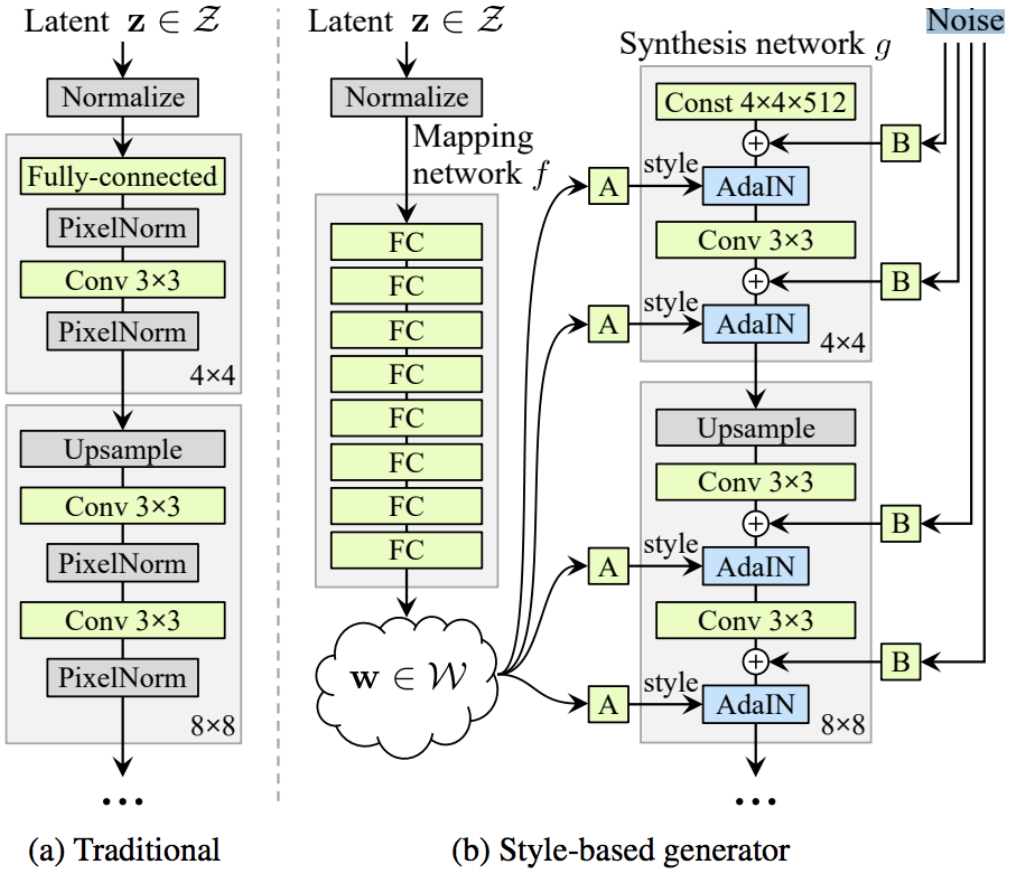
(+Total Variation regularization (maybe))



Progressive GAN, Karras 2018

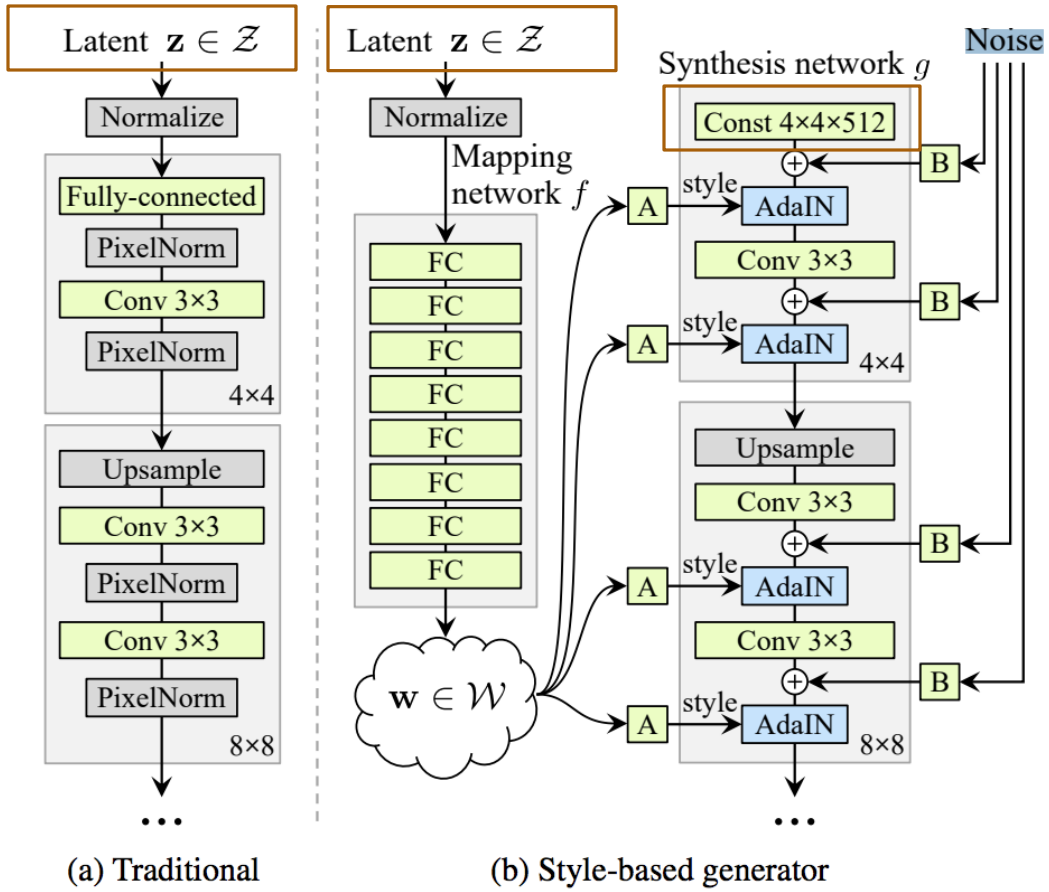


StyleGAN, Karras 2018



A Style-Based Generator Architecture for Generative Adversarial Networks, 2018.

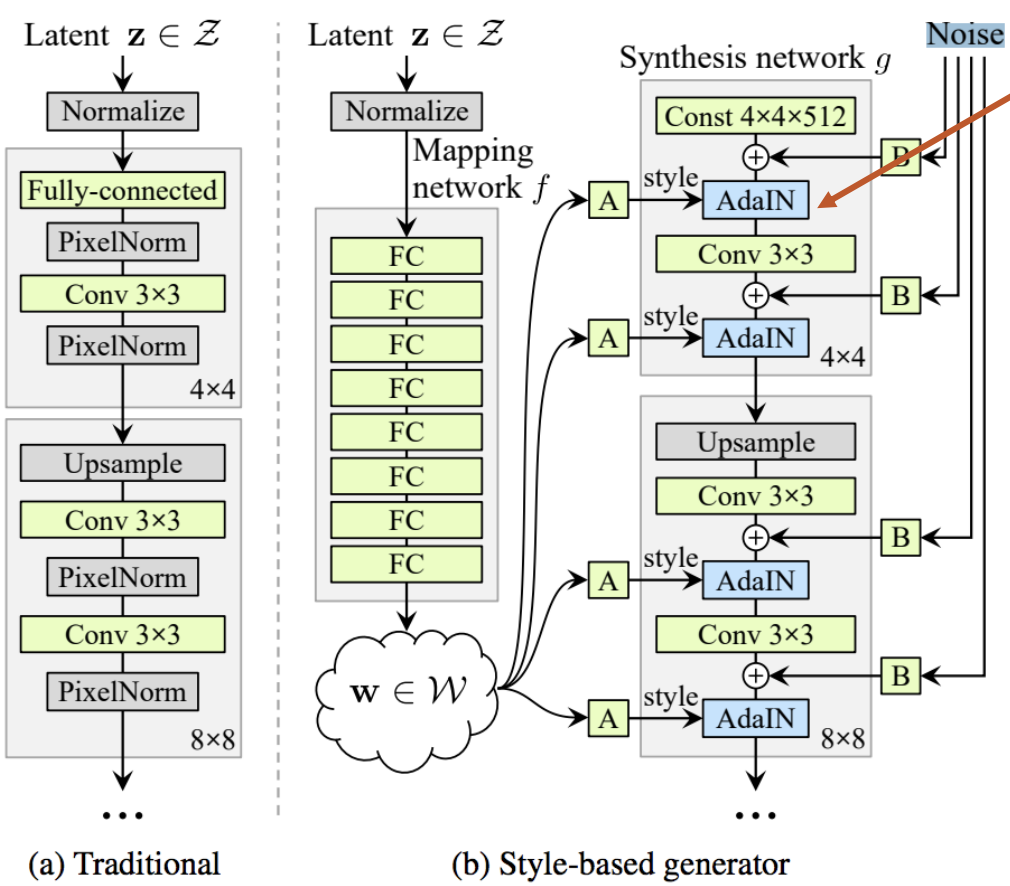
StyleGAN, Karras 2018



A Style-Based Generator Architecture for Generative Adversarial Networks, 2018.

StyleGAN, Karras 2018

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$



Adaptive Instance Normalization (AdaIN)

A Style-Based Generator Architecture for Generative Adversarial Networks, 2018.

StyleGAN, Karras 2018

https://youtu.be/kSLJriaOumA