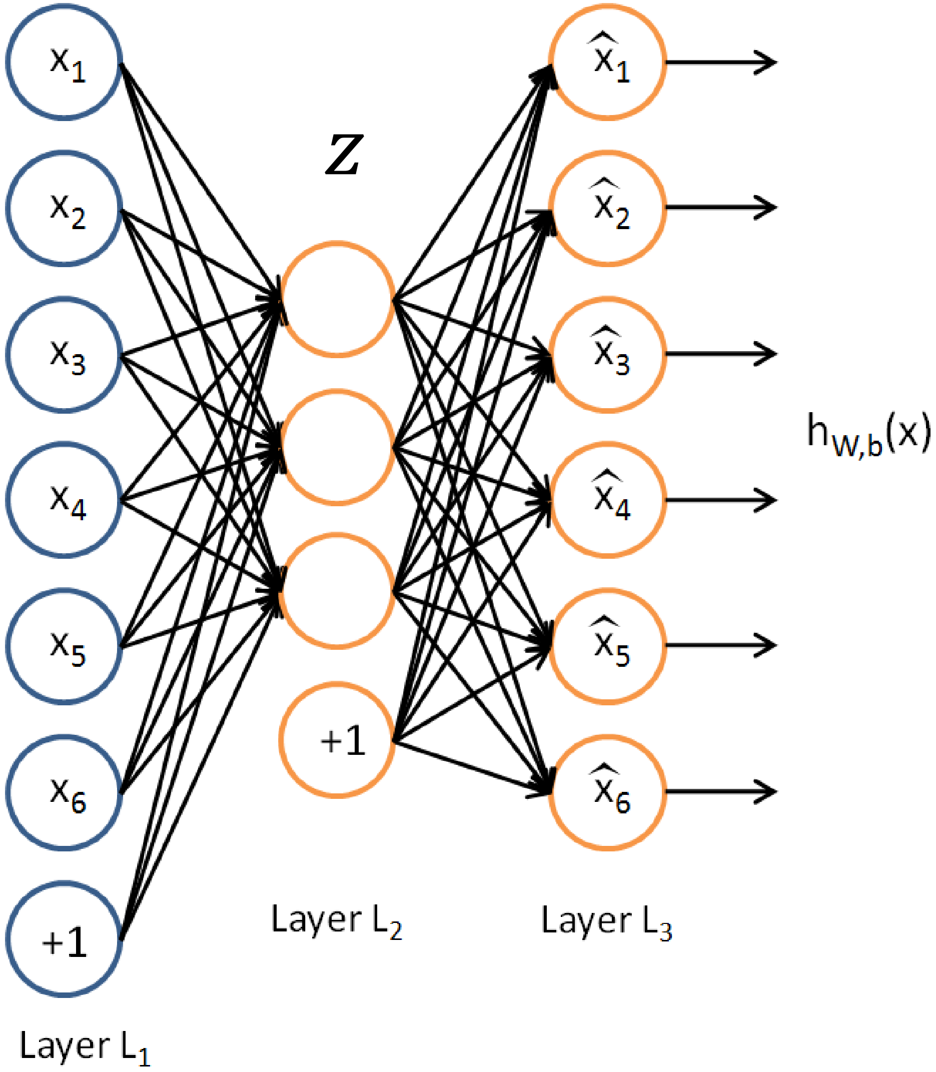
Encoder-decoder model

Biplab Banerjee

Deep CNN based image segmentation

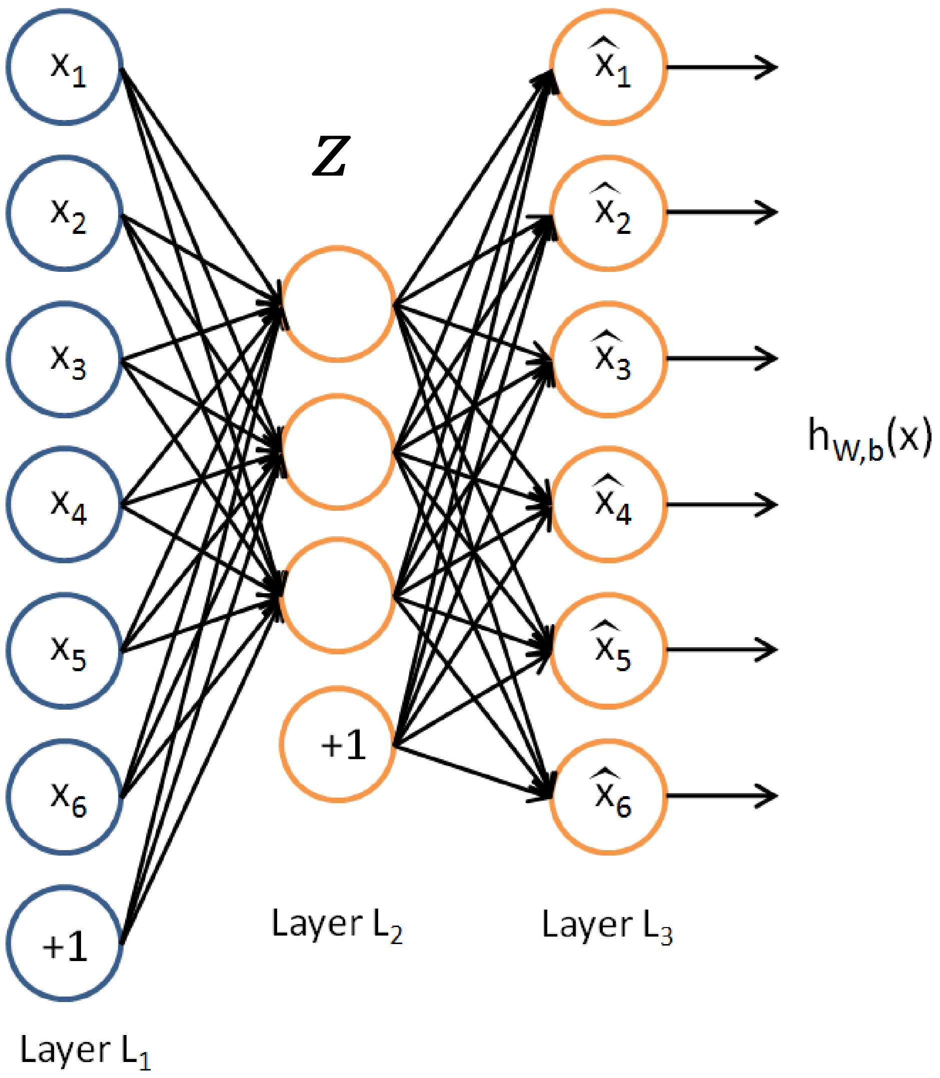
- Auto-encoder
- Regularized auto-encoder
- Class-encoder
- Image segmentation

Traditional Autoencoder



Traditional Autoencoder

- Unlike the **PCA** now we can use activation functions to achieve non-linearity.
- It has been shown that an AE without activation functions achieves the **PCA** capacity. (later)



Uses

- The autoencoder idea was a part of NN history for decades (LeCun et al, 1987).

- Traditionally an autoencoder is used for dimensionality reduction and feature learning.
- Representation learning

Simple Idea

- Given data x (no labels) we would like to learn the functions f (encoder) and g (decoder) where:

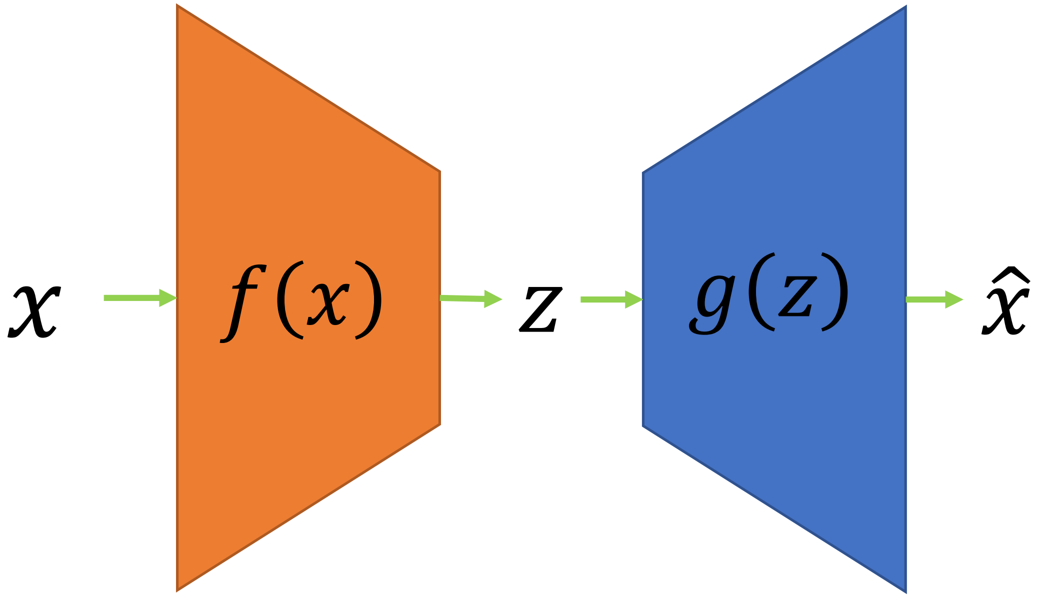
$$f(x) = s(wx + b) = z$$

and

$$g(z) = s(w'z + b') = \hat{x}$$

$$\text{s.t } h(x) = g\big(f(x)\big) = \hat{x}$$

where h is an **approximation** of the identity function.



(z is some **latent** representation or **code** and s is a non-linearity such as the sigmoid)

($\hat{x}$ is x's reconstruction)

Training the AE

Using **Gradient Descent** we can simply train the model as any other FC NN with:

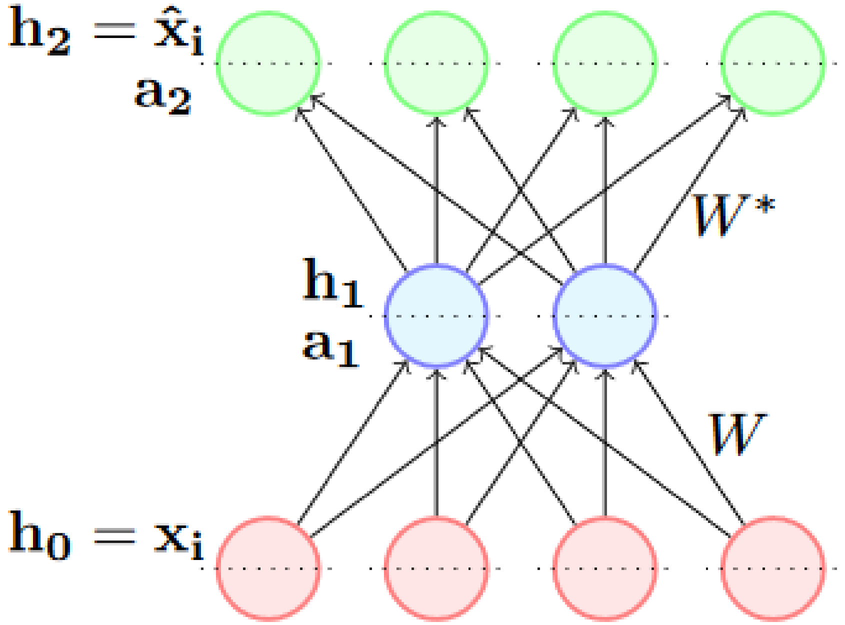
- Traditionally with *squared error* loss function

$$L(x, \hat{x}) = ||x - \hat{x}||^2$$

- If our input is interpreted as bit vectors or vectors of bit probabilities the *cross entropy* can be used

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

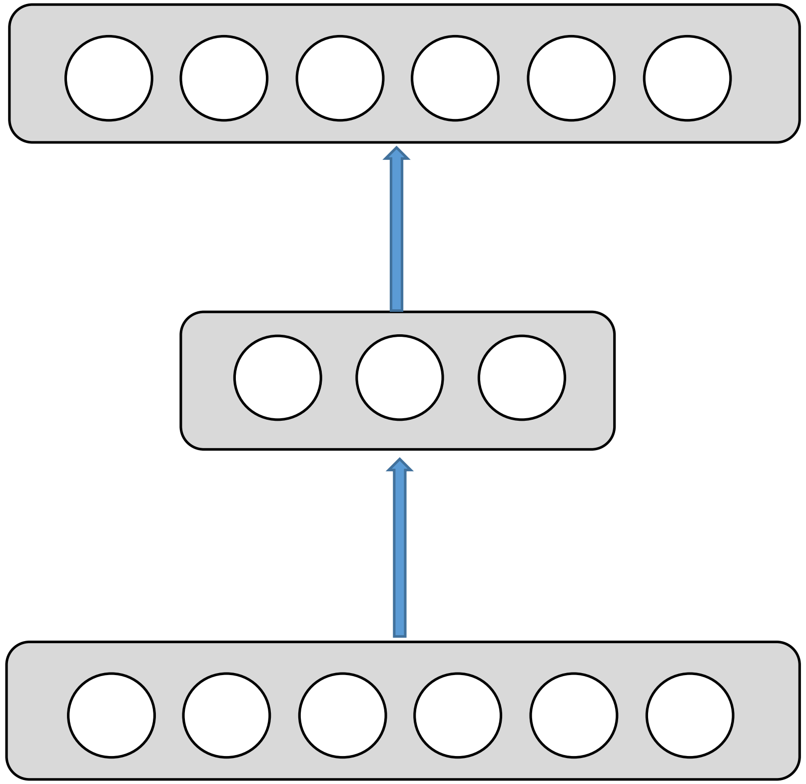
$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$

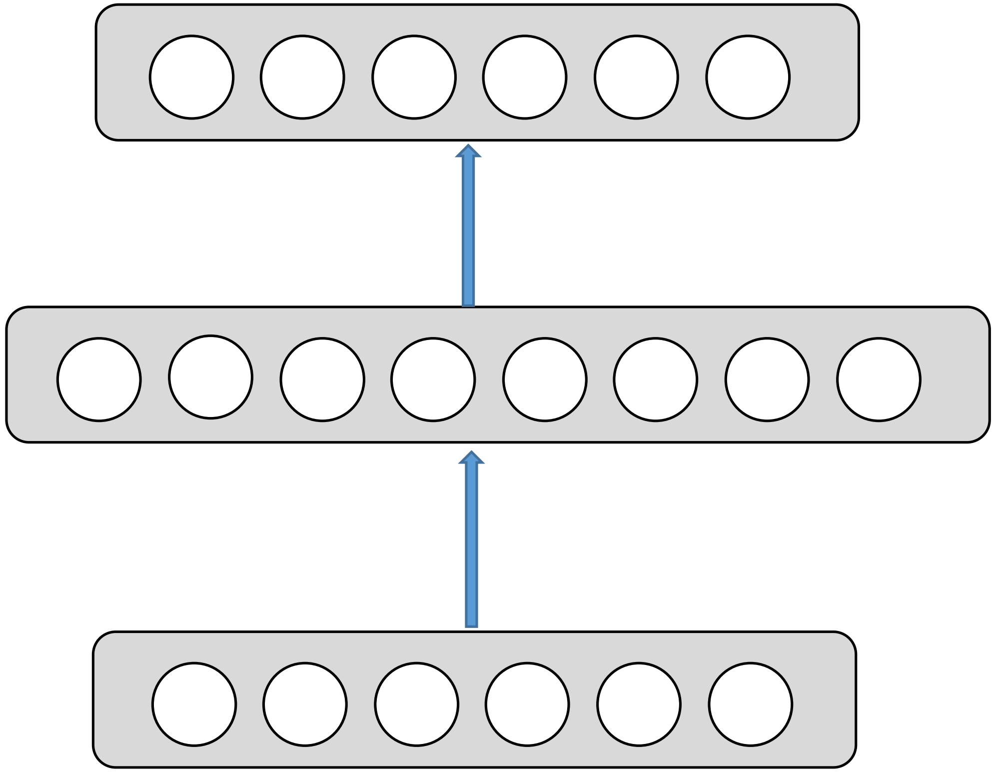


- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$
- $$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$$

Undercomplete AE VS overcomplete AE

We distinguish between two types of AE structures:

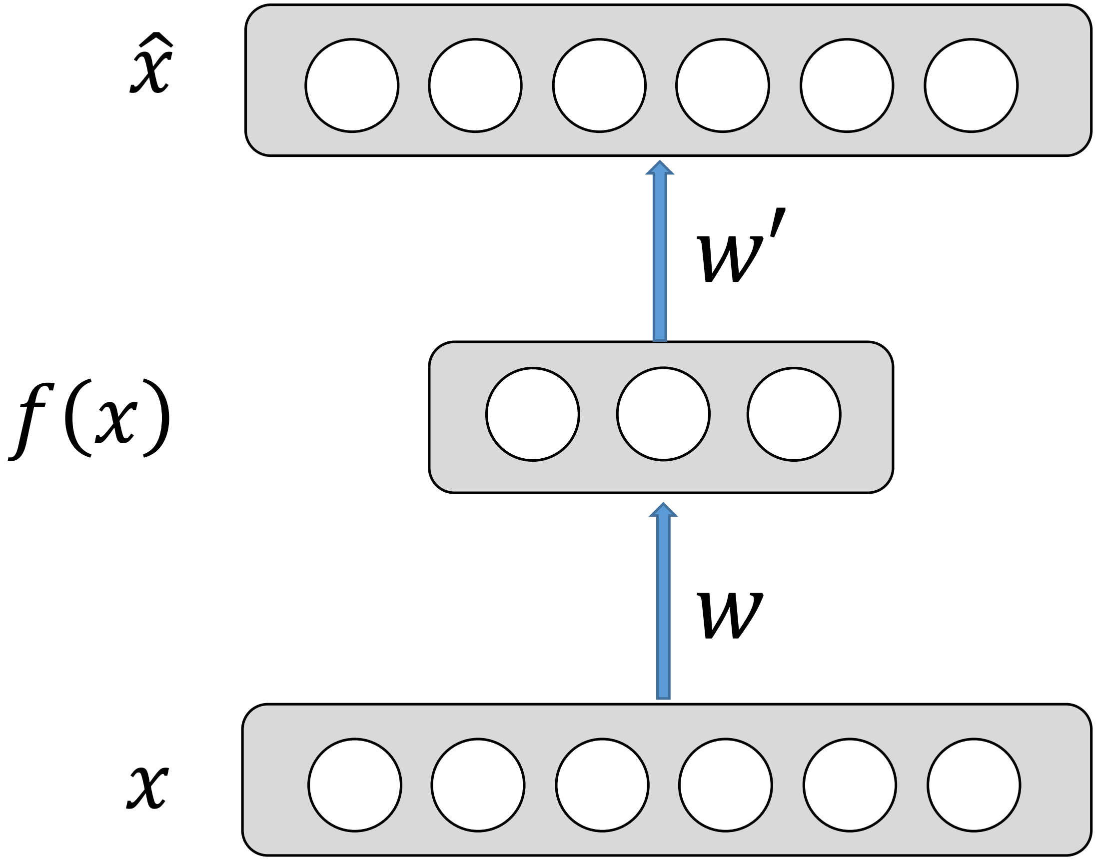




Undercomplete AE

- Hidden layer is **Undercomplete** if smaller than the input layer
  - ❑Compresses the input
  - ❑Compresses well only for the training dist.

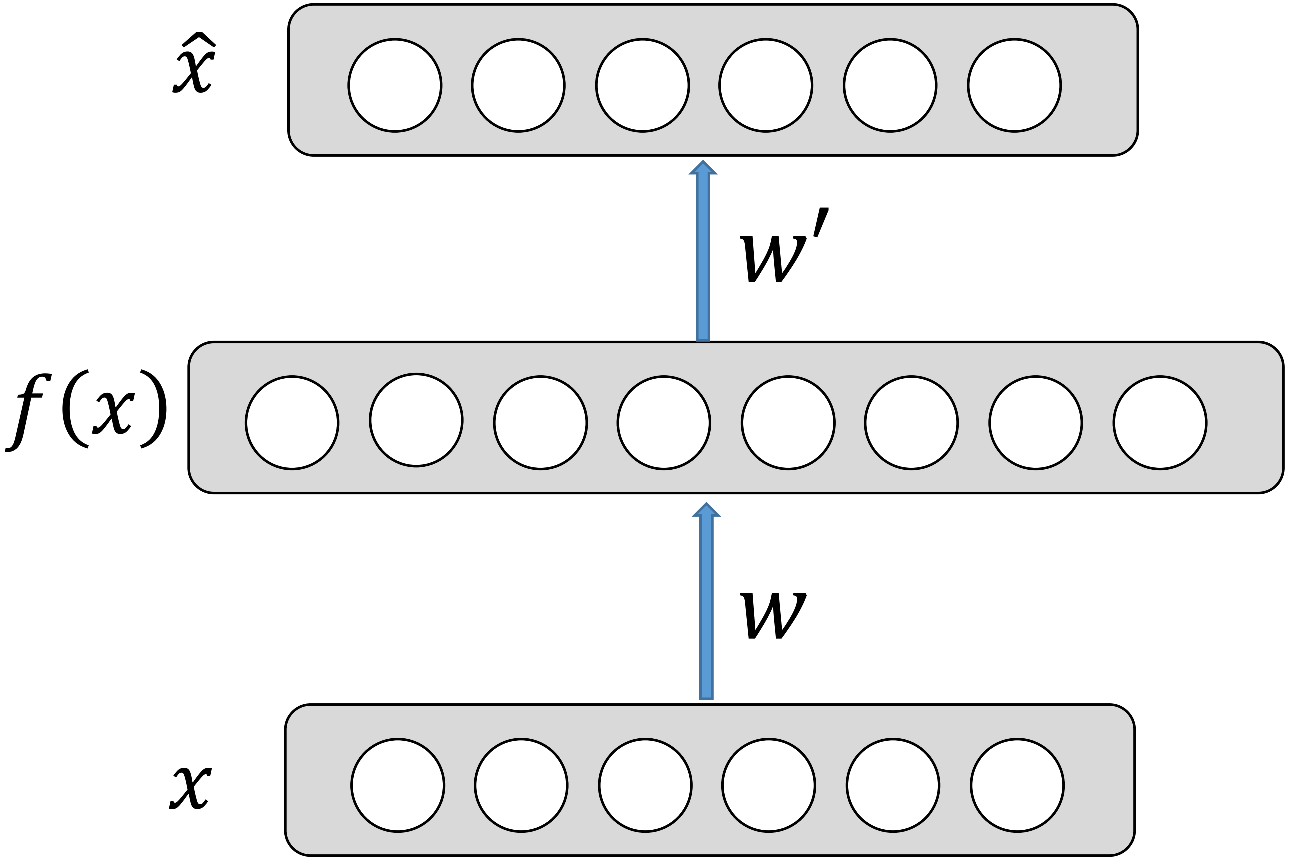
- Hidden nodes will be
  - ❑Good features for the training distribution.
  - ❑Bad for other types on input



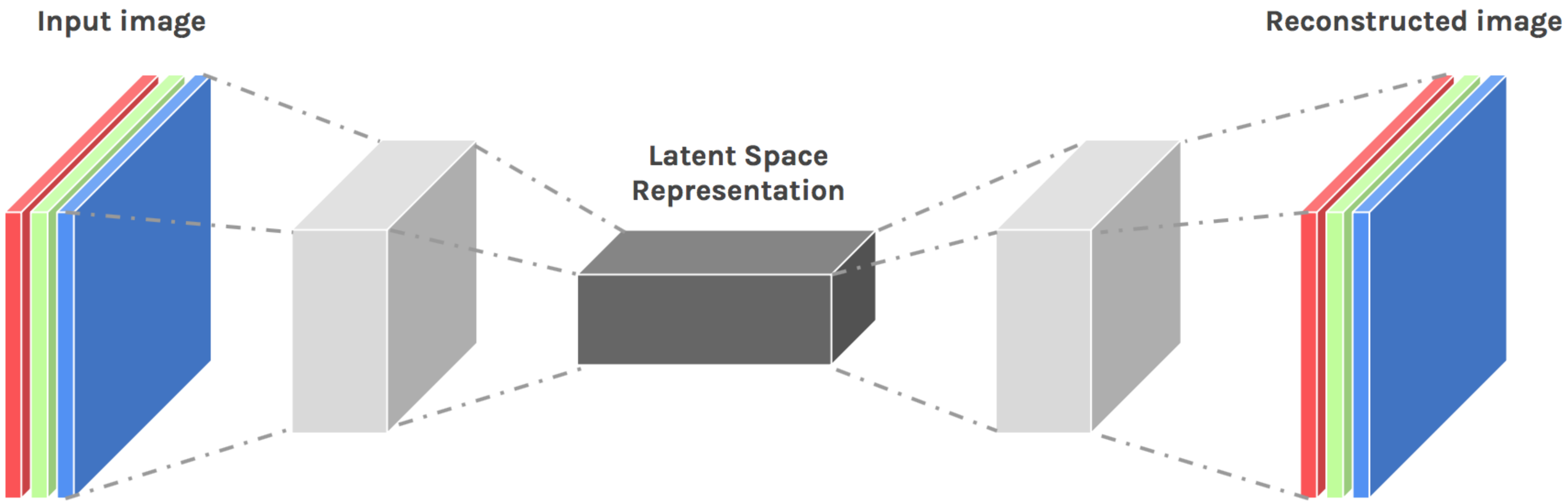
Overcomplete AE

- Hidden layer is **Overcomplete** if greater than the input layer
  - ❑ No compression in hidden layer.
  - ❑ Each hidden unit could copy a different input component.

- No guarantee that the hidden units will extract meaningful structure.

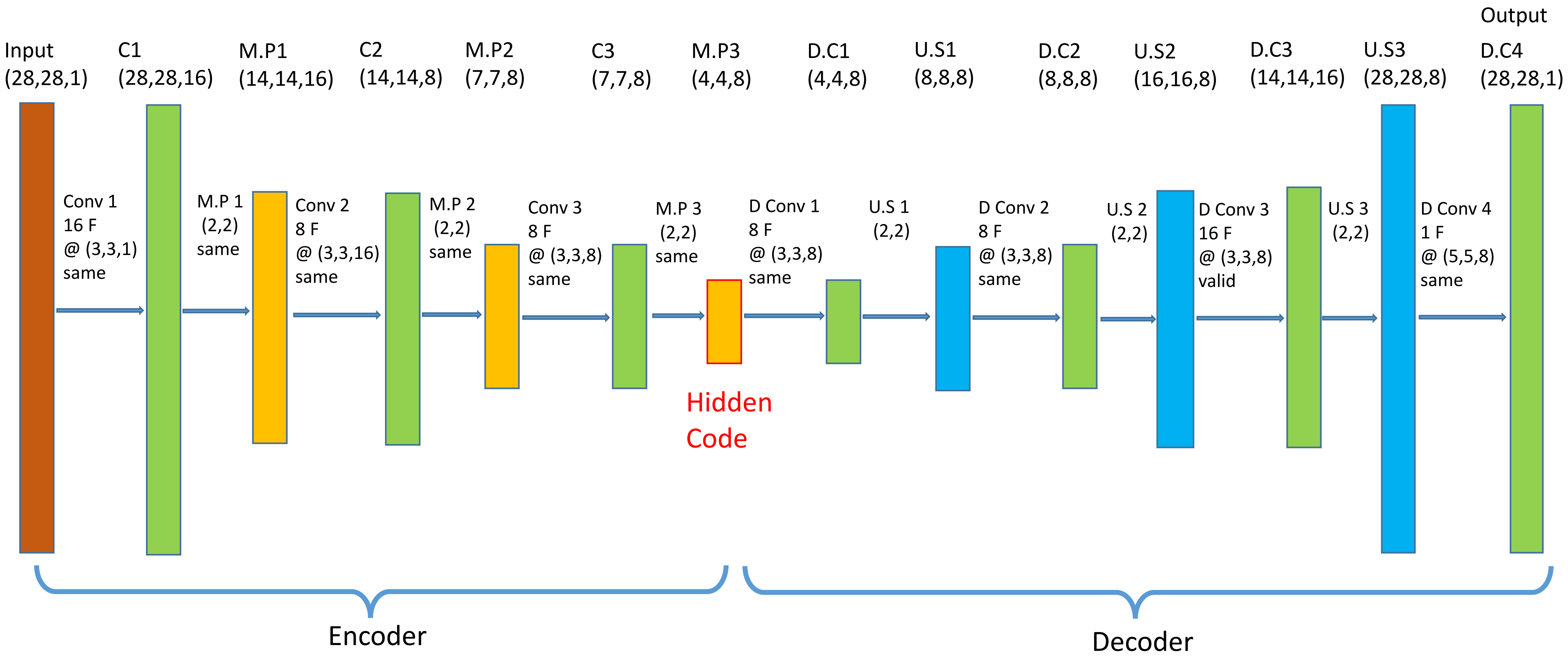


Convolutional AE

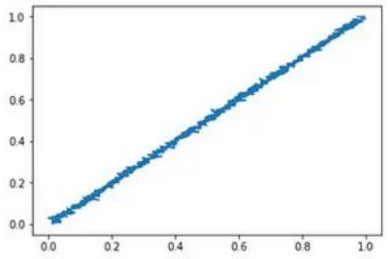
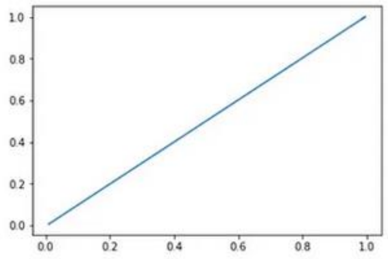
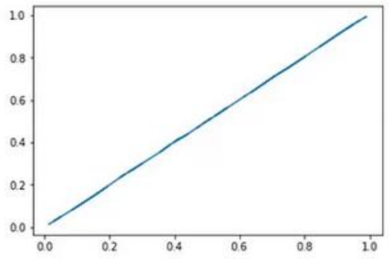
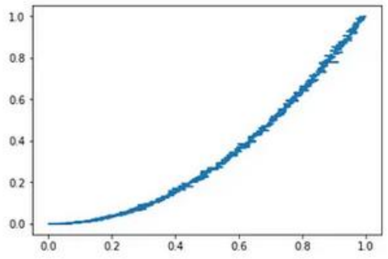
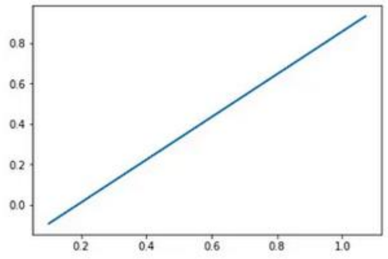
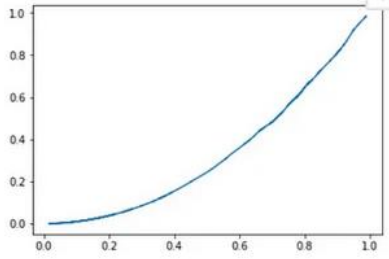
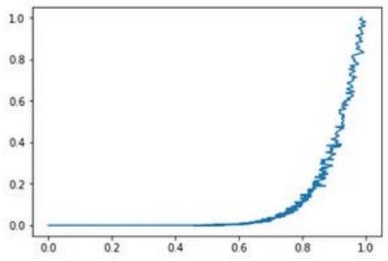
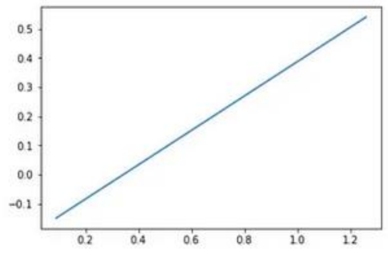
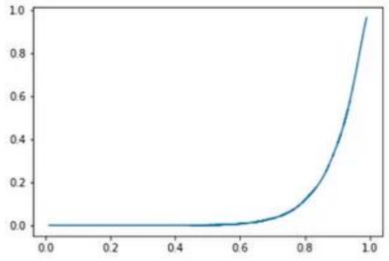


* Input values are normalized
* All of the conv layers activation functions are relu except for the last conv which is sigm

Convolutional AE



AE vs PCA

Function	Feature Space	PCA Reconstruction	Auto Encoder Reconstruction
$y=mx+c$			
$y=mx^2+c$			
$y=mx^8+c$			

Differences between AE and PCA

1.Linearity vs. Non-Linearity: PCA is a linear technique, whereas Autoencoders can learn non-linear representations of data.

2.Interpretability vs. Deep Learning: PCA’s principal components are interpretable as they are linear combinations of the original variables. Autoencoder results can be less interpretable due to the complexity of the neural network.

3.Supervision: PCA is an unsupervised technique, while Autoencoders can be used in both unsupervised and supervised settings.

4.Flexibility: Autoencoders offer greater flexibility in capturing complex, non-linear relationships in data compared to the linear combinations of PCA.

Regularization

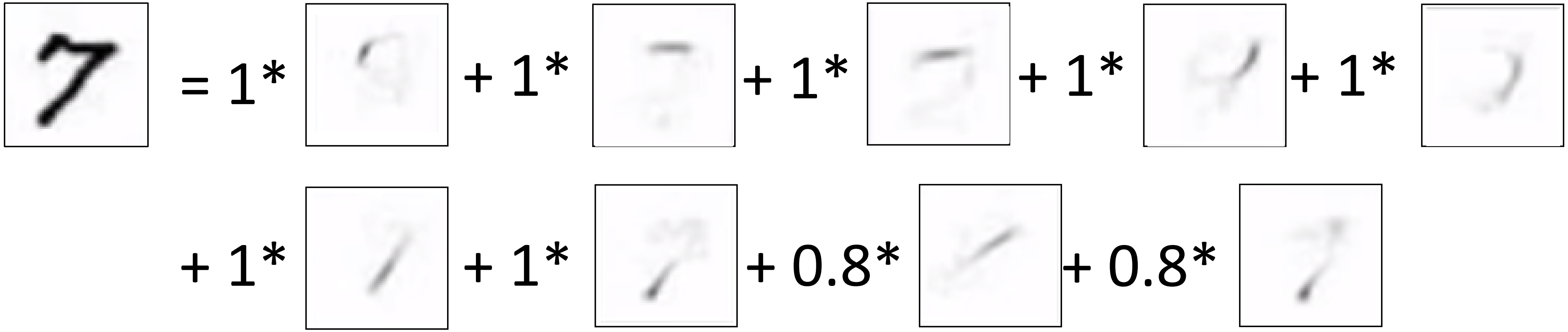
Motivation:

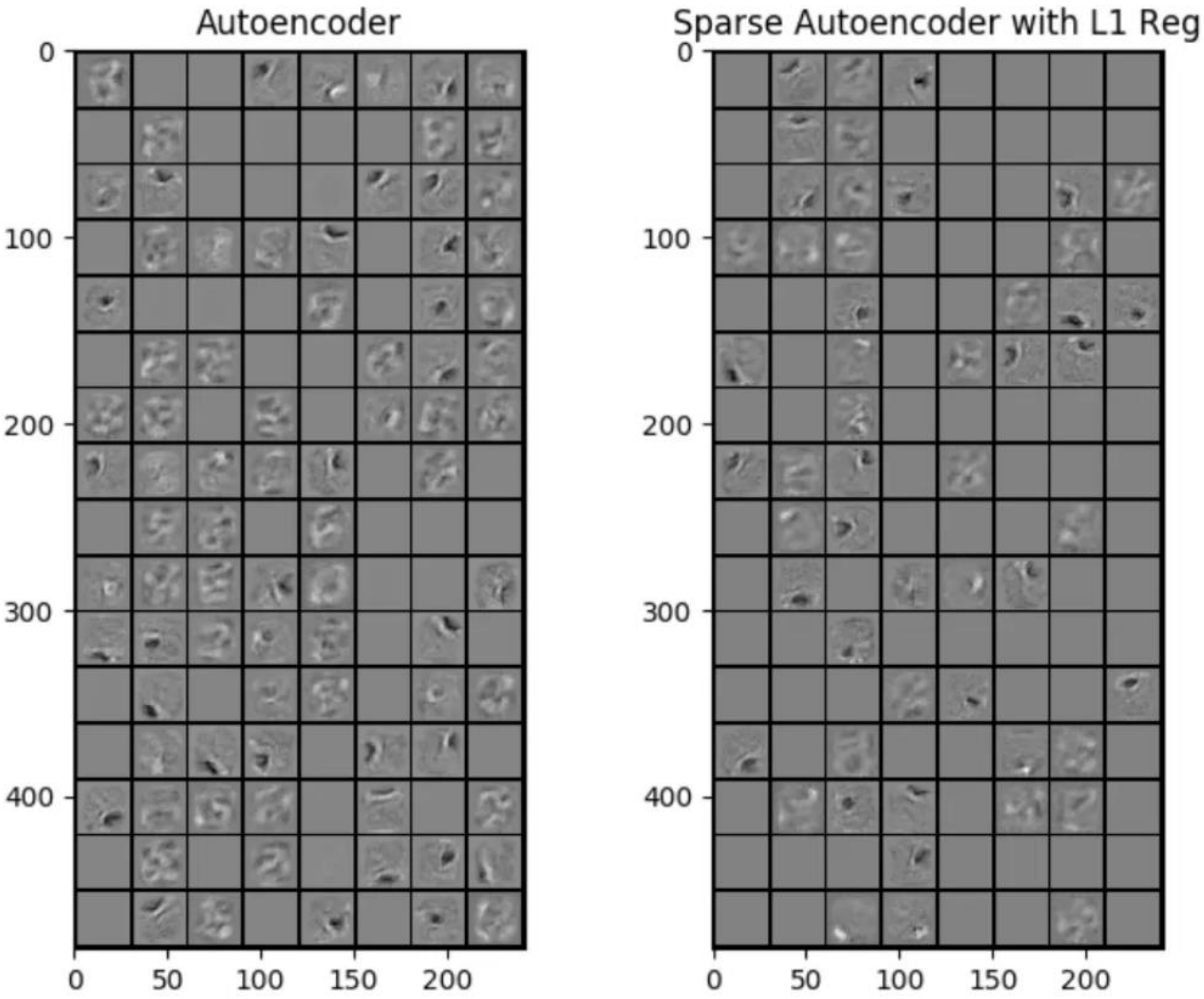
- We would like to learn meaningful features **without** altering the code’s dimensions (Overcomplete or Undercomplete).

The solution: imposing other constraints on the network.

Sparsely Regulated Autoencoders

- We want our learned features to be as **sparse** as possible.
- With sparse features we can generalize better.





Autoencoder
Sparse Autoencoder with L1 Reg
0
100
200
300
400
0
50
100
150
200

Sparsely Regulated Autoencoders

Further let,

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j\ \left(x^{(i)}\right)\right]$$

be the average activation of hidden unit j (over the training set).

Thus we would like to force the constraint:

$$\hat{\rho}_j = \rho$$

where ρ is a “sparsity parameter”, typically small. In other words, we want the average activation of each neuron j to be close to ρ.

Sparsely Regulated Autoencoders

- We need to penalize $\hat{\rho}_j$ for deviating from ρ.
- Many choices of the penalty term will give reasonable results.

- For example: $$\sum_{j=1}^{Bn} KL\left(\rho|\hat{\rho}_j\right)$$

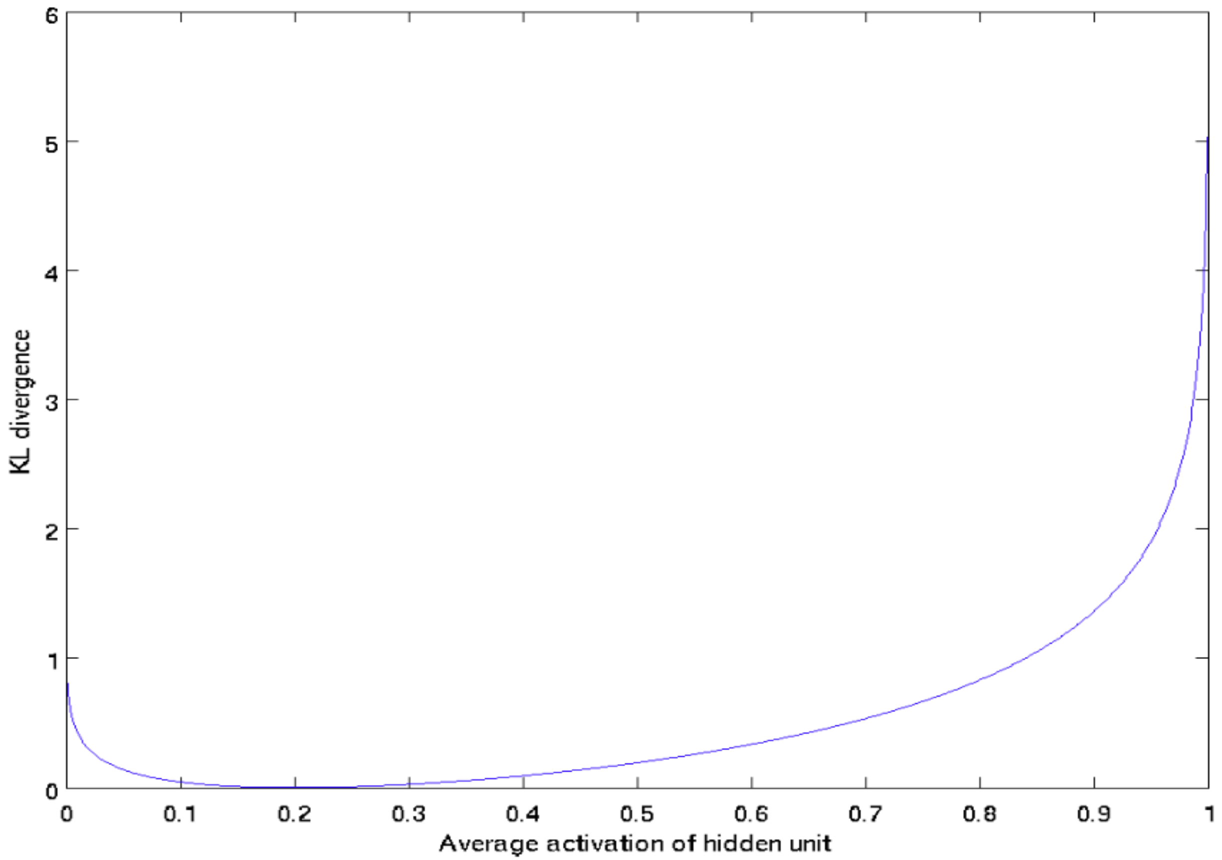
where $KL\left(\rho|\hat{\rho}_j\right)$ is a Kullback-Leibler divergence function.

Sparsely Regulated Autoencoders

- A reminder:
  - KL is a standard function for measuring how different two distributions are, which has the properties:

$$KL(\rho|\hat{\rho}_j) = 0 \text{ if } \hat{\rho}_j = \rho$$

otherwise it is increased monotonically.



$\rho = 0.2$

Sparsely Regulated Autoencoders

- Our overall cost functions is now:

$$J_S(W,b) = J(W,b) + \beta \sum_{j=1}^{Bn} KL\left(p|\hat{\rho}_j\right)$$

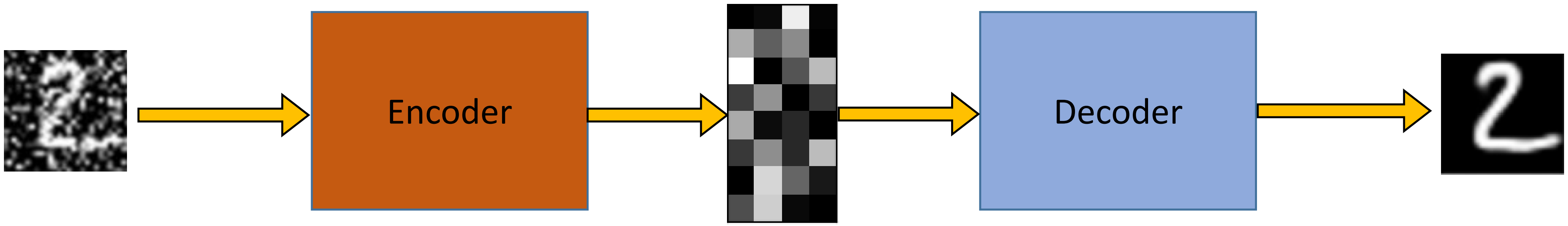
*Note: We need to know $\hat{\rho}_j$ before hand,
so we have to compute a forward pass on all the training set.

Denoising Autoencoders

Intuition:

- We still aim to encode the input and to NOT mimic the identity function.
- We try to undo the effect of *corruption* process stochastically applied to the input.

A more robust model



Noisy Input

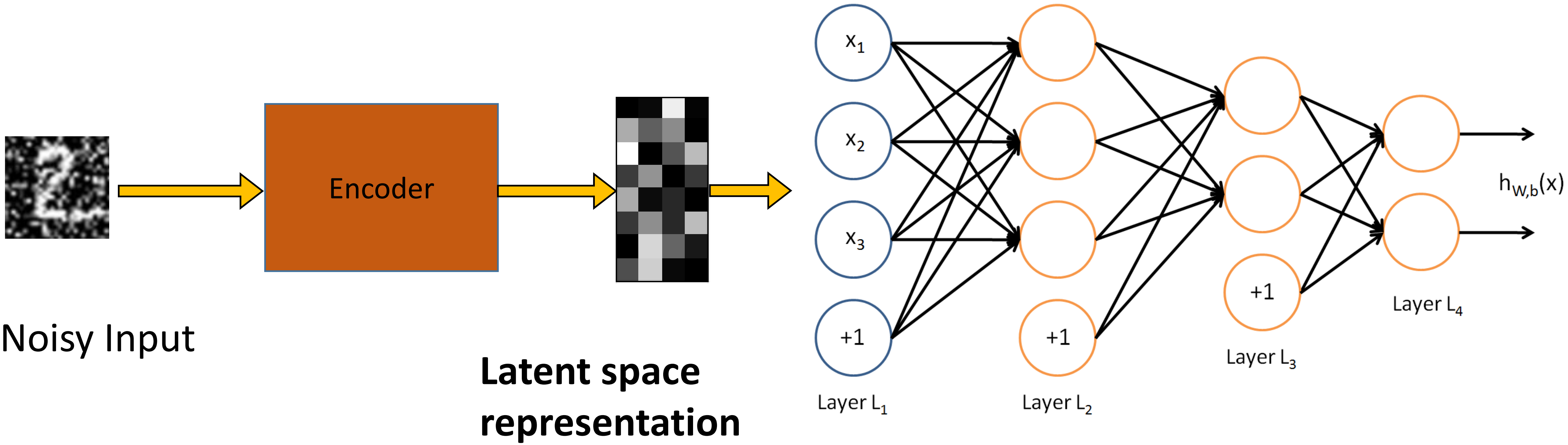
Latent space representation

Denoised Input

Denoising Autoencoders

Use Case:

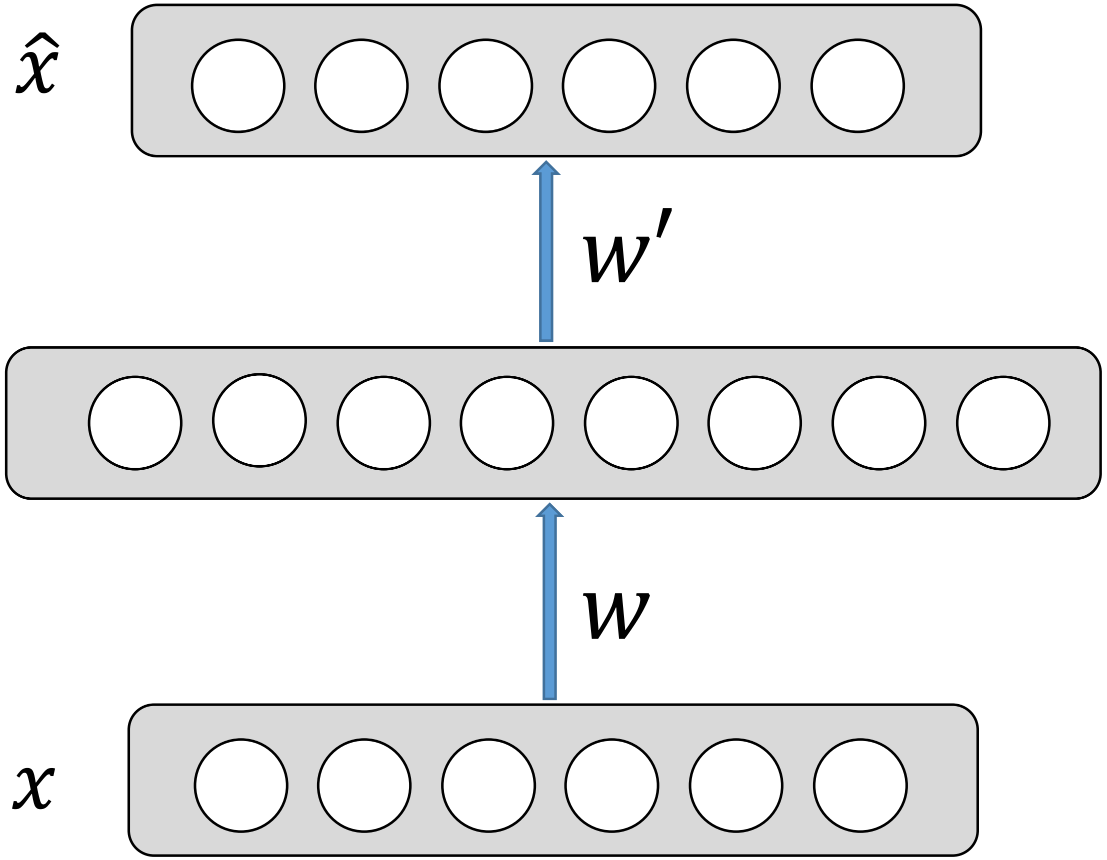
- Extract robust representation for a NN classifier.



Contractive autoencoders

- We wish to extract features that **only** reflect variations observed in the training set. We would like to be invariant to the other variations.

- Points close to each other in the input space should maintain that property in the latent space.



Contractive autoencoders

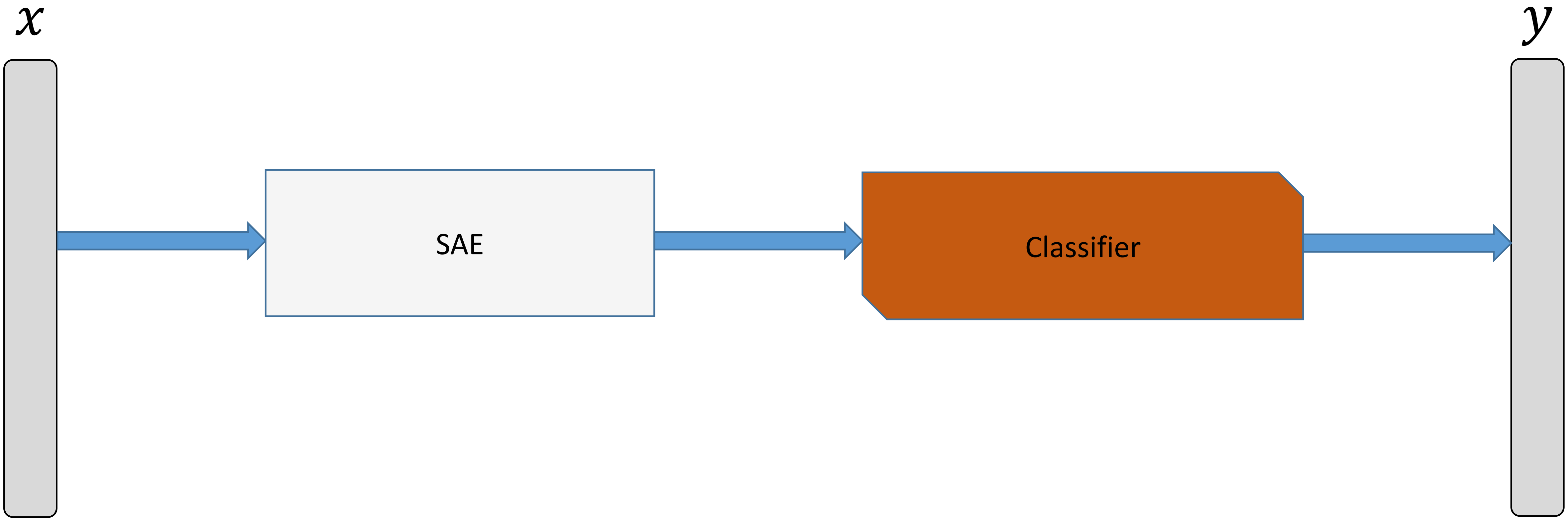
- Definitions and reminders:
- - Frobenius norm (L2): $\|A\|_F = \sqrt{\Sigma_{i,j} |a_{ij}|^2}$
- - Jacobian Matrix: $J_f(x) = \frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f(x)_1}{\partial x_1} & \cdots & \frac{\partial f(x)_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(x)_m}{\partial x_1} & \cdots & \frac{\partial f(x)m}{\partial x_n} \end{bmatrix}$

Contractive autoencoders

- Our new loss function would be:
- $$L^*(x) = L(x) + \lambda\Omega(x)$$
- where $\Omega(x) = \|J_f(x)\|_F^2$ or simply: $\sum_{i,j}\left(\frac{\partial f(x)_j}{\partial x_i}\right)^2$

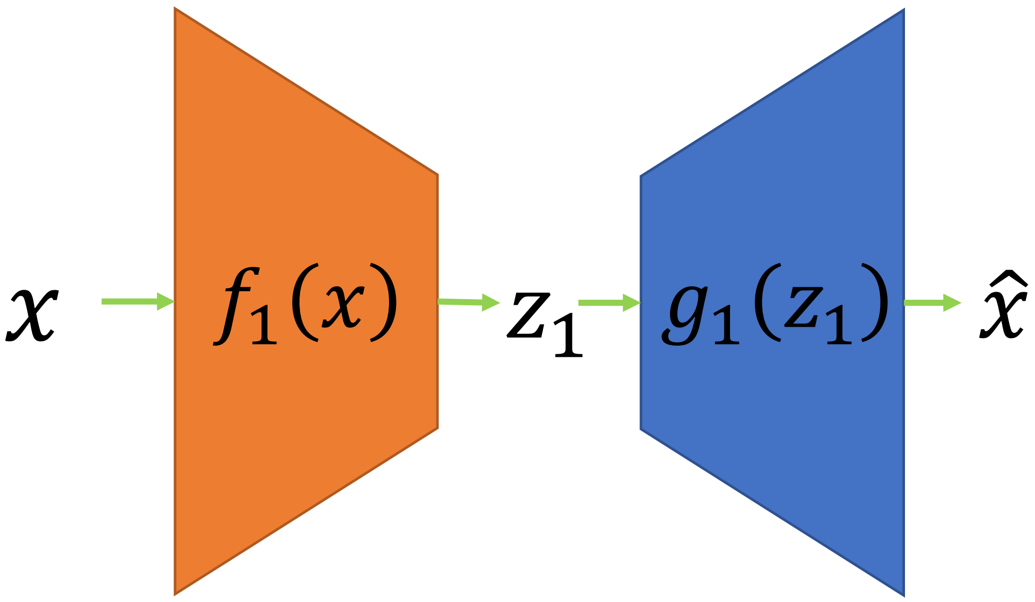
and where λ controls the balance of our reconstruction objective and the hidden layer “flatness”.

Stacked AE



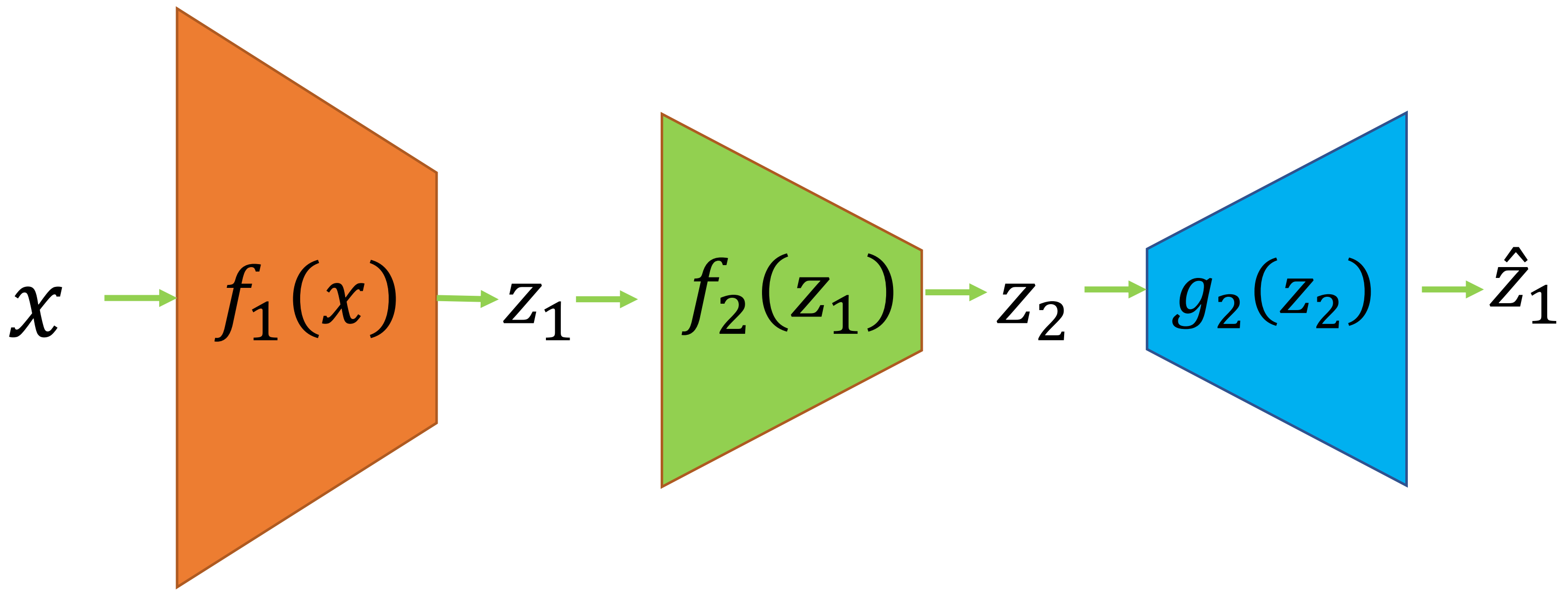
Stacked AE – train process

First Layer Training (AE 1)



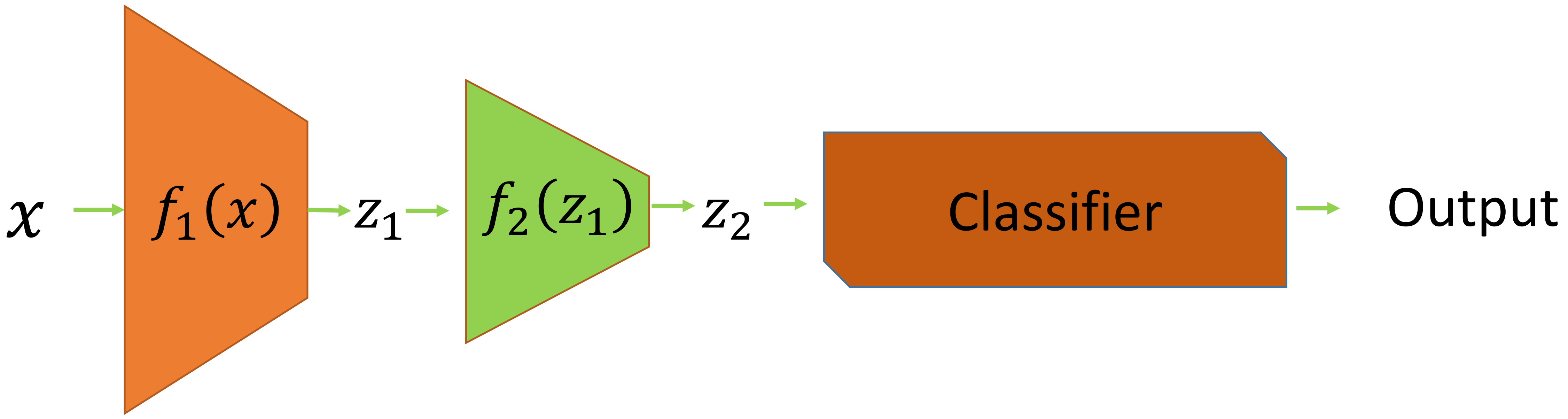
Stacked AE – train process

Second Layer Training (AE 2)



Stacked AE – train process

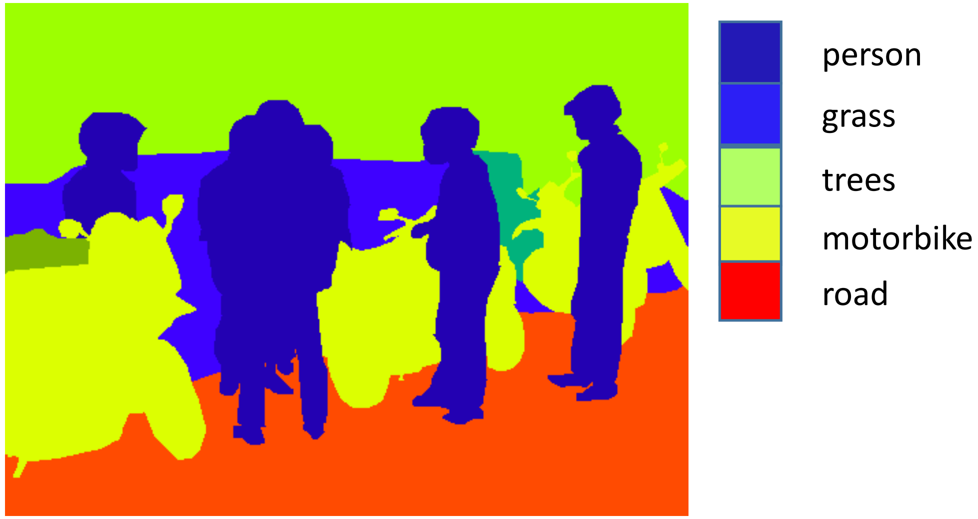
Add any classifier



Class encoder

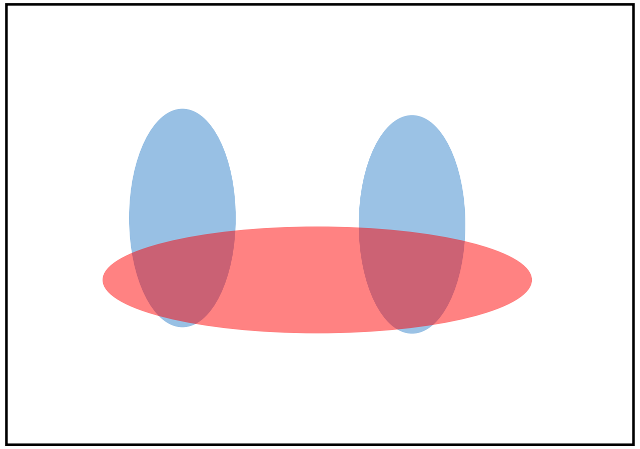
- Define an autoencoder for each class
- Cross sample reconstruction for each class
- Can we put some constraint in the encoder space to ensure non-redundancy in the latent features?

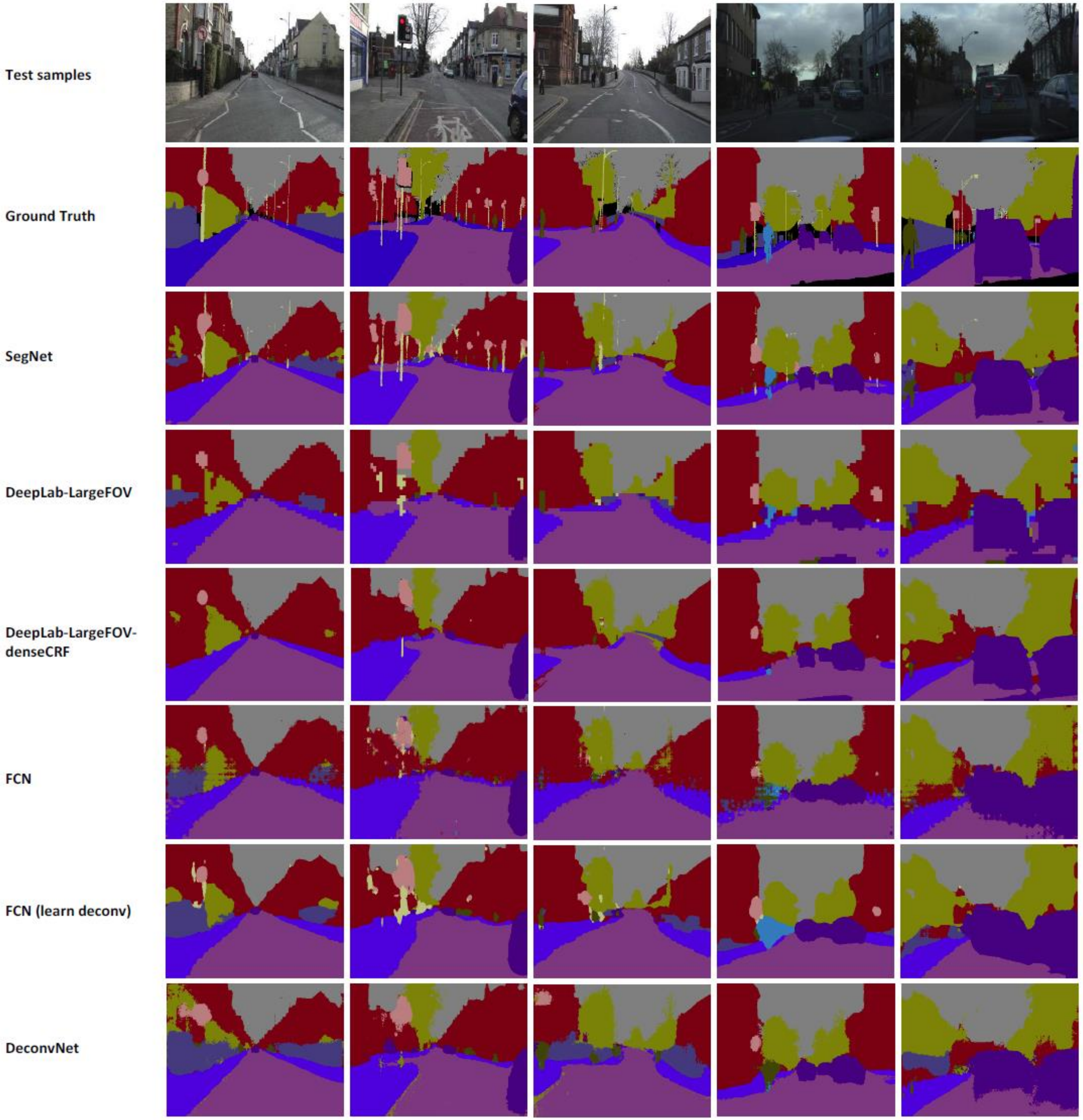
The Task

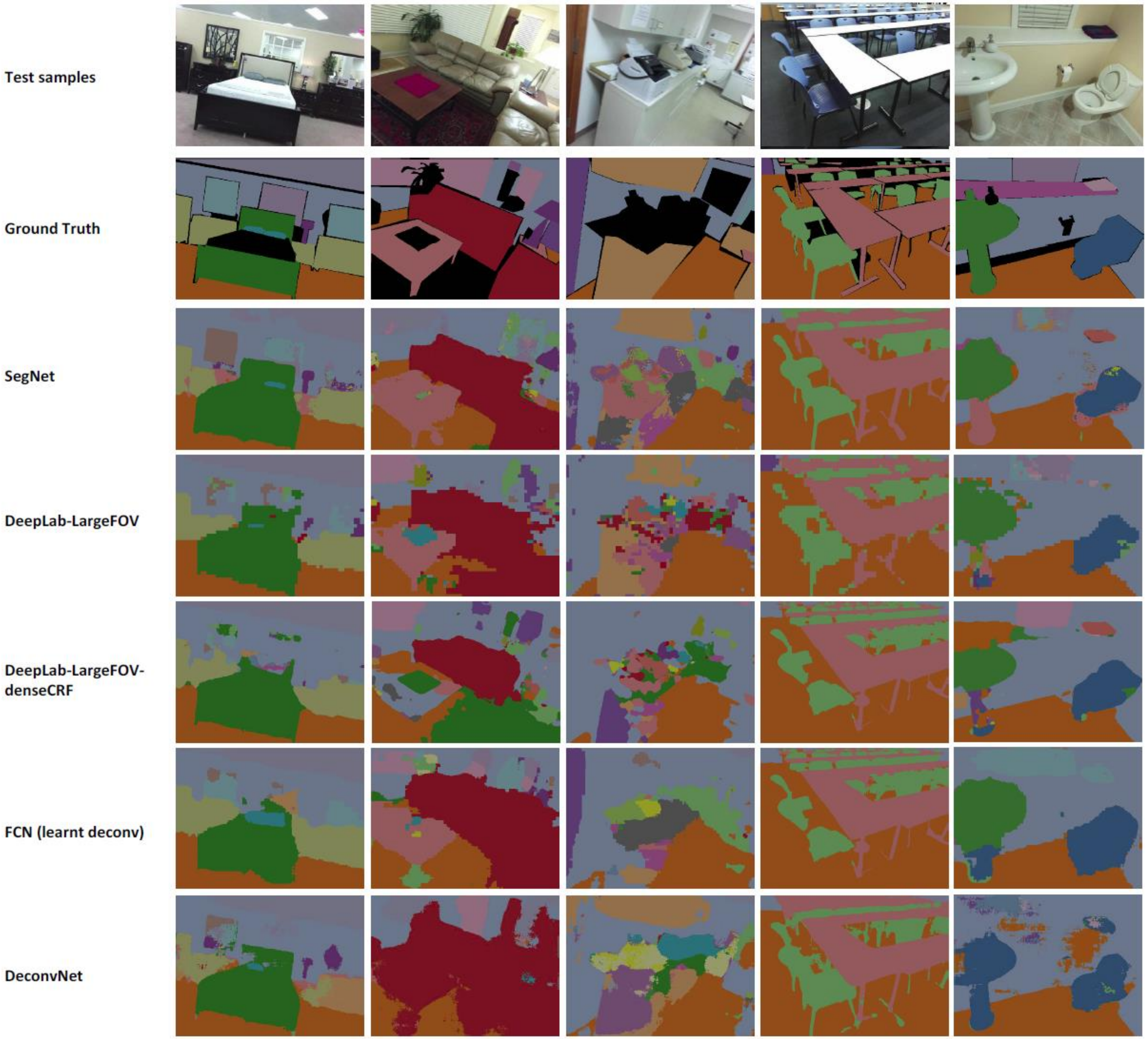


Evaluation metric

- Pixel classification!
- Accuracy?
  - Heavily unbalanced
  - Common classes are over-emphasized
- *Intersection over Union*
  - Average across classes and images
- Per-class accuracy
  - Compute accuracy for every class and then average




Test samples
Ground Truth
SegNet
DeepLab-LargeFOV
DeepLab-LargeFOV-denseCRF
FCN
FCN (learn deconv)
DeconvNet


Test samples
Ground Truth
SegNet
DeepLab-LargeFOV
DeepLab-LargeFOV-denseCRF
FCN (learnt deconv)
DeconvNet

Things vs Stuff

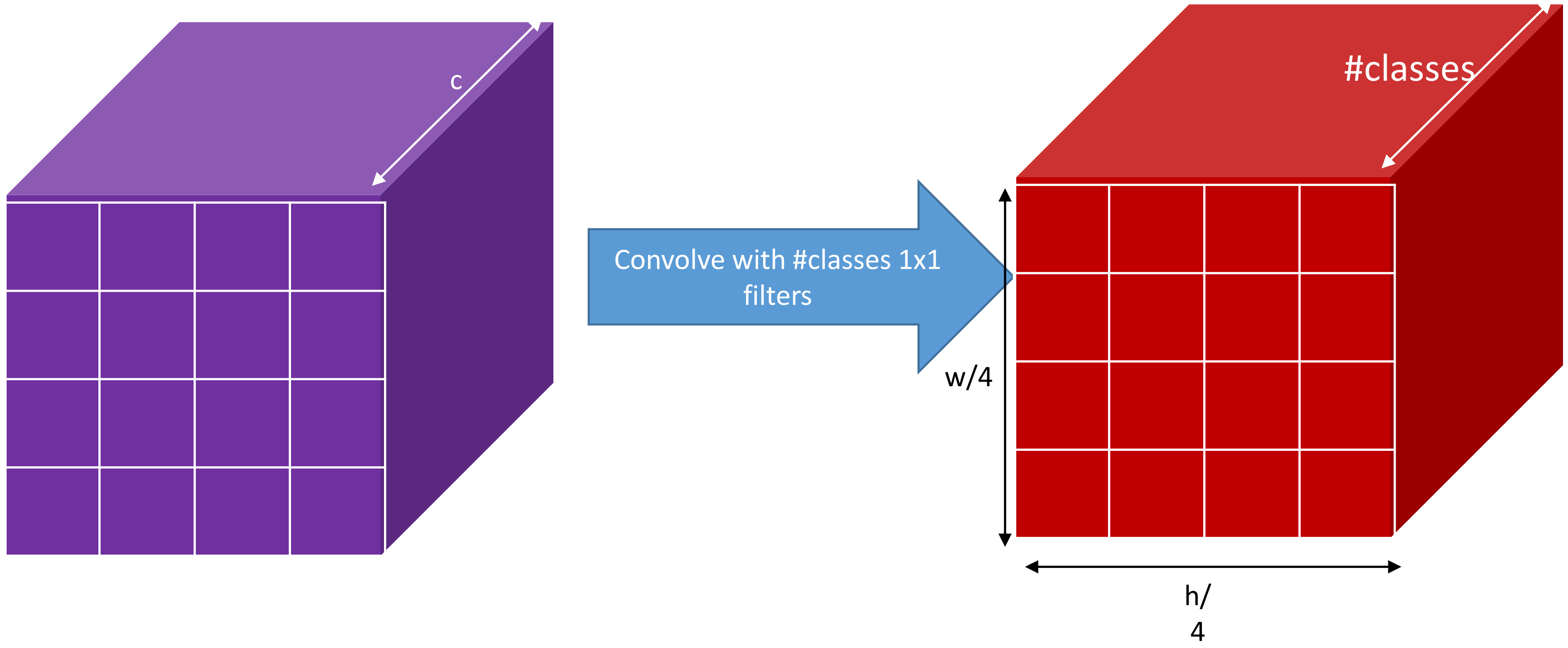
THINGS

- Person, cat, horse, etc
- Constrained shape
- Individual instances with separate identity
- May need to look at objects

STUFF

- Road, grass, sky etc
- Amorphous, no shape
- No notion of instances
- Can be done at pixel level
- “texture”

Semantic segmentation using convolutional networks



The resolution issue

- Problem: Need fine details!
- Shallower network / earlier layers?
  - Deeper networks work better: more abstract concepts
  - Shallower network => Not very semantic!
- Remove subsampling?
  - Subsampling allows later layers to capture larger and larger patterns
  - Without subsampling => Looks at only a small window!

Image pyramids

Use of multiple image resolution helps considering scale

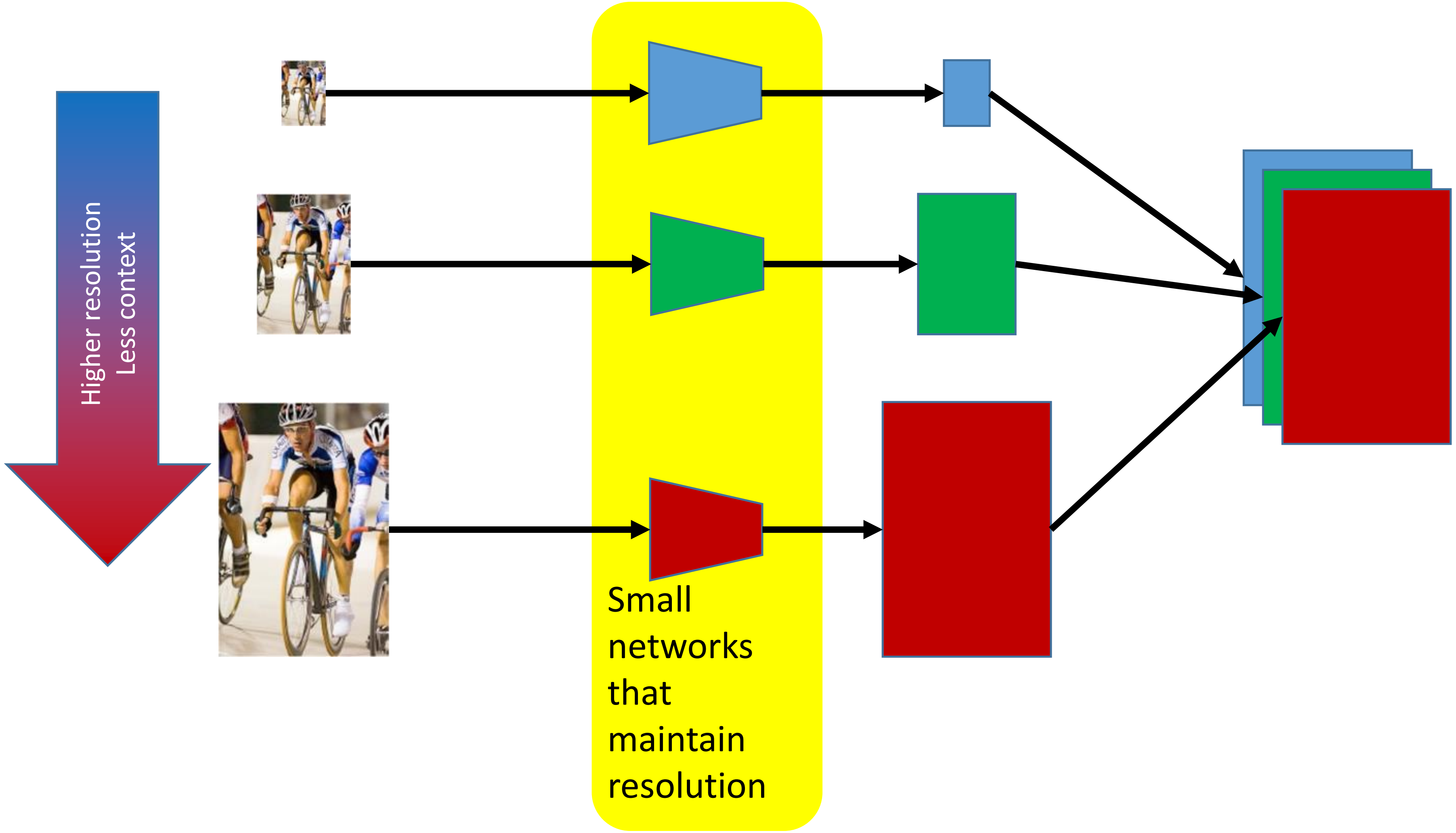
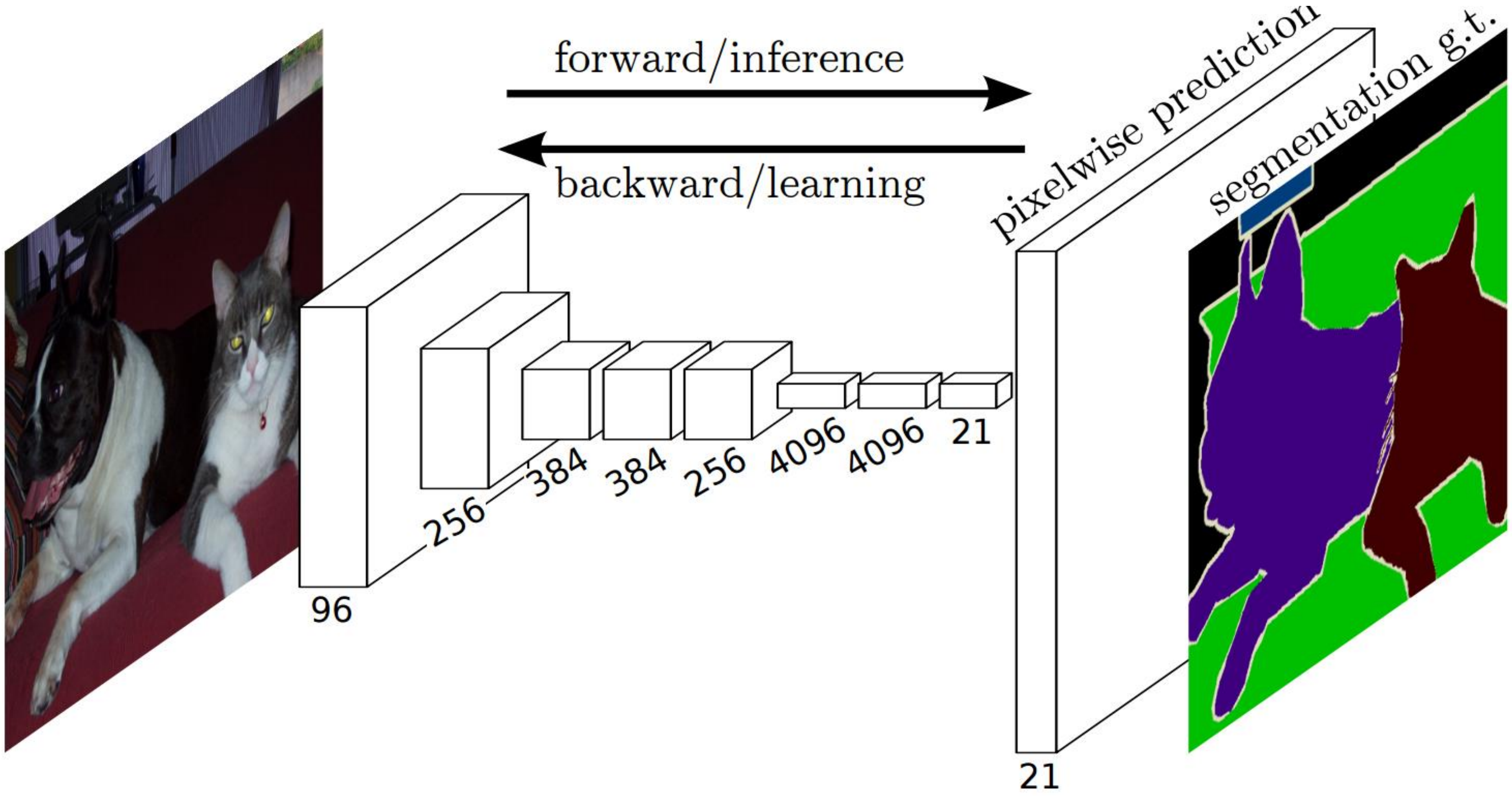
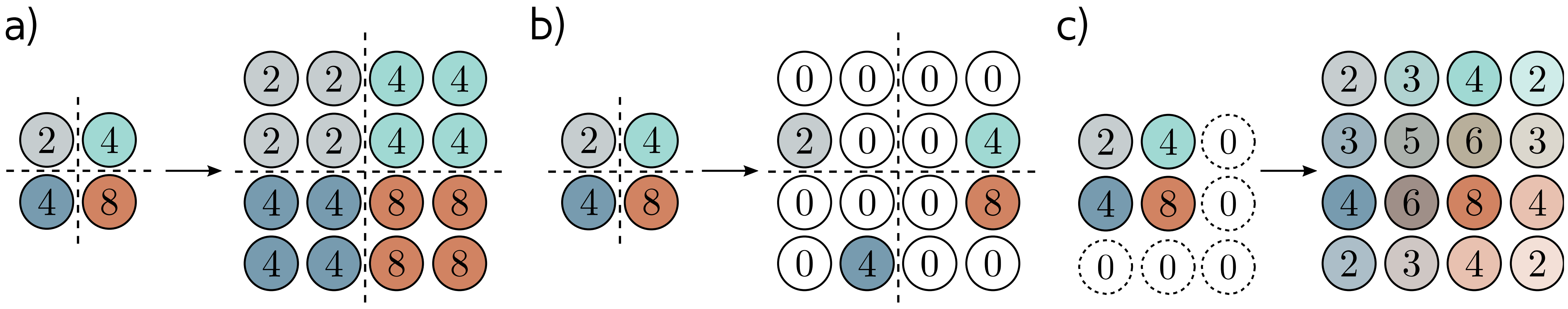


Image segmentation FCN



Upsampling

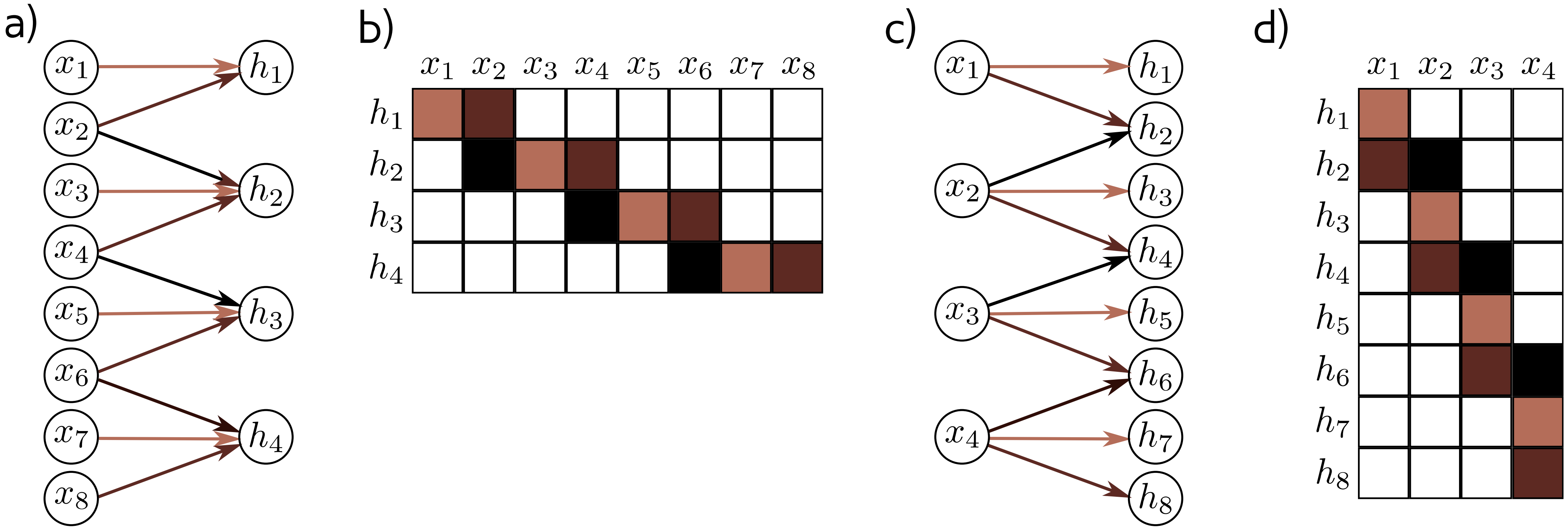


Duplicate

Max-upsampling

Bilinear interpolation

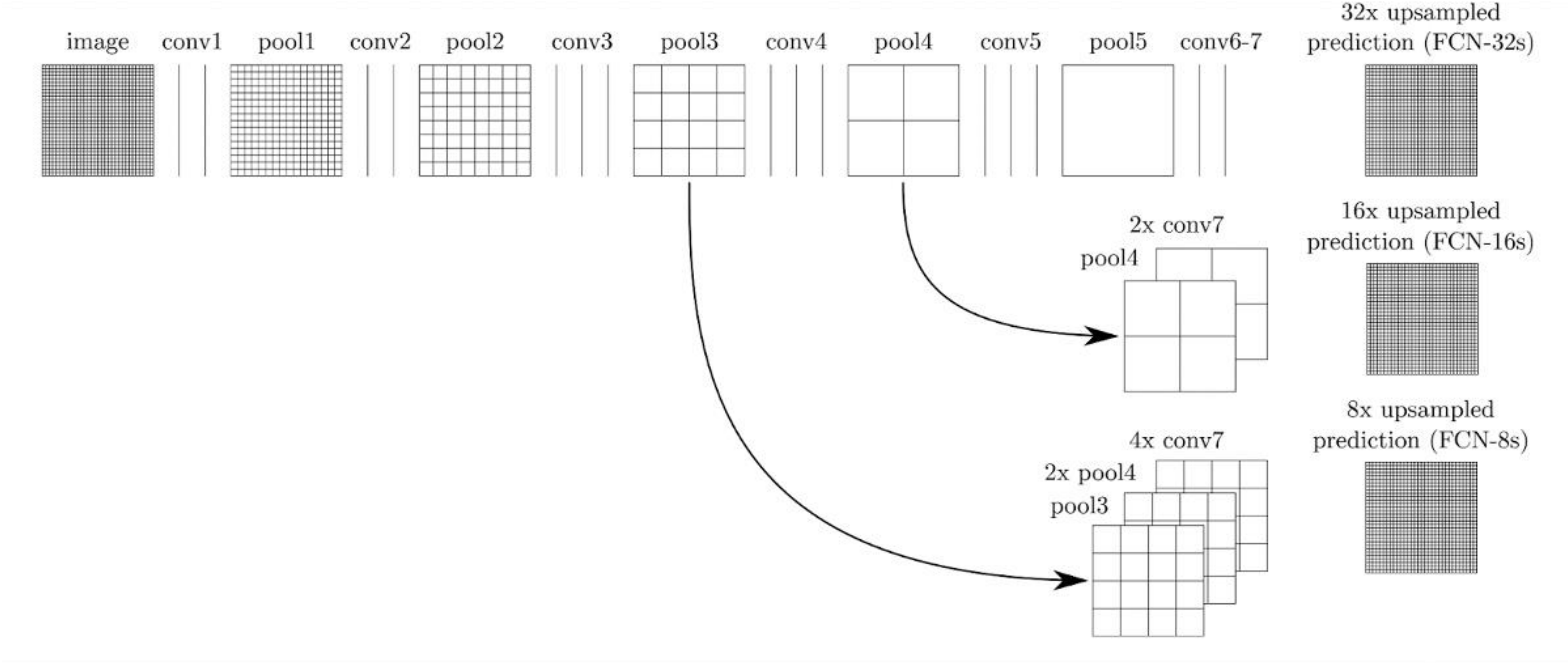
Transposed convolutions



Kernel size 3, Stride 2 convolution

Transposed convolution

Combining responses from multiple layers



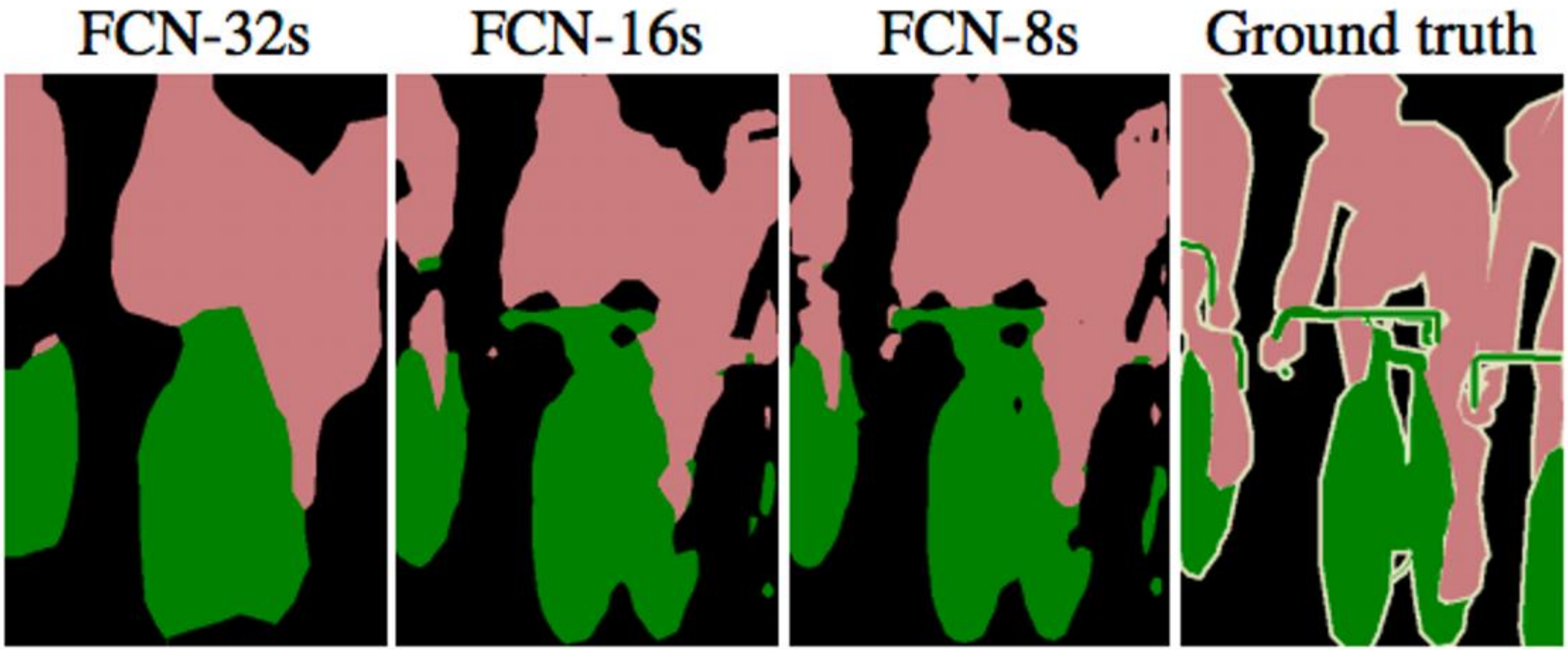
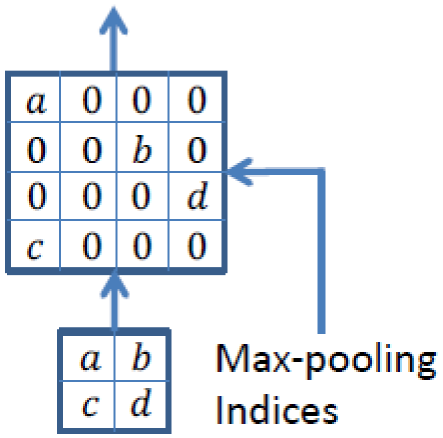


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

Image segmentation - segnet



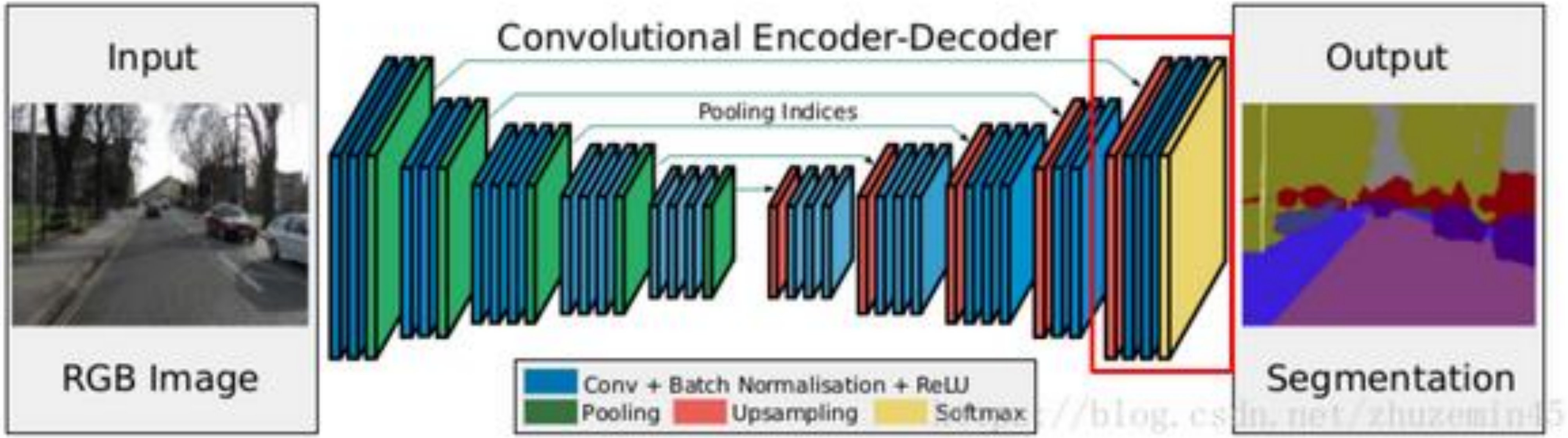
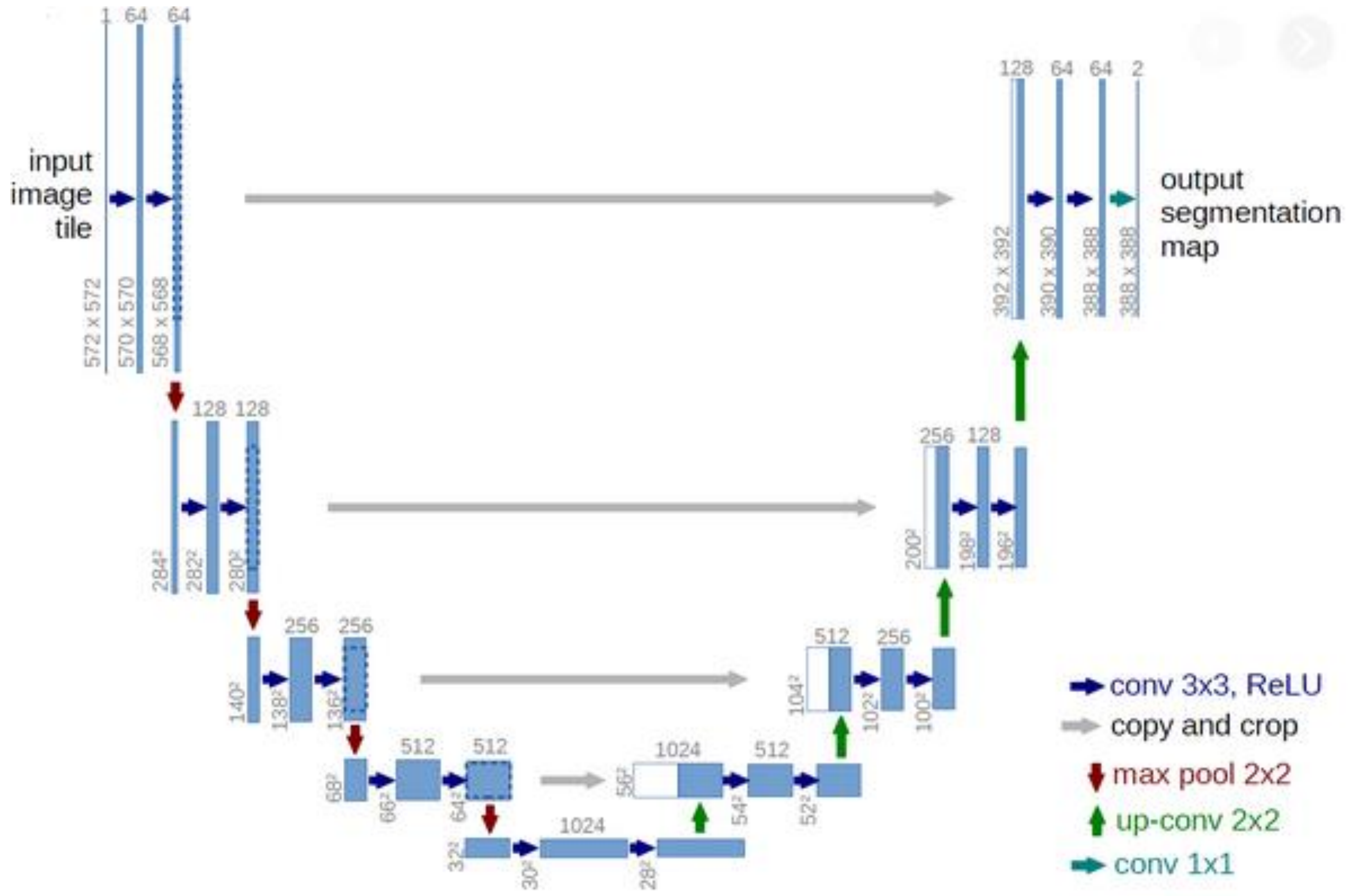


Image segmentation U-net



Mainly used for biomedical data, where precise localization is important

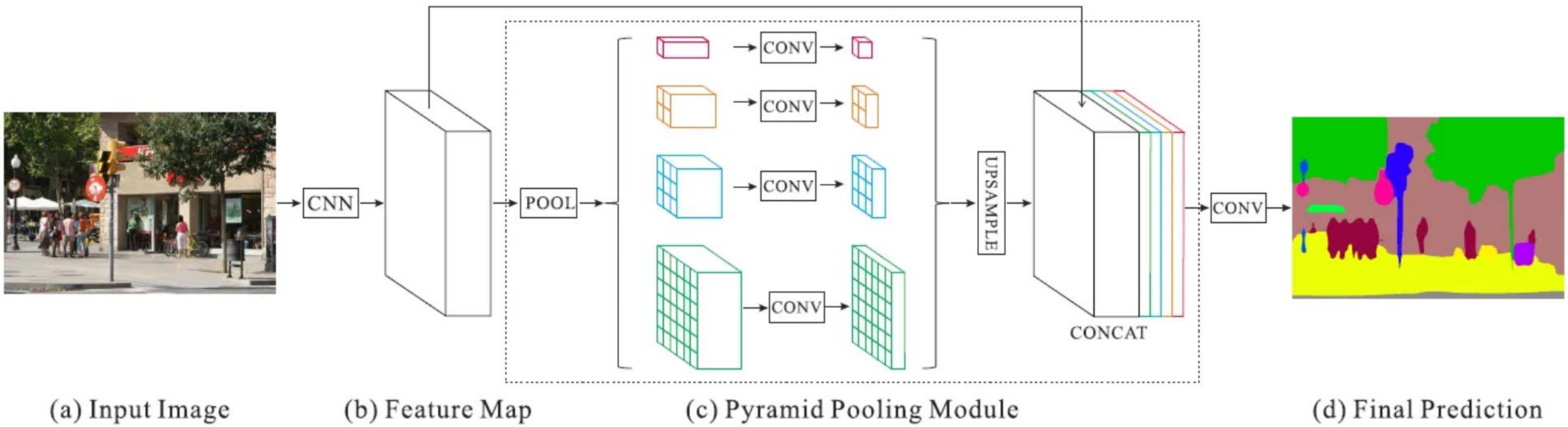
```python
# Encoder
conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

conv4 = Conv2D(512, 3, activation='relu', padding='same')(pool3)
conv4 = Conv2D(512, 3, activation='relu', padding='same')(conv4)
drop4 = Dropout(0.5)(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

# Bottleneck
conv5 = Conv2D(1024, 3, activation='relu', padding='same')(pool4)
conv5 = Conv2D(1024, 3, activation='relu', padding='same')(conv5)
drop5 = Dropout(0.5)(conv5)
```

```python
# Decoder
up6 = UpSampling2D(size=(2, 2))(drop5)
up6 = Conv2D(512, 2, activation='relu', padding='same')(up6)
merge6 = concatenate([drop4, up6], axis=3)
conv6 = Conv2D(512, 3, activation='relu', padding='same')(merge6)
conv6 = Conv2D(512, 3, activation='relu', padding='same')(conv6)

up7 = UpSampling2D(size=(2, 2))(conv6)
up7 = Conv2D(256, 2, activation='relu', padding='same')(up7)
merge7 = concatenate([conv3, up7], axis=3)
conv7 = Conv2D(256, 3, activation='relu', padding='same')(merge7)
conv7 = Conv2D(256, 3, activation='relu', padding='same')(conv7)

up8 = UpSampling2D(size=(2, 2))(conv7)
up8 = Conv2D(128, 2, activation='relu', padding='same')(up8)
merge8 = concatenate([conv2, up8], axis=3)
conv8 = Conv2D(128, 3, activation='relu', padding='same')(merge8)
conv8 = Conv2D(128, 3, activation='relu', padding='same')(conv8)

up9 = UpSampling2D(size=(2, 2))(conv8)
up9 = Conv2D(64, 2, activation='relu', padding='same')(up9)
merge9 = concatenate([conv1, up9], axis=3)
conv9 = Conv2D(64, 3, activation='relu', padding='same')(merge9)
conv9 = Conv2D(64, 3, activation='relu', padding='same')(conv9)

# Output layer (using sigmoid activation for binary segmentation)
conv10 = Conv2D(1, 1, activation='sigmoid')(conv9)
```

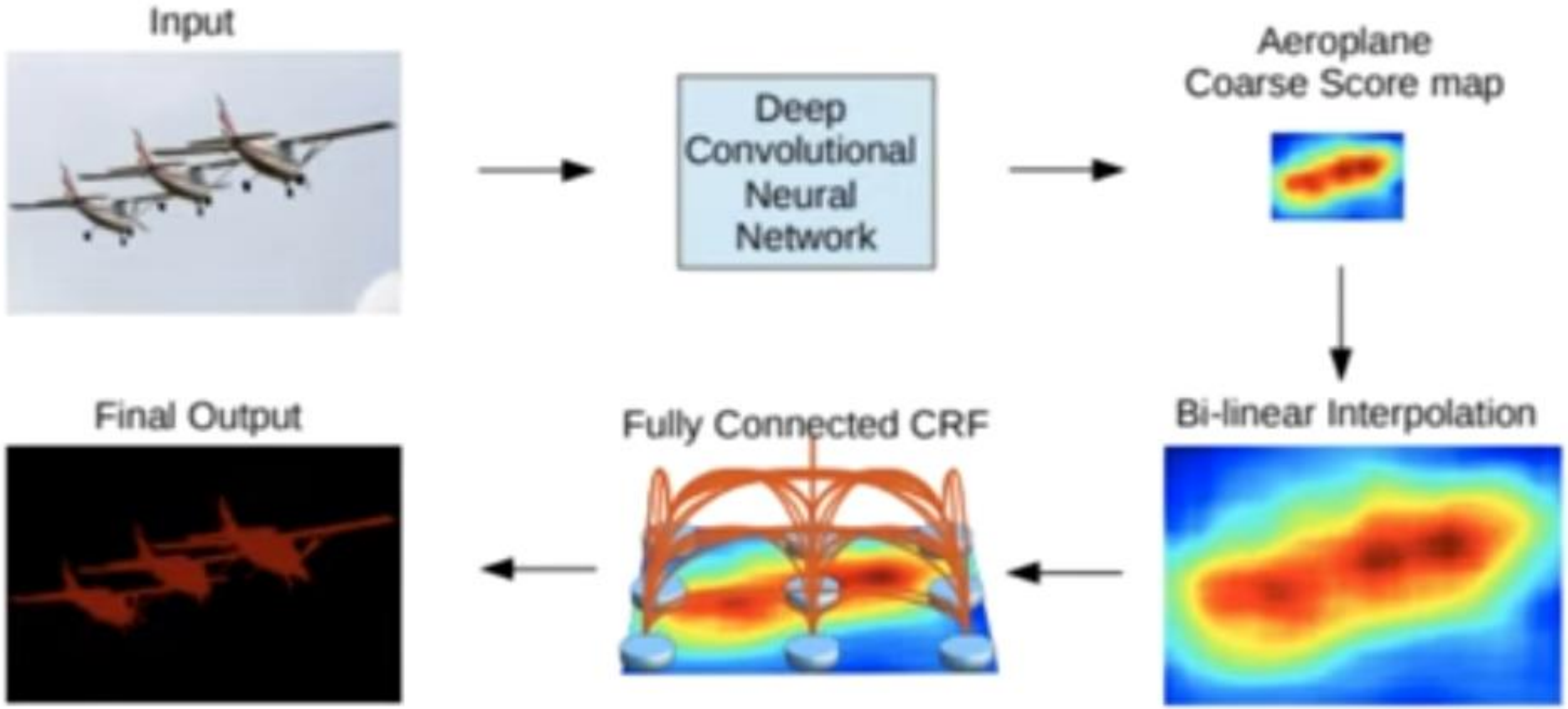
Comparison

- **U-Net:** Uses skip connections to combine high-level context with low-level details for precise segmentation.
- **SegNet:** Relies on pooling indices for upsampling, emphasizing memory efficiency and boundary preservation without needing extensive skip connections.

PSPNet (Pyramid scene parsing)



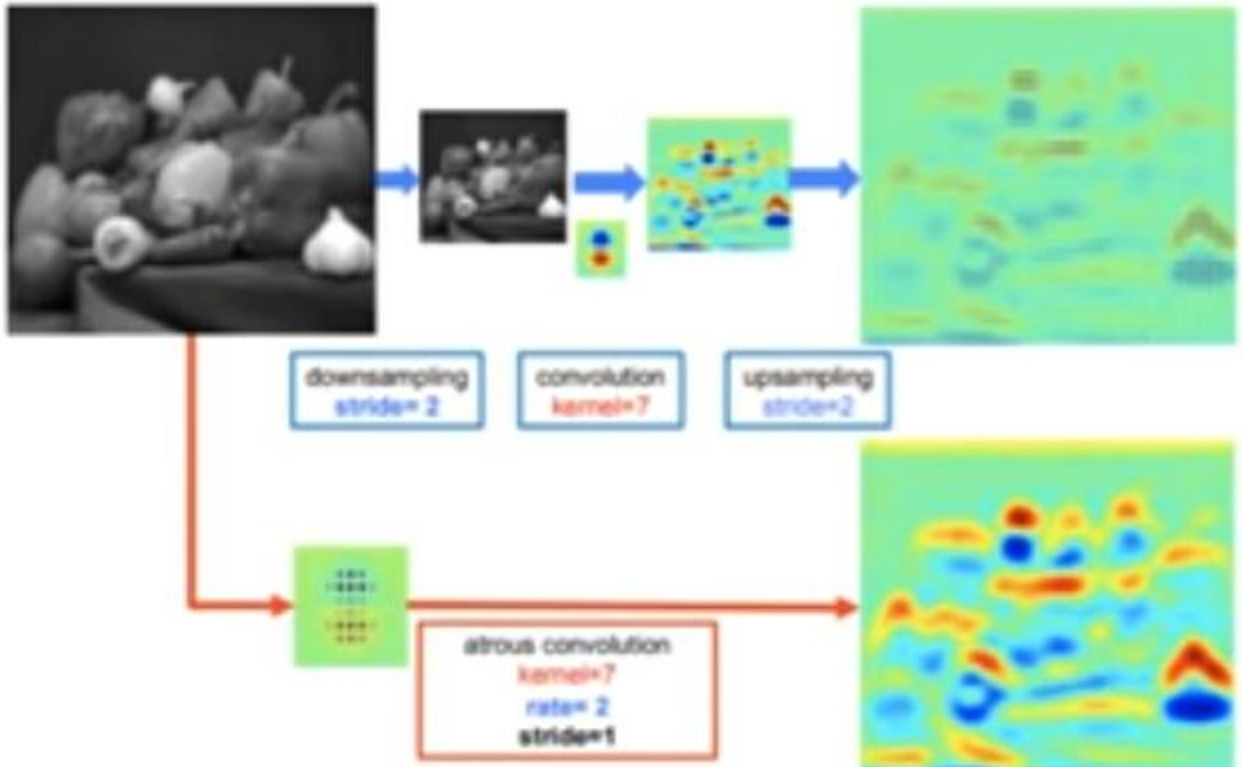
Deeplab idea



Challenges handled

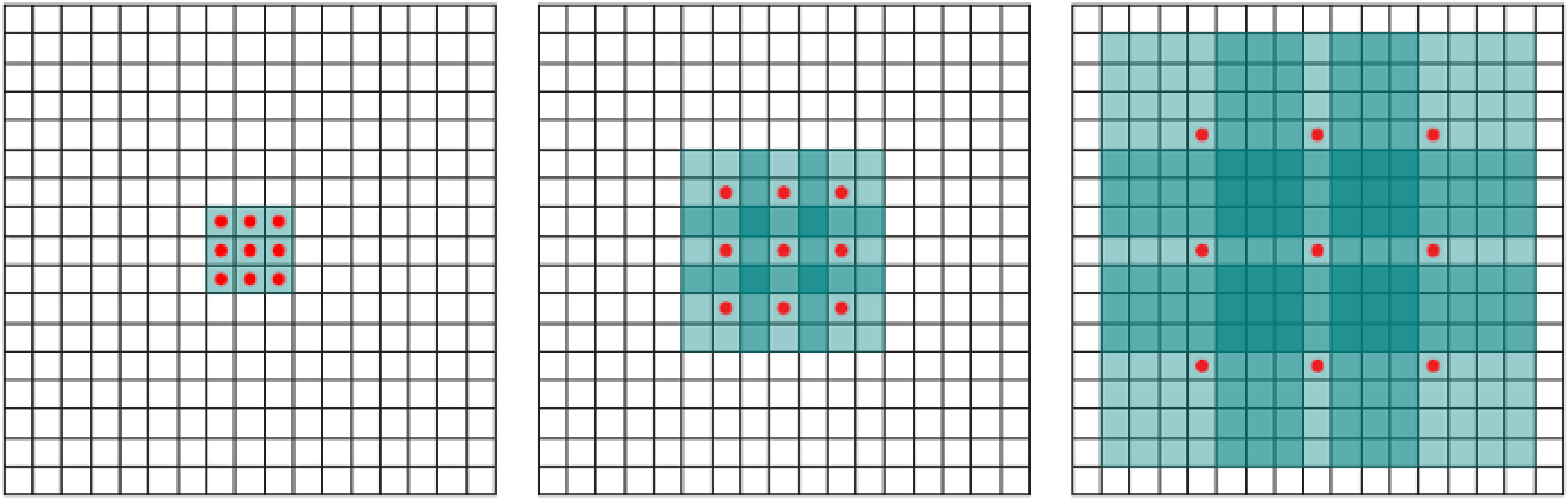
- Reduced feature resolution – Use Atrous convolution
- Objects exist in multiple scale – Pyramid pooling
- Poor localization at the edges – Refinement using CRF

Atrous convolution



Sparse vs dense feature map generation

Dialated convolution



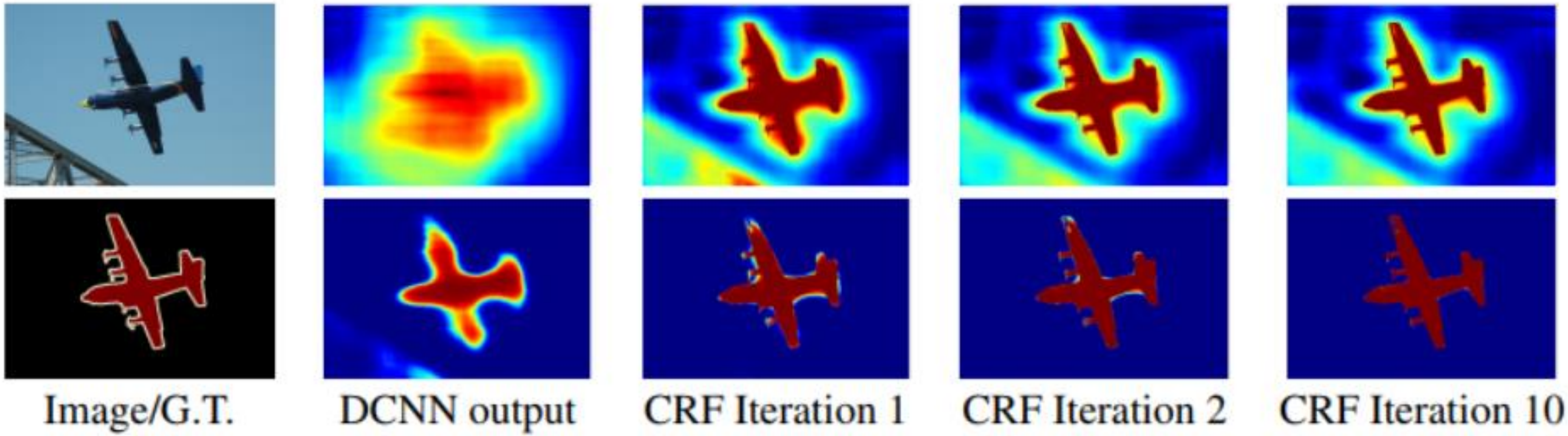
CRF

- Boykov and Jolly (2001)

$$E(x,y) = \sum_i \varphi(x_i, y_i) + \sum_{ij} \psi(x_i, x_j)$$

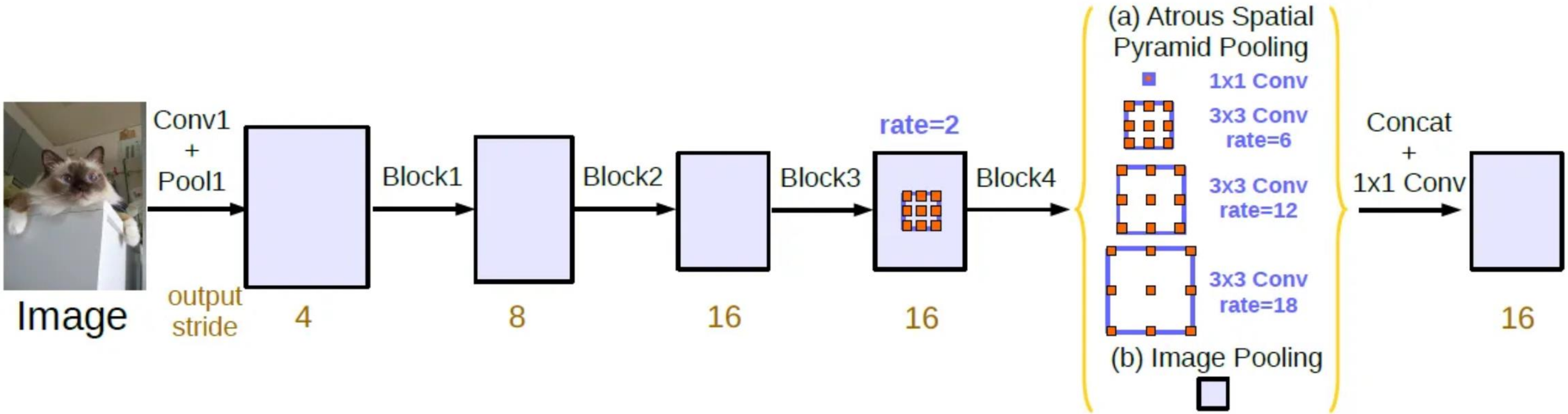
- Variables
  - x_i: Binary variable
    - foreground/background
  - y_i: Annotation
    - foreground/background/empty
- Unary term
  - $\varphi(x_i, y_i) = K[x_i \neq y_i]$
  - Pay a penalty for disregarding the annotation
- Pairwise term
  - $\psi(x_i, x_j) = [x_i \neq x_j]w_{ij}$
  - Encourage smooth annotations
  - w_{ij} affinity between pixels i and j

Effects of CRF

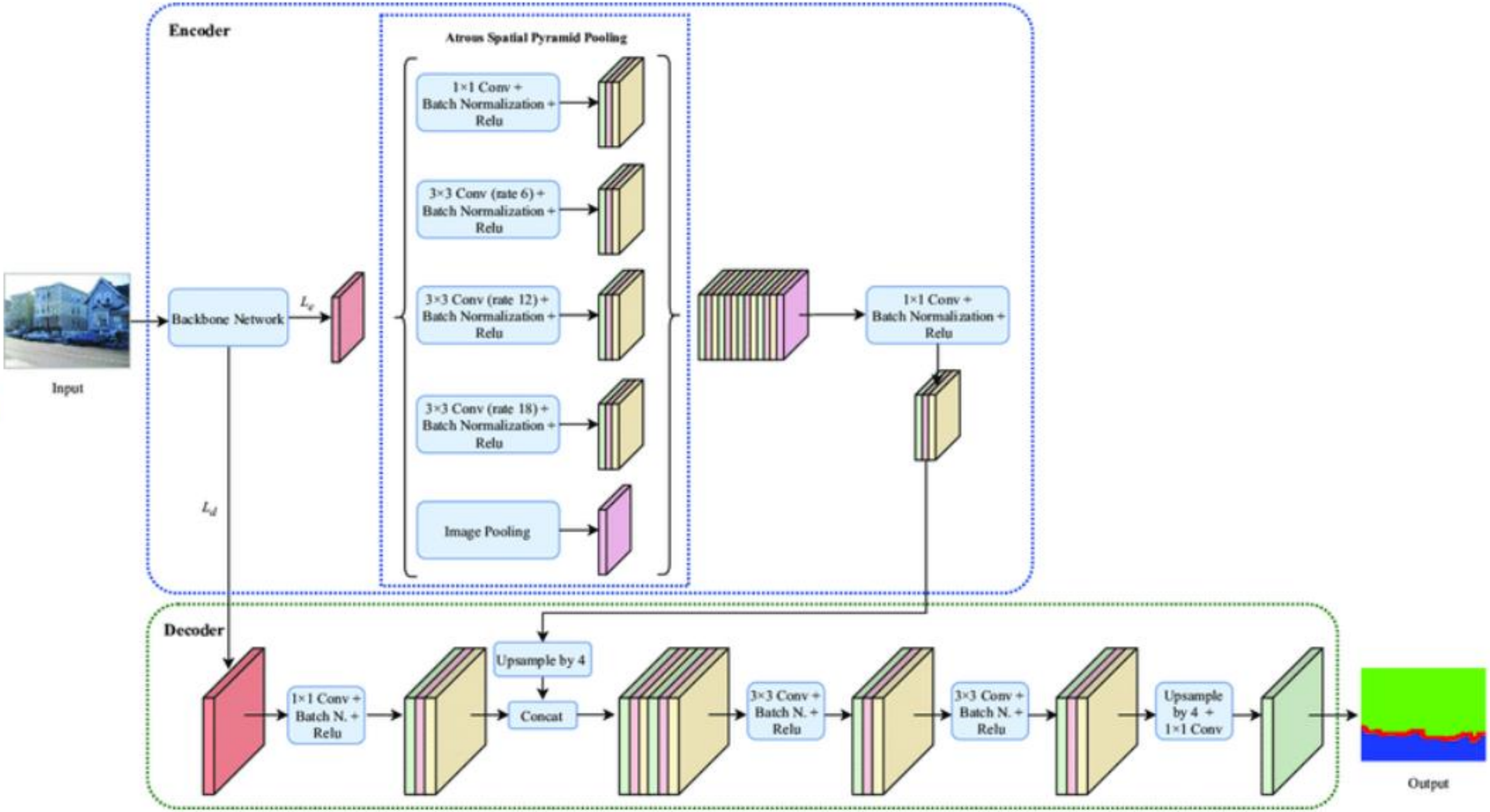


Score map (input before softmax function) and belief map (output of softmax function) for Aeroplane. The image shows the score (1st row) and belief (2nd row) maps after each mean field iteration. The output of last DCNN layer is used as input to the mean field inference.

Deeplab

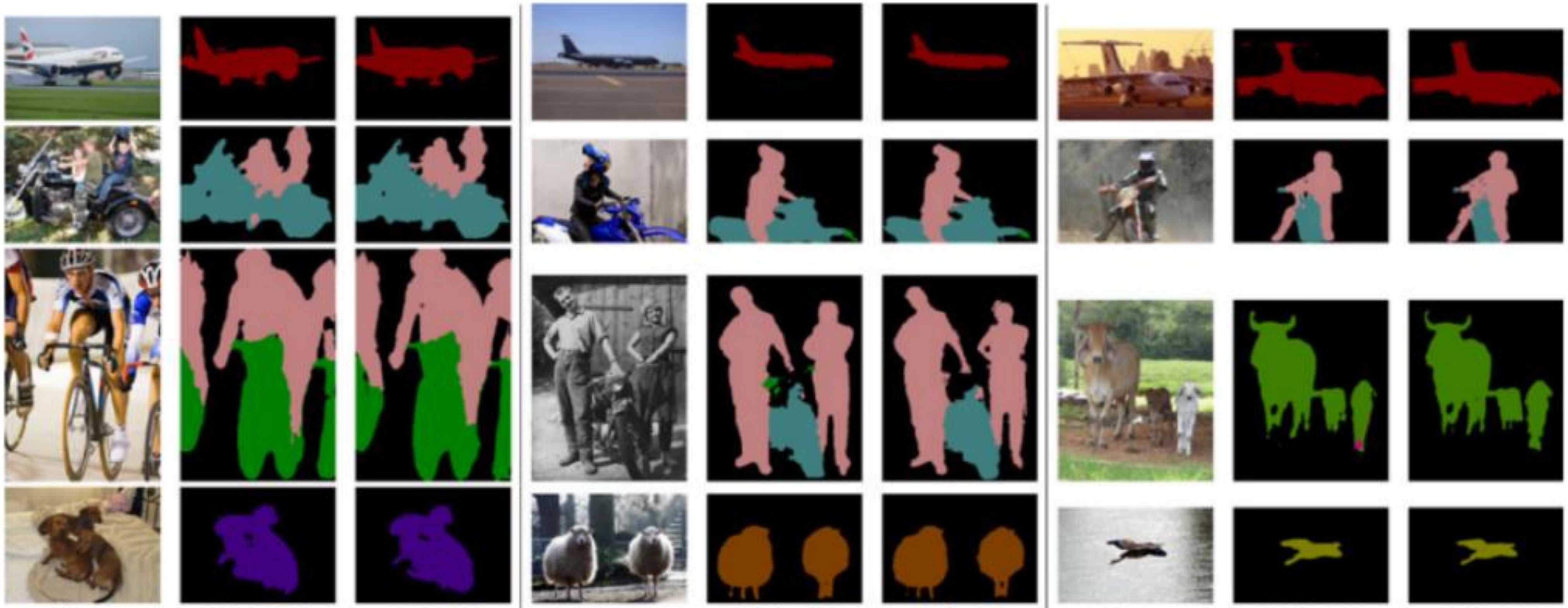


Atrous Spatial Pyramid Pooling (ASPP)

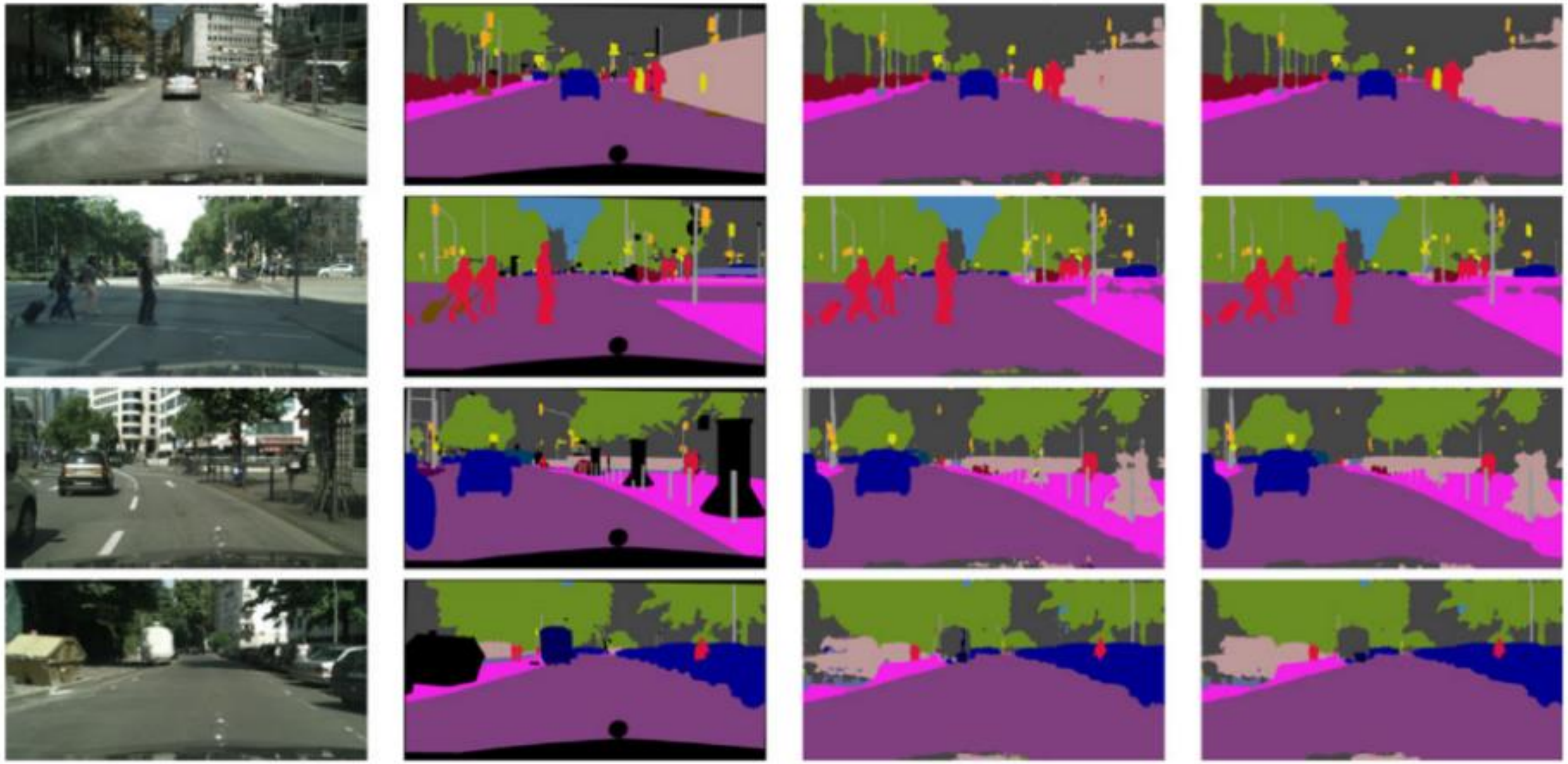


Architecture of DeepLabV3+ with backbone network.

Deeplab results



(a) Image (b) Before CRF (c) After CRF (a) Image (b) Before CRF (c) After CRF (a) Image (b) Before CRF (c) After CRF



(a) Image (b) G.T. (c) Before CRF (d) After CRF

Instance segmentation