

Convolution Neural Net: Basics to Recent Research

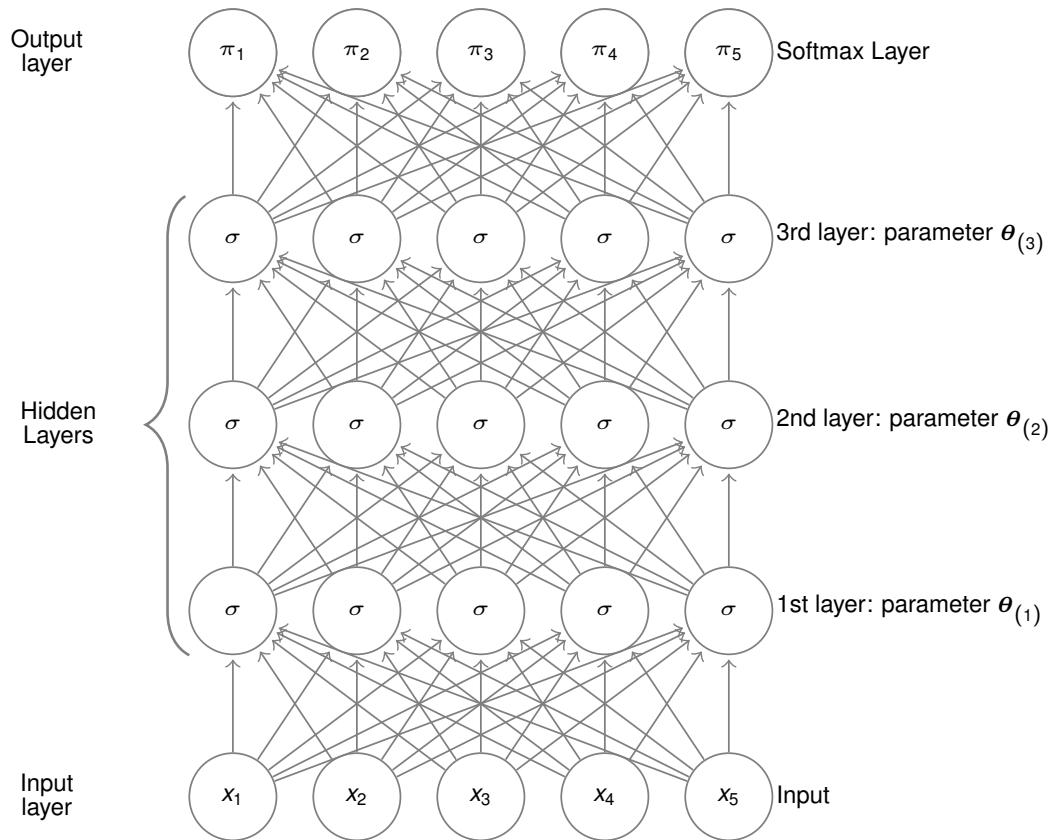
A/Prof Richard Yi Da Xu
Yida.Xu@uts.edu.au
Wechat: aubedata

<https://github.com/roboticcam/machine-learning-notes>

University of Technology Sydney (UTS)

March 20, 2018

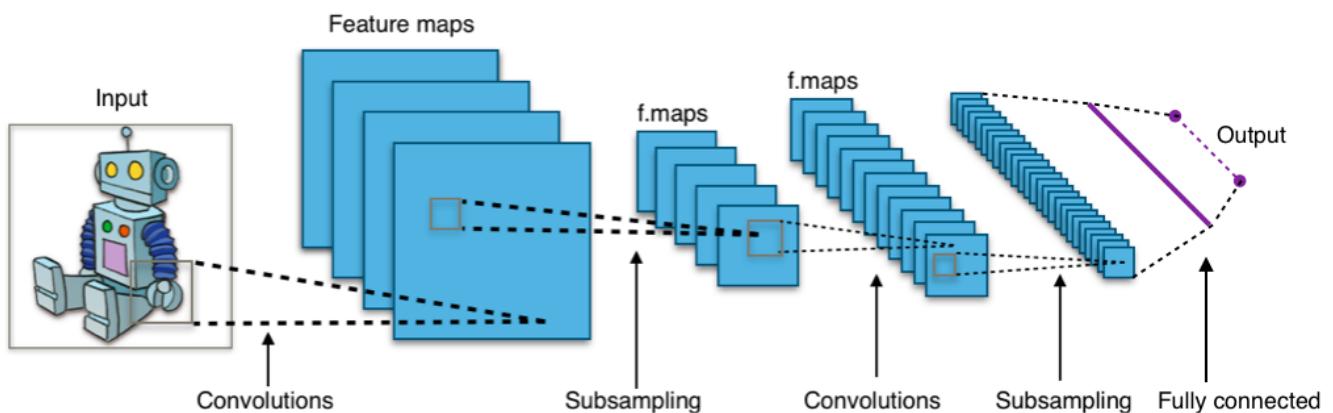
Good old Feedforward Neural Network with Many layers



Feedforward Neural Network with Many layers

- ▶ For complicated problem, one hidden layer may NOT be enough.
- ▶ σ Layers can be flexibly designed: linear, Softmax, ReLU or any other
- ▶ Each layers may have their own parameters $\theta_{(l)}$
- ▶ There are many such layers, hence many parameters
- ▶ Think about the case of **one meg pixel image**.
- ▶ How may we reduce the number of parameters from the fully connected network?
- ▶ keyword of the day: **parameter sharing**

A high level view of Convolution Neural Network (CNN)



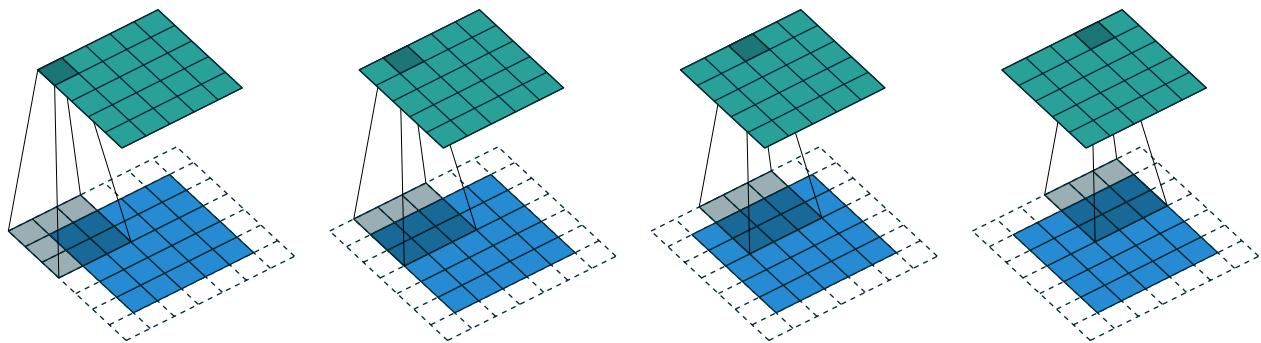
Convolution Neural Networks: What is a convolution?

- ▶ from a signal processing perspective:

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau = \int_{-\infty}^{\infty} f(t - \tau) g(\tau) d\tau.$$

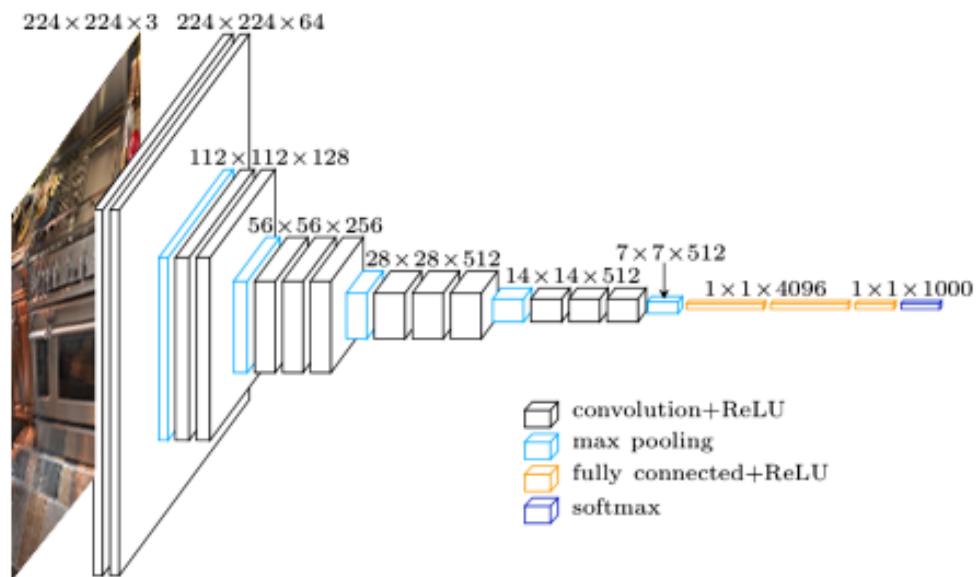
What is a 2D Convolution in Discrete Imaging?

An example of Convolving a 3×3 kernel over a 5×5 input with padding1.

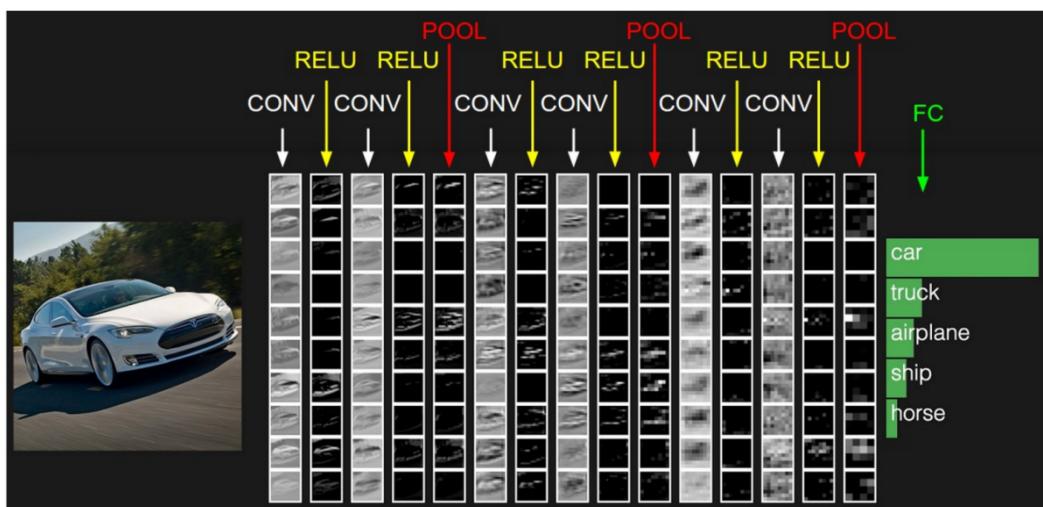


- ▶ Exploiting the strong spatially local correlation present in natural images.
- ▶ In term of number of parameters, assume in a layer, you have 128 convolution Kernels (the grey square), you create 64 “copies” of the image.
- ▶ Assume in a layer, if each kernel is 5×5 in size, we have something like $128 \times 5 \times 5 = 3200$ parameters.
- ▶ A lot less than fully connected neural networks

CNN for image classification: look at it again



What does the “learnt” image feature represent?



CNN Feed-forward: convolution as a linear operator

- ▶ let's forget about only considers:

$$x_{ij}^{(l)} = W * (y^{(l-1)}) \quad y_{ij}^{(l)} = \sigma(x_{ij}^{(l)})$$

- ▶ as matter of fact, in $1 - D$ convolution, one may replace convolution by matrix multiplication:

$$x^l = W * y^{l-1} = \begin{bmatrix} w_1 & 0 & \dots & 0 & 0 \\ w_2 & w_1 & \dots & \vdots & \vdots \\ w_3 & w_2 & \dots & 0 & 0 \\ \vdots & w_3 & \dots & w_1 & 0 \\ w_{m-1} & \vdots & \dots & w_2 & w_1 \\ w_m & w_{m-1} & \vdots & \vdots & w_2 \\ 0 & w_m & \dots & w_{m-2} & \vdots \\ 0 & 0 & \dots & w_{m-1} & w_{m-2} \\ \vdots & \vdots & \vdots & w_m & w_{m-1} \\ 0 & 0 & 0 & \dots & w_m \end{bmatrix} \begin{bmatrix} y_1^{l-1} \\ y_2^{l-1} \\ y_3^{l-1} \\ \vdots \\ y_n^{l-1} \end{bmatrix}$$

CNN Feed-forward: looking at one layer only

- ▶ say at layer $(l - 1)$: $y^{(l-1)}$ has $N \times N$ square neurons
- ▶ use $m \times m$ filter ω
- ▶ when stride 0 used, $x^{(l)}$ will be of size $(N - m + 1) \times (N - m + 1)$
- ▶ for simplicity - shift the centre by $(\frac{m}{2}, \frac{m}{2})$

$$x_{ij}^{(l)} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{(l-1)}$$
$$y_{ij}^{(l)} = \sigma(x_{ij}^{(l)})$$

CNN back-propagation

- ▶ we need four derivative quantities:

$$\blacktriangleright \frac{\partial y_{ij}^{(l)}}{\partial x_{ij}^{(l)}} = \sigma'(x_{ij}^{(l)})$$

$$\blacktriangleright \frac{\partial x_{ij}^{(l)}}{\partial \omega_{ab}} = y_{(i+a)(j+b)}^{(l-1)}$$

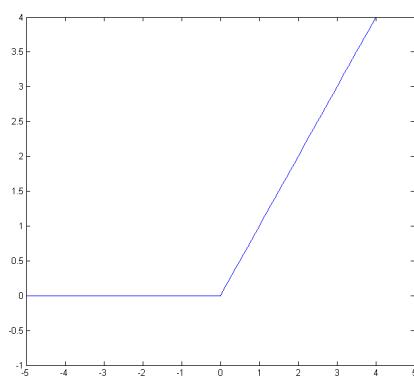
$$\blacktriangleright \frac{\partial x_{(i-a)(j-b)}^{(l)}}{\partial y_{ij}^{(l-1)}} = \omega_{ab}$$

- ▶ say we already knew $\frac{\partial E}{\partial y_{ij}^{(l)}}$, let's find $\frac{\partial E}{\partial y_{ij}^{(l-1)}}$:

$$\begin{aligned} \frac{\partial E}{\partial y_{ij}^{(l-1)}} &= \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{(l)}} \frac{\partial x_{(i-a)(j-b)}^{(l)}}{\partial y_{ij}^{(l-1)}} && y_{ij}^{(l-1)} \text{ contribute to a window-full of } x_{ij}^{(l)} \\ &= \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial y_{ij}^{(l)}} \sigma' \left(x_{(i-a)(j-b)}^{(l)} \right) \omega_{ab} \end{aligned}$$

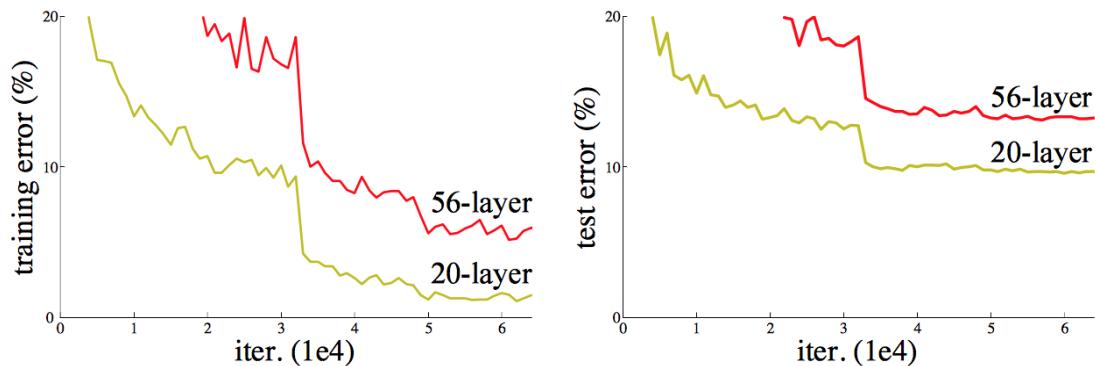
The winning ingredient: Rectified Linear Unit

- ▶ How to combine the different “convolved” images together?
- ▶ A Linear + **Activation function** $f(x) = \max(0, x)$



- ▶ High school mathematics is useful
- ▶ **its derivative can only be** $\{0, 1\}$. This is useful to prevent gradient vanishing and exploding
- ▶ They do not require any exponential computation (such as those required in sigmoid)
- ▶ reported around 6X speed over existing activation functions

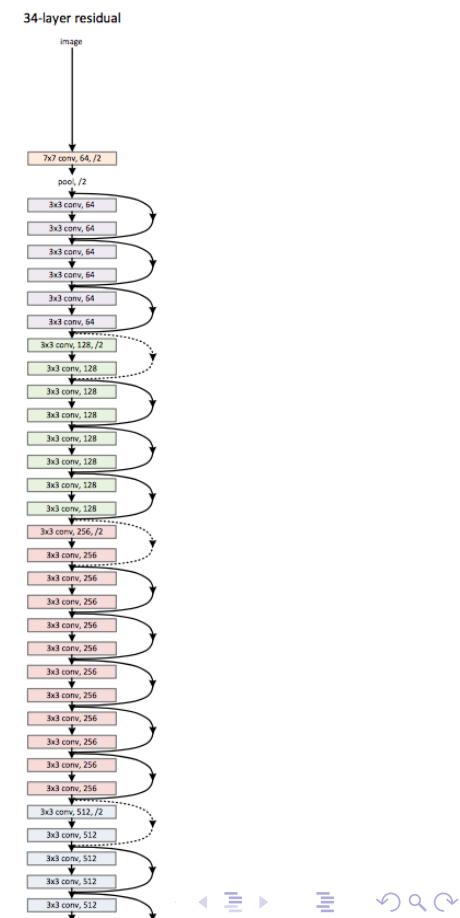
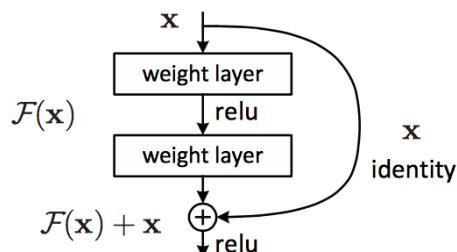
New network designs: Deep Residual Learning



- ▶ when network depth increasing, accuracy gets saturated and then degrades rapidly.
- ▶ authors claim such degradation is **not** caused by overfitting

New network designs: Deep Residual Learning:

He., et al, (2016), Deep Residual Learning for Image Recognition, CVPR, pp. 770-778



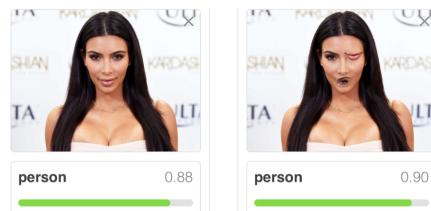
- ▶ instead of few stacked layers directly fit a desired underlying mapping $H(x)$
- ▶ explicitly let these layers fit a residual mapping $F(x)$: $H(x) = F(x) + x$
- ▶ **hypothesize**: easier to optimize residual mapping than optimize original unreferenced mapping.
- ▶ if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

One way we may able to argue its advantage

$$\begin{aligned}x_2 &= F(x_1) + x_1 \quad \dots \quad x_t = F(x_{t-1}) + x_{t-1} \\ \frac{\partial x_t}{\partial x_1} &= \frac{\partial x_t}{\partial x_{t-1}} \frac{\partial x_{t-1}}{\partial x_{t-2}} \dots \frac{\partial x_2}{\partial x_1} \\ &= (F'(x_{t-1}) + 1)(F'(x_{t-2}) + 1) \dots (F'(x_1) + 1) \\ &= (\text{some polynomial terms of } x_1, \dots, x_t) + \mathbf{1}\end{aligned}$$

Therefore, change x_1 propagates some of it to x_t even though t may be large

Capsule Networks



neural network

$$z_j = W_{j,:}^\top x_i \quad a_j = \sigma(z_j)$$

capsule layer

$\hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i$ a unique matrix link between capsules of adjacent layers

$\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$ dynamic routing

$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1 + \|\mathbf{s}_j\|^2} \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$ normalise a vector to ensure $\|\mathbf{v}_j\| \leq 1$

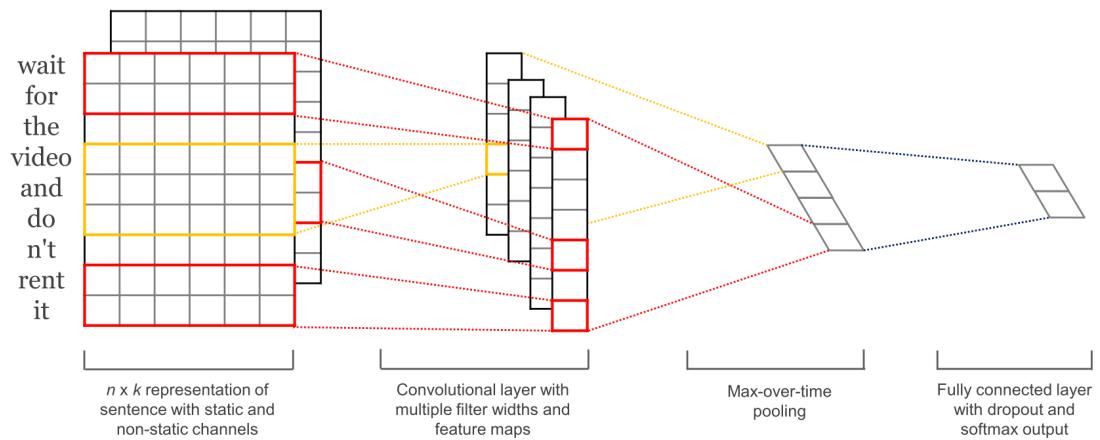
Capsule Networks - Dynamic Routing

Procedure 1 Routing algorithm.

```
1: procedure ROUTING( $\hat{u}_{j|i}$ ,  $r$ ,  $l$ )
2:   for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :  $b_{ij} \leftarrow 0$ .
3:   for  $r$  iterations do
4:     for all capsule  $i$  in layer  $l$ :  $\mathbf{c}_i \leftarrow \text{softmax}(\mathbf{b}_i)$             $\triangleright \text{softmax}$  computes Eq. 3
5:     for all capsule  $j$  in layer  $(l + 1)$ :  $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$ 
6:     for all capsule  $j$  in layer  $(l + 1)$ :  $\mathbf{v}_j \leftarrow \text{squash}(\mathbf{s}_j)$             $\triangleright \text{squash}$  computes Eq. 1
7:     for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :  $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$ 
return  $\mathbf{v}_j$ 
```

- ▶ lower layer is sent to higher level capsule that agrees with it
- ▶ the agreement is iterative updated through its value c_{ij}

CNN for text classification

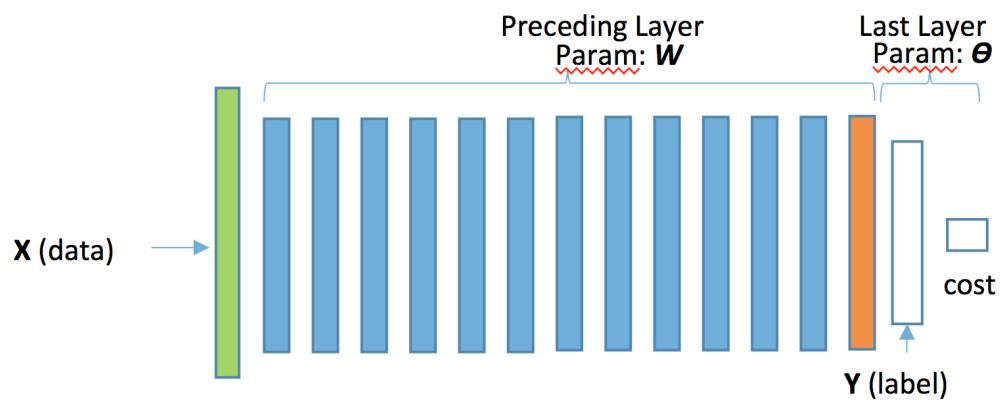


excercise why can't we perform 2-D in text classification?

excercise how does it solve the problem of variable text length?

Generic Deep Neural Network Structure

- ▶ The generic framework for deep learning:



The Last layer of Convolution Neural Network

The structure can be thought of as:

- ▶ The **last layer**: dictating the **objective** of the neural network.
 - ▶ It has parameter set θ . Usually, its dimension has been just a few.
- ▶ The **preceding layers**: **data transformation/preparation** to serve the objective (specified in the **last layer**) better
 - ▶ It has parameter set W . Usually, Its dimension is HUGE!
- ▶ A more “scholarly” way of saying this: **preceding** layers brings **feature embeddings** to serve the function of the last layer.
- ▶ When data is less complex, one may apply the **last layer** directly without the preceding layers, i.e., **no embeddings** is required.

We will look at some of the **Last layer** of CNN

- ▶ Softmax
- ▶ Linear
- ▶ Centre
- ▶ Contrastive
- ▶ Triplet
- ▶ and some of the unique last layers for **object detection**

First of them all: Softmax Layer

$$\mathcal{C}(\theta) = - \sum_{i=1}^N \sum_{k=1}^K \underbrace{y_{i,k}}_{\text{label}} \underbrace{\log \left[\frac{\exp(\mathbf{x}_i^T \theta_k)}{\sum_{l=1}^K \exp(\mathbf{x}_i^T \theta_l)} \right]}_{\text{prob of belong to a class } k}$$

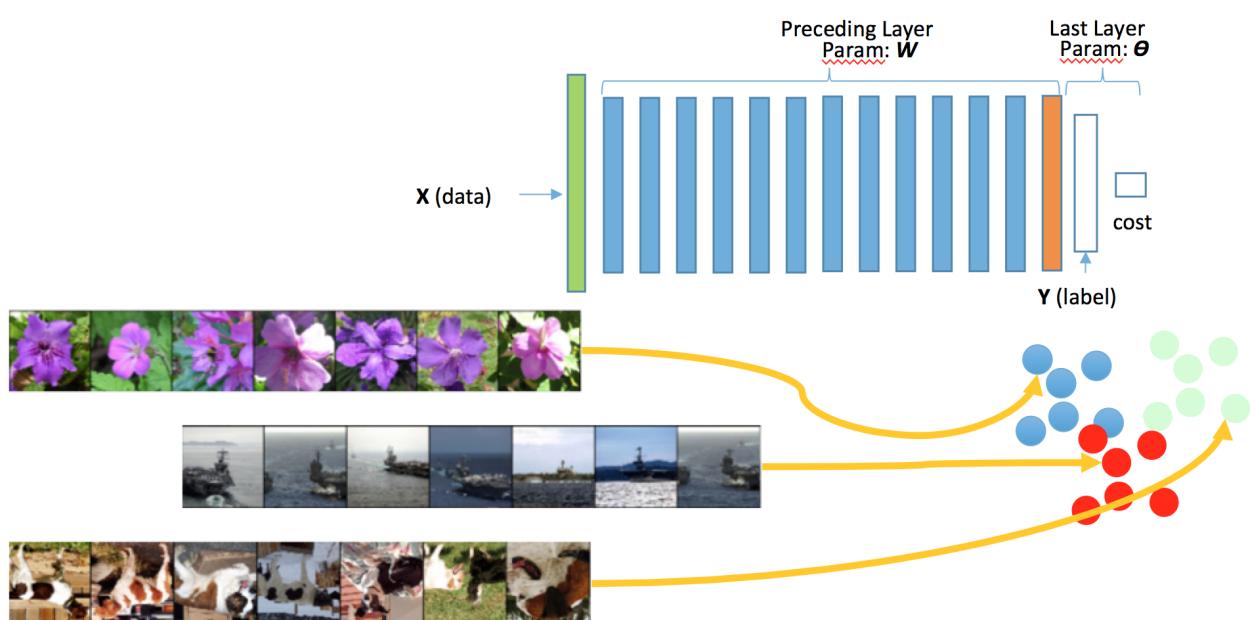
- ▶ Used for classification (Most commonly used)
- ▶ See `soft_max.m` where Softmax is applied **without** the preceding networks.
- ▶ Just for fun: $\arg \min (\mathcal{C}(\theta)) \implies \arg \max (\exp(-\mathcal{C}(\theta)))$:

$$\begin{aligned}\exp(-\mathcal{C}(\theta)) &= \exp \left(\sum_{i=1}^N \sum_{k=1}^K y_{i,k} \log \left[\frac{\exp(\mathbf{x}_i^T \theta_k)}{\sum_{l=1}^K \exp(\mathbf{x}_i^T \theta_l)} \right] \right) \\ &= \prod_{i=1}^N \prod_{k=1}^K \exp \left(y_{i,k} \log \left[\frac{\exp(\mathbf{x}_i^T \theta_k)}{\sum_{l=1}^K \exp(\mathbf{x}_i^T \theta_l)} \right] \right) \\ &= \prod_{i=1}^N \prod_{k=1}^K \left[\frac{\exp(\mathbf{x}_i^T \theta_k)}{\sum_{l=1}^K \exp(\mathbf{x}_i^T \theta_l)} \right]^{y_{i,k}}\end{aligned}$$

- ▶ It is not hard to tell $\exp(-\mathcal{C}(\theta))$ is maximised, when **red** and **blue** parts are **identical**.

Softmax Layer

- ▶ **Without** using preceding networks, Softmax does NOT work well for complicated input, e.g., faces:



- ▶ the **preceding networks** plays the role of the **feature embedding**.
- ▶ Sometimes, pre-trained networks are used to generate **feature embedding** for unseen data (in the same context), i.e., last layer do NOT involved.

Centre loss

- ▶ Wen, et. al, (2016) A Discriminative Feature Learning Approach for Deep Face Recognition, ECCV
- ▶ introduce so-called center loss to minimize the intra-class distances of the deep features
- ▶ c_j are also variables

$$\mathcal{L} = \mathcal{L}_s + \lambda \mathcal{L}_c = - \underbrace{\sum_{i=1}^N \log \frac{\exp^{W_{y_i} x_i}}{\sum_{j=1}^M \exp^{W_j x_i}}}_{\text{softmax}} + \underbrace{\frac{\lambda}{2} \sum_{i=1}^N \|x_i - c_{y_i}\|^2}_{\text{centre}}$$

- 1: **while** not converge **do**
- 2: $t \leftarrow t + 1$
- 3: Compute the joint loss $\mathcal{L}^t = \mathcal{L}_s^t + \lambda \mathcal{L}_c^t$
- 4: Compute the backpropagation error:
$$\frac{\partial \mathcal{L}^t}{\partial x_i^t} = \frac{\partial \mathcal{L}_s^t}{\partial x_i^t} + \lambda \frac{\partial \mathcal{L}_c^t}{\partial x_i^t} \quad W^{t+1} = W^t - \mu^t \frac{\partial \mathcal{L}^t}{\partial W^t} = W^t - \mu^t \frac{\partial \mathcal{L}_s^t}{\partial W^t}$$
- 5: Update each centre: $c_j^{t+1} = c_j^t - \alpha \nabla c_j^t$
- 6: Update convolution parameters: $\theta_C^{t+1} = \theta_C^t - \mu^t \sum_{i=1}^N \frac{\partial \mathcal{L}^t}{\partial x_i^t} \frac{\partial x_i^t}{\partial \theta_C^t}$
- 7: **end while**

Centre Loss

- ▶ Wen, et. al, (2016) A Discriminative Feature Learning Approach for Deep Face Recognition, ECCV

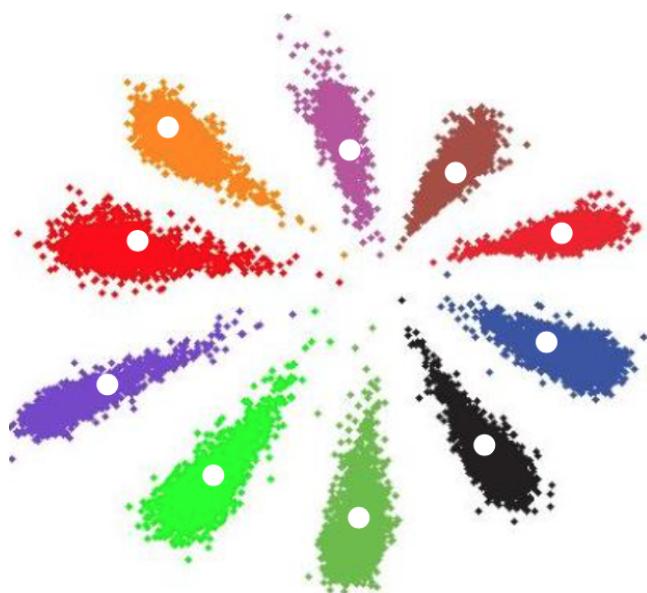


Figure: Softmax only

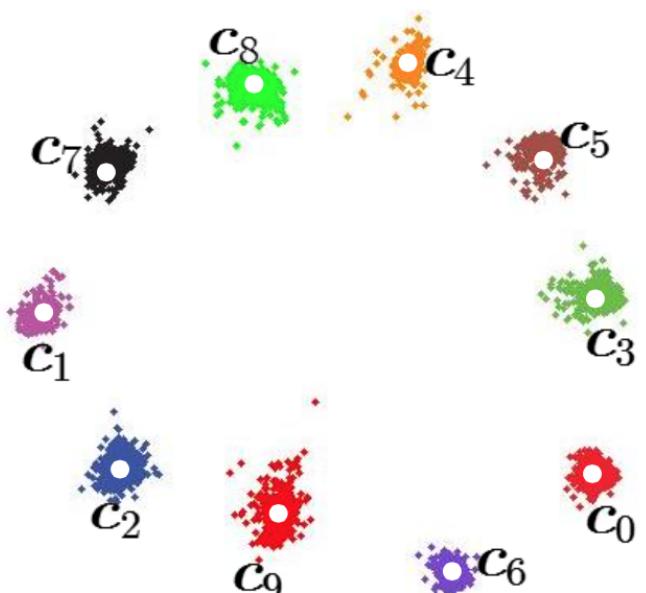


Figure: Softmax + Centre Loss

Center Loss

Our own experiments using Centre Loss training for job descriptions + VET courses

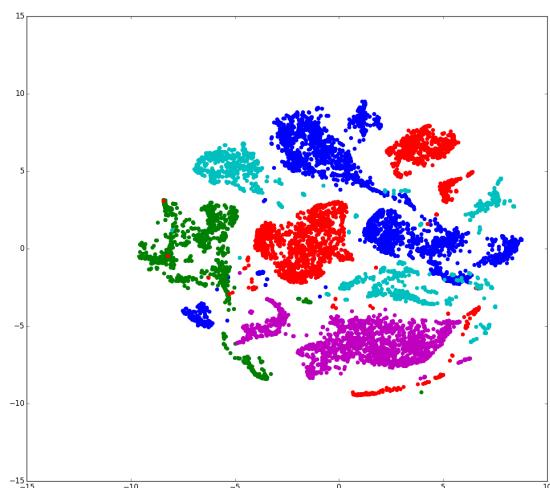


Figure: Softmax only

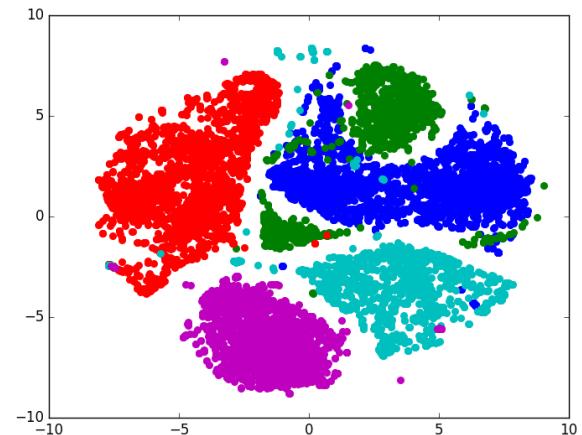


Figure: Softmax + Centre Loss

Support Vector Machine Layer

- ▶ Instead of Softmax layer, one may try SVM layer
- ▶ Tang, Yichuan. "Deep learning using linear support vector machines (2013)": author claimed it works better on certain dataset
- ▶ Soft Margin: allow "some" wrong sides:

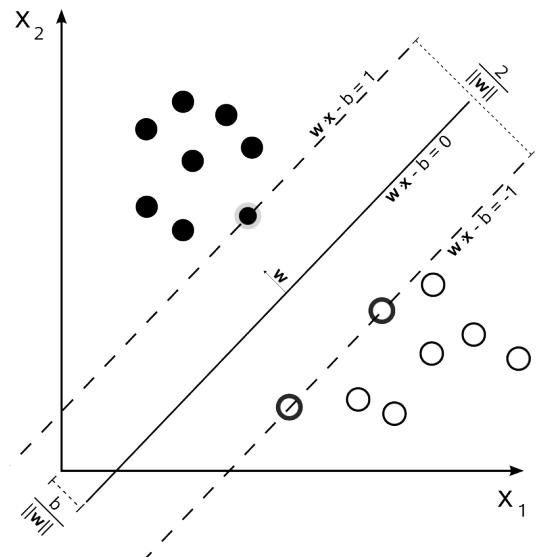
$$\min \left(\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \underbrace{\mathbf{1} [y_i(\mathbf{w}^T \mathbf{x}_i + b) < 0]}_{\text{sum of number of mis-classified}} \right)$$

- ▶ Add a convex upper bound to the indicator function:

$$\min \left(\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n h [y_i(\mathbf{w}^T \mathbf{x}_i + b)] \right)$$

- ▶ Hinge loss function $h(x)$:

$$h(x) = \begin{cases} 1 - x & 1 - x > 0 \\ 0 & \text{otherwise} \end{cases}$$



Linear Layer

- ▶ Deep Learning can be used for Regression: The last layer can be:

$$U^\top x^{(f)}$$

- ▶ In words, the preceding layers are “embedding the data”, such that the output from the final linear layer, $x_i^{(f)}$ and response variable y_i forms a linear relationship.

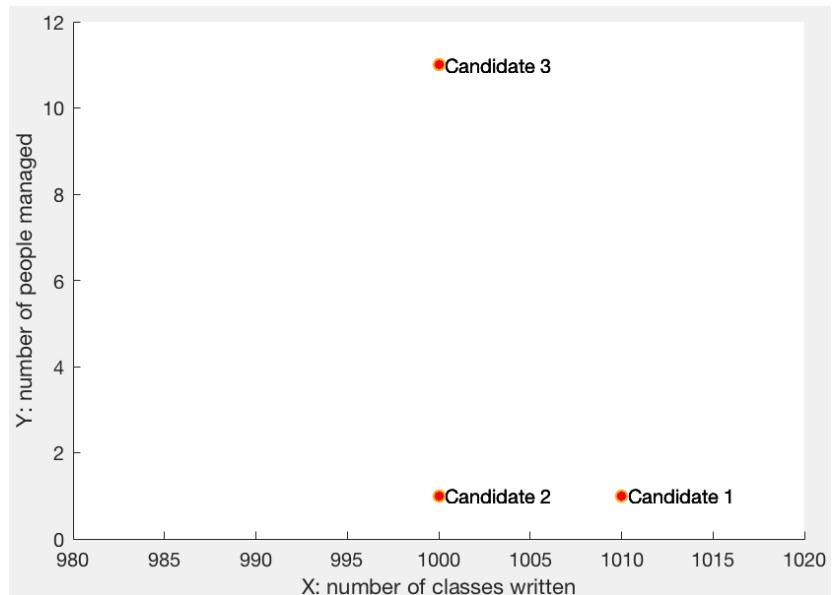
Is there any other task can be carried by Deep Learning dictated by using a different last layer?

- ▶ Let's look at **Learning a Similarity Metric**

Distance Metric Learning

To understand **Contrastive** or **Triplet** loss, we need to know **metric learning**:

- ▶ Suppose 3 candidates competing for technical lead role: (both manager and software skills are essential)
- ▶ You want to decide between candidate 1 and candidate 3, whom is more closely matched with candidate 2:
- ▶ candidate 1 and candidate 2, **differs 10 in (OO) classes of code.**
- ▶ candidate 3 and candidate 2, **differs 10 in number of people managed.**
- ▶ It is obvious that number of people managed should attract higher weight



$$d(\text{candidate}_i, \text{candidate}_j) = \left(\begin{bmatrix} X_i \\ Y_i \end{bmatrix} - \begin{bmatrix} X_j \\ Y_j \end{bmatrix} \right)^T \begin{bmatrix} w_1^{-1} & 0 \\ 0 & w_2^{-1} \end{bmatrix} \left(\begin{bmatrix} X_i \\ Y_i \end{bmatrix} - \begin{bmatrix} X_j \\ Y_j \end{bmatrix} \right)$$

Distance Metric Learning (2)

- ▶ In some situations, setting the weight matrix isn't intuitive, but under many scenarios, external labelling can be used:
- ▶ **triplet**

$$d_{\theta}(\text{[Image of Donald Trump]}, \text{[Image of Donald Trump]}) \leq d_{\theta}(\text{[Image of Donald Trump]}, \text{[Image of Kim Kardashian]})$$

- ▶ **contrastive**

$$(\text{[Image of Donald Trump]}, \text{[Image of Donald Trump]}) \in S \quad (\text{[Image of Donald Trump]}, \text{[Image of Bruno Mars]}) \in D$$

$$(\text{[Image of Kim Kardashian]}, \text{[Image of Kim Kardashian]}) \in S \quad (\text{[Image of Kim Kardashian]}, \text{[Image of Eminem]}) \in D$$

Traditional Metric Learning

A distance between two vectors x and y can be defined as:

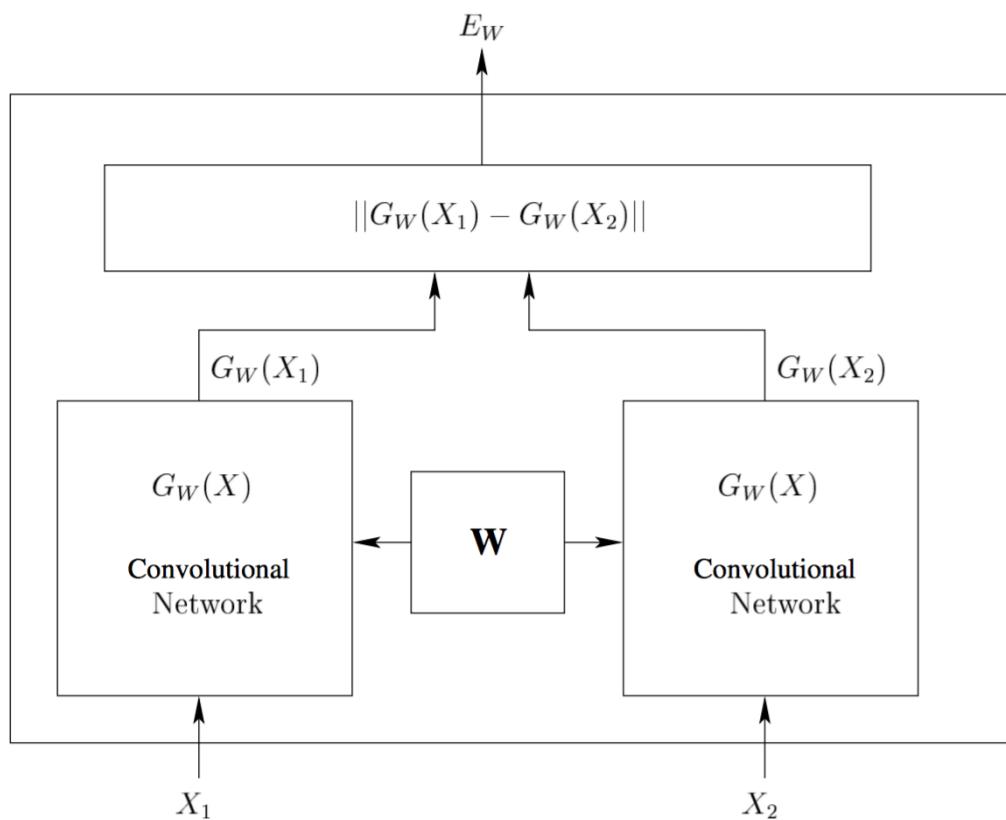
$$d(x, y) = d_W(x, y) = \|x - y\|_W = \sqrt{(x - y)^T W (x - y)}$$

where W is a positive semi-definite (PSD) matrix, or a metric:

$$\begin{aligned} & \min_W \sum_{(x_i, x_j) \in \mathcal{S}} \|x_i - x_j\|_W^2 \\ \text{s.t. } & \sum_{(x_i, x_j) \in \mathcal{D}} \|x_i - x_j\|_W^2 \geq c \end{aligned}$$

for some arbitrary scalar c .

Bring on the Deep Learning! The Siamese Network



Contrastive Loss

$$\mathcal{C}(X_i, X_j) = \frac{2}{m} \sum_{(i,j)}^m y_{i,j} \|G_W(X_i) - G_W(X_j)\| + (1 - y_{i,j}) \max(0, \alpha - \|G_W(X_i) - G_W(X_j)\|)^2$$

Training process encourages the network to find an embedding where:

- ▶ $y_{i,j} \in \{0, 1\}$ indicates whether a pair (x_i, x_j) is from the same class or not
- ▶ Minimises the distance between a pair of examples with the same class label
- ▶ penalises the negative pair distances for being smaller than the margin α
- ▶ $\max(0, .)$ is called the **hinge-loss**

Triplet Loss

$$\mathcal{C}(X_i, X_j) = \frac{2}{m} \sum_i^m \max(0, \|G_W(X_i^a) - G_W(X_i^p)\| - \|G_W(X_i^a) - G_W(X_i^n)\| + \alpha)$$

Training process encourages the network to find an embedding where:

- ▶ the distance between x_a and x_n cost function is larger than the distance between x_a and x_p plus some margin α

What other examples of Loss function are available?

- ▶ Energy Loss

$$\mathcal{L}(Y_i, \mathbb{E}(W, Y, X_i)) = \mathbb{E}(W, Y_i, X_i)$$

- ▶ Generalized Perceptron Loss

$$\mathcal{L}(Y_i, \mathbb{E}(W, Y, X_i)) = \mathbb{E}(W, Y_i, X_i) - \min_{y \in \mathcal{Y}} \mathbb{E}(W, Y, X_i)$$

- ▶ Let $\bar{Y} = \arg \min_{Y \in \mathcal{Y}, Y=Y_i} \mathbb{E}(W, Y, X_i)$, the label that has the lowest energy among all answers that are incorrect
- ▶ Generalized Margin Losses - Hinge loss:

$$\mathcal{L}(Y_i, \mathbb{E}(W, Y, X_i)) = \max(0, m + \mathbb{E}(W, Y_i, X_i) - \mathbb{E}(W, \bar{Y}_i, X_i))$$

Penalized linearly when $E(W, \bar{Y}_i, X_i) - E(W, Y_i, X_i) < m$. Loss only depends on energy differences, hence individual energies are not constrained to take any particular value.

- ▶ Generalized Margin Losses - Log loss:

$$\log \left(1 + \exp^{\mathbb{E}(W, Y_i, X_i) - \mathbb{E}(W, \bar{Y}_i, X_i)} \right)$$

This is a softer version of hinge loss.

All about that loss: Negative Log-Likelihood Loss

- ▶ A Gibbs distribution is defined as:

$$p(Y|X) = \frac{p(X, Y)}{p(X)} = \frac{\exp^{-\beta \mathbb{E}(Y, X)}}{\int_{y \in \mathcal{Y}} \exp^{-\beta \mathbb{E}(Y, X)}}$$

- ▶ Energy can measure in arbitrary units, but the fraction can cancel the effect.

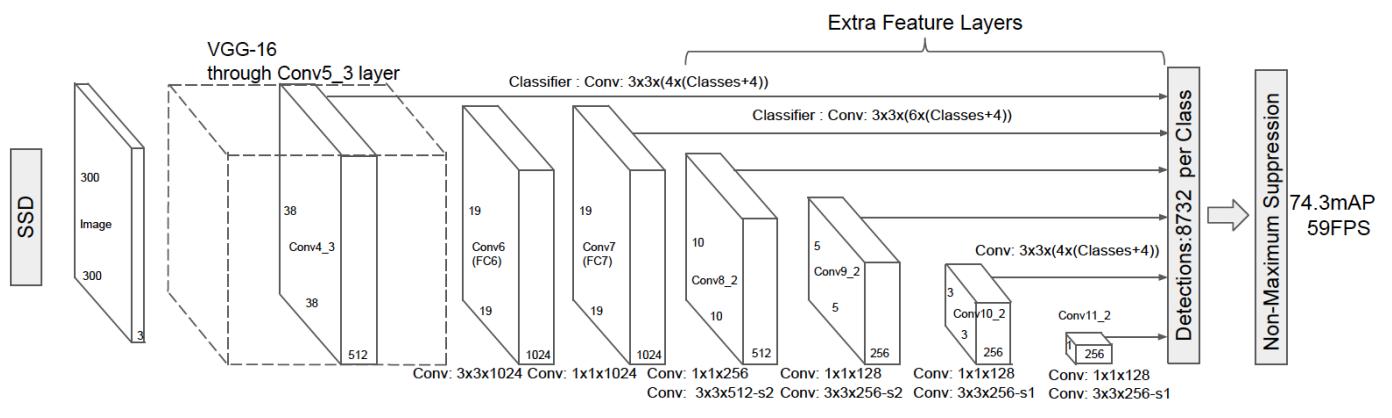
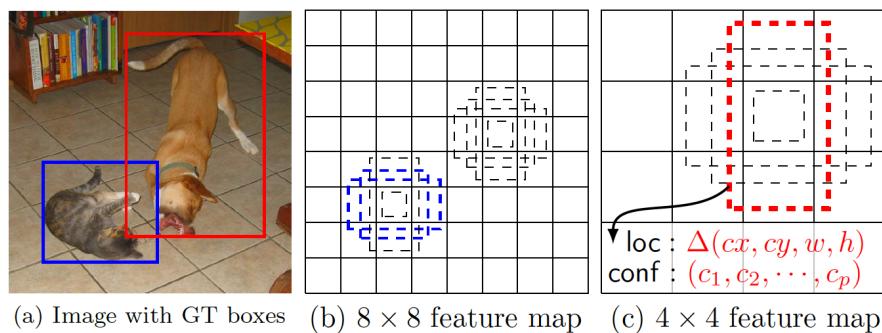
$$\begin{aligned} -\log \prod_{i=1}^N p(Y_i | X_i, W) &= -\sum_{i=1}^N \log (p(Y_i | X_i, W)) \\ &= -\sum_{i=1}^N \log \left(\frac{\exp^{-\beta \mathbb{E}(Y_i, X_i | W)}}{\int_{y \in \mathcal{Y}} \exp^{-\beta \mathbb{E}(Y, X_i | W)}} \right) \\ &= -\sum_{i=1}^N \log \left(\exp^{-\beta \mathbb{E}(Y_i, X_i | W)} \right) + \sum_{i=1}^N \int_{y \in \mathcal{Y}} \exp^{-\beta \mathbb{E}(Y, X_i | W)} \\ &= \sum_{i=1}^N \beta \mathbb{E}(Y_i, X_i | W) + \sum_{i=1}^N \int_{y \in \mathcal{Y}} \exp^{-\beta \mathbb{E}(Y, X_i | W)} \end{aligned}$$

In the previous example, $\beta = -1$
When $\beta > 0$:

$$\mathcal{C}(W) = \frac{1}{N} \left(\sum_{i=1}^N \mathbb{E}(Y_i, X_i | W) + \frac{1}{\beta} \int_{y \in \mathcal{Y}} \exp^{-\beta \mathbb{E}(Y, X_i | W)} \right)$$

CNN for detection: Single Shot MultiBox Detector (SSD)

Liu., et. al., (2016) SSD: Single Shot MultiBox Detector, ECCV2016



CNN for detection: Single Shot MultiBox Detector (SSD)

- ▶ $x_{i,j}^p = \{1, 0\}$ indicator for matching i -th **default box** to j -th **ground truth box** of category p
- ▶ $I \equiv (I^{cx}, I^{cy}, I^w, I^h)$: predicted box (transformed representation)
- ▶ $d \equiv (d^{cx}, d^{cy}, d^w, d^h)$: default box (original representation)
- ▶ $g \equiv (g^{cx}, g^{cy}, g^w, g^h)$: ground-truth box (original representation)
- ▶ $\hat{g} \equiv (\hat{g}^{cx}, \hat{g}^{cy}, \hat{g}^w, \hat{g}^h)$: ground-truth box (transformed representation)

Training objective:

$$\mathcal{C}(x, c, I, g) = \frac{1}{N} (\mathcal{C}_{\text{conf}}(x, c) + \alpha \mathcal{C}_{\text{loc}}(x, I, g))$$

- ▶ $\mathcal{C}_{\text{conf}}(x, c)$:

$$\mathcal{C}_{\text{conf}}(x, c) = - \sum_{i \in \text{pos}} x_{i,j}^p \log(\hat{c}_i^p) - \sum_{i \in \text{neg}} \log(\hat{c}_i^0) \quad \text{where } \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

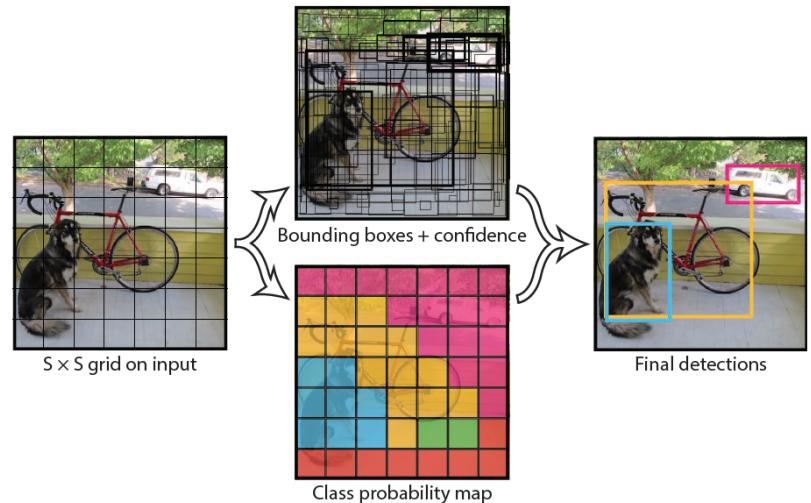
- ▶ $\mathcal{C}_{\text{loc}}(x, I, g)$:

$$\mathcal{C}_{\text{loc}}(x, I, \hat{g}) = \sum_{i \in \text{pos}} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(I_i^m - \hat{g}_j^m)$$

- ▶ all the parameters making them happen are **convolution filters**

CNN for detection: You Only Look Once (YOLO)

- ▶ divides input image into an $S \times S$ grid
- ▶ if center of object falls into a grid cell, that grid cell is responsible for detecting that object
- ▶ each bounding box consists of 5 predictions: (x, y, w, h, C)
- ▶ (x, y) centre of box relative to bounds of grid cell
- ▶ (w, h) weight and height of bound relative to the whole image
- ▶ C confidence prediction:
Intersection Over Union (IOU)
between predicted box and any ground truth box



CNN for detection: You Only Look Once (YOLO)

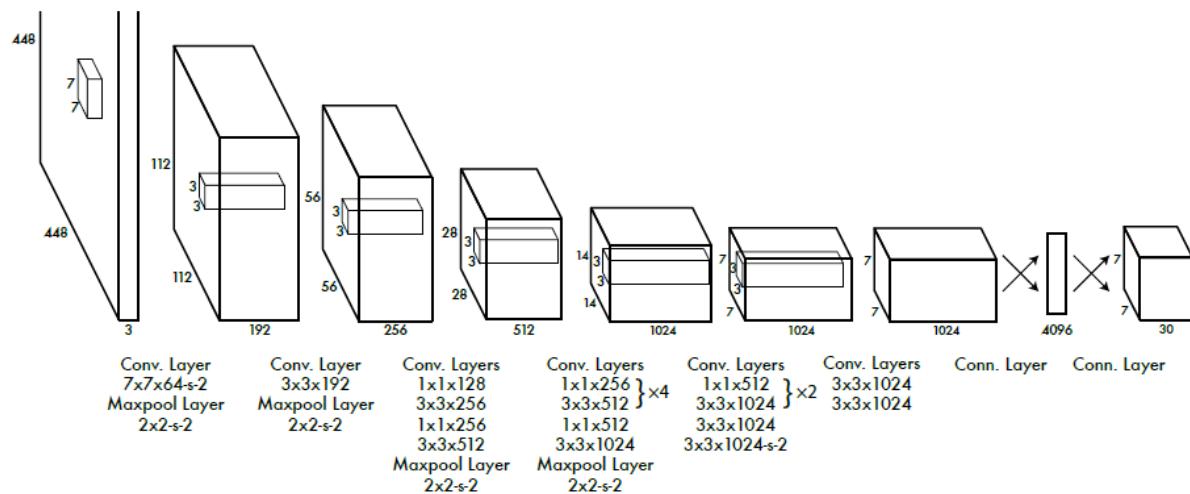
Redmon, et. al., (2016) *You Only Look Once: Unified, Real-Time Object Detection*, ICCV

- ▶ objective function:

$$\begin{aligned} \mathcal{C} = & \lambda_{\text{coord}} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbf{1}_{i,j}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbf{1}_{i,j}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbf{1}_{i,j}^{\text{obj}} \left[(C_i - \hat{C}_i)^2 \right] \\ & + \lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbf{1}_{i,j}^{\text{noobj}} \left[(C_i - \hat{C}_i)^2 \right] \\ & + \sum_{i=1}^{S^2} \mathbf{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

- ▶ $\mathbf{1}_{i,j}^{\text{obj}}$: j -th bounding box predictor in cell i is responsible for that prediction
- ▶ $\mathbf{1}_i^{\text{obj}}$: if object appears in cell i

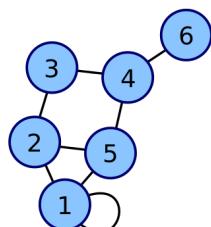
CNN for detection: You Only Look Once (YOLO)



degree matrix

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the associated degree matrix D is a $n \times n$ diagonal matrix:

$$d_{i,j} := \deg(v_i) \text{ if } i = j, \quad d_{i,j} := 0 \text{ otherwise}$$



$$\begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- ▶ $\deg(v_i)$ counts the number of times an edge terminates at that vertex
- ▶ in undirected graph, each new loop increases degree of vertex by **two**:
- ▶ see node 1 above
- ▶ **simple graph** as opposed to a **multigraph**, is an undirected graph in which both multiple edges and loops are disallowed
- ▶ in a simple graph with n vertices, the degree of every vertex is at most $n - 1$

matrix representation of a graph

- ▶ Given a simple graph \mathcal{G} with n vertices, its Laplacian matrix $L^{n \times n}$:

$$L = D - A$$

- ▶ D is the degree matrix and A is the adjacency matrix of the graph.
- ▶ Since \mathcal{G} is a simple graph, A only contains 1s or 0s and its diagonal elements are all 0s.
- ▶ Symmetric normalized Laplacian:

$$L^{\text{sym}} := D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

$$L_{i,j}^{\text{sym}} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

CNN for Graph

- ▶ undirected and connected graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$
- ▶ $W \in R^{n \times n}$ is a weighted adjacency
- ▶ **diagonal degree matrix:** $D \in R^{n \times n}$ is the diagonal degree matrix with $D^{ii} = \sum_j W_{ij}$

essential operator in spectral graph analysis is **graph Laplacian** L :

- ▶ **combinatorial definition** $L = D - W \in R^{n \times n}$
- ▶ **normalized definition** is $L = I_n - D^{-1/2}WD^{-1/2}$