

Recent Research on Convolution Neural Network (circa 2018)

A/Prof Richard Yi Da Xu

Yida.Xu@uts.edu.au

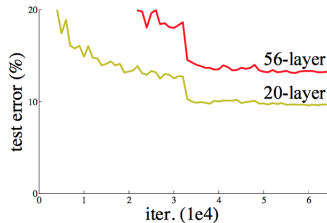
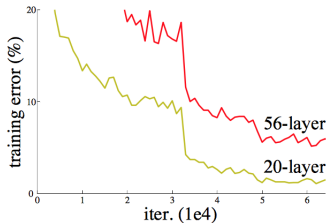
Wechat: aubedata

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

University of Technology Sydney (UTS)

August 16, 2022

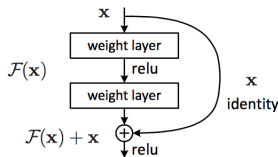
New network designs: Deep Residual Learning



- ▶ 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.

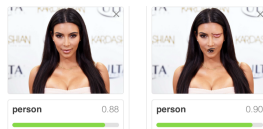
34-layer residual



One way we may be able to argue its advantage

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

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



neural network

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

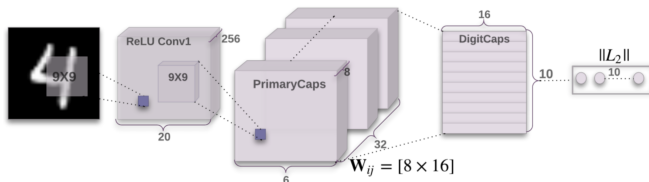
capsule layer

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

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

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

Capsule Networks - diagram



- ▶ Each PrimaryCaps contains 8 convolutional units with a 9×9 kernel and stride 2:
- ▶ in its **loss function**:
 - ▶ it doesn't use **softmax** loss
 - ▶ its loss function is:

$$L_c = T_c \max(0, m^+ - \|\mathbf{v}_c\|)^2 + \lambda(1 - T_c) \max(0, \|\mathbf{v}_c\| - m^-)^2$$

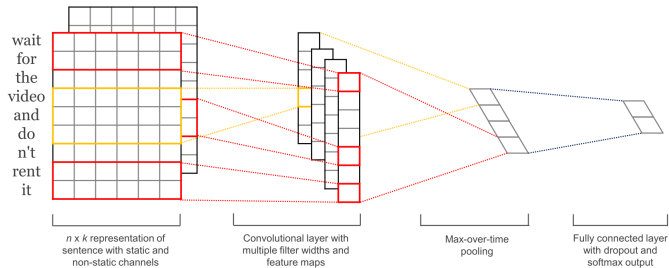
- ▶ $m^+ = 0.9$ and $m^- = 0.1$:
- ▶ it penalise correctly classified data if it's below 0.9
- ▶ it penalise incorrectly classified data if it's above 0.1

Procedure 1 Routing algorithm.

```
1: procedure ROUTING( $\hat{\mathbf{u}}_{j|i}$ ,  $r$ ,  $l$ )
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  $l$ :  $\mathbf{c}_i \leftarrow \text{softmax}(\mathbf{b}_i)$  ▷ 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  $(l + 1)$ :  $\mathbf{v}_j \leftarrow \text{squash}(\mathbf{s}_j)$  ▷ 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

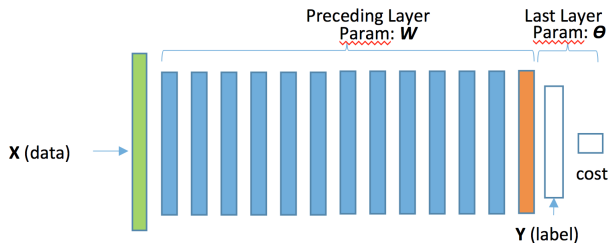


exercise why can't we perform 2-D in text classification?

exercise 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

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

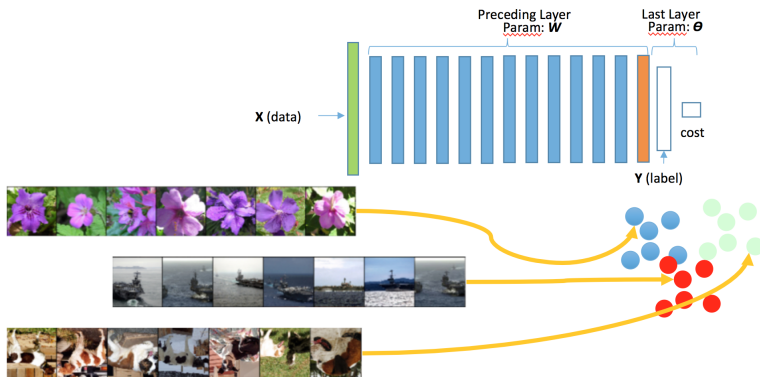
- ▶ Used for classification (Most commonly used)
- ▶ See `soft_max.m` where Softmax is applied **without** the preceding networks.
- ▶ Just for fun: $\arg \min (\mathcal{C}(\theta)) \implies \arg \max (\exp(-\mathcal{C}(\theta)))$:

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

- ▶ 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.

- ▶ 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:

$$\frac{\partial \mathcal{L}^t}{\partial x_i^t} = \frac{\partial \mathcal{L}_s^t}{\partial x_i^t} + \lambda \frac{\partial \mathcal{L}_c^t}{\partial x_i^t} \qquad W^{t+1} = W^t - \mu^t \frac{\partial \mathcal{L}^t}{\partial W^t} = W^t - \mu^t \frac{\partial \mathcal{L}_s^t}{\partial W^t}$$

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**

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

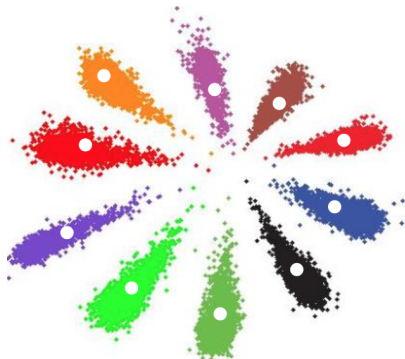


Figure: Softmax only

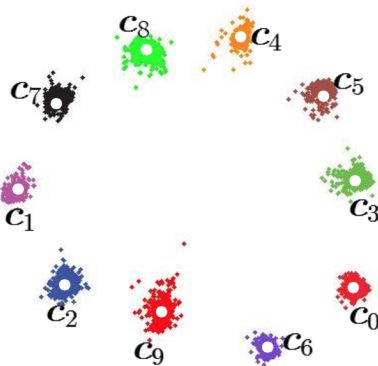


Figure: Softmax + Centre Loss

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

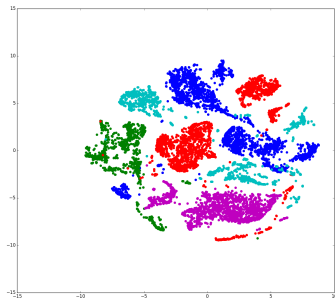


Figure: Softmax only

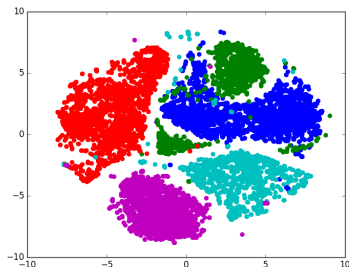


Figure: Softmax + Centre Loss

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:

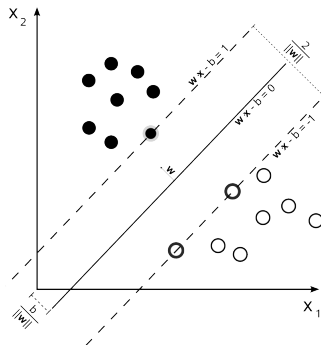
$$\min \left(\frac{1}{2} \|\mathbf{w}\|^2 + C \underbrace{\sum_{i=1}^n \mathbf{1} [y_i(\mathbf{w}^T x_i + b) < 0]}_{\text{sum of number of mis-classified}} \right)$$

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

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

- ▶ Hinge loss function $h(x)$:

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



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

$$U^T x^{(f)}$$

- ▶ 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.

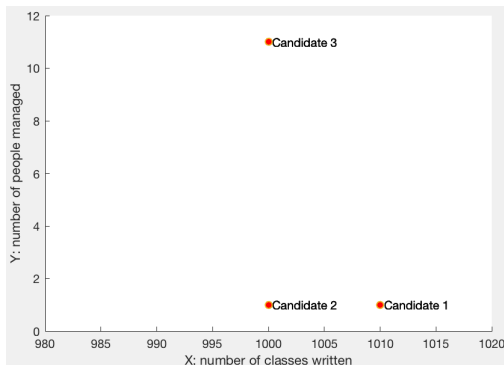
Is there any other task can be carried by Deep Learning dictated by using a different last layer?

- ▶ 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**

$$d_{\theta}(\text{img}_1, \text{img}_2) \leq d_{\theta}(\text{img}_1, \text{img}_3)$$

- ▶ **contrastive**

$$\begin{aligned} & \left(\text{img}_1, \text{img}_2 \right) \in S \quad \left(\text{img}_1, \text{img}_3 \right) \in D \\ & \left(\text{img}_4, \text{img}_5 \right) \in S \quad \left(\text{img}_4, \text{img}_6 \right) \in D \end{aligned}$$

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

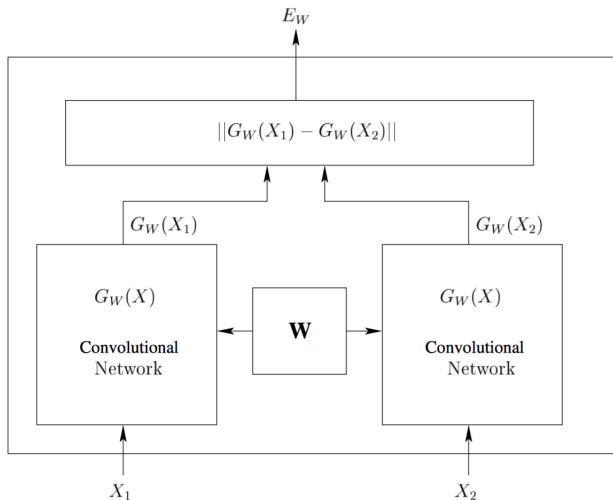
$$d(x, y) = d_W(x, y) = \|x - y\|_W = \sqrt{(x - y)^T W (x - y)}$$

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

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

for some arbitrary scalar c .

Bring on the Deep Learning! The Siamese Network



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

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**

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

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

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

- ▶ Generalized Perceptron Loss

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

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

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

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

- ▶ Generalized Margin Losses - Log loss:

$$\log(1 + \exp^{\mathcal{E}(W, Y_i, X_i) - \mathcal{E}(W, \tilde{Y}_i, X_i)})$$

This is a softer version of hinge loss.

All about that loss: Negative Log-Likelihood Loss

- ▶ A Gibbs distribution is defined as:

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

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

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

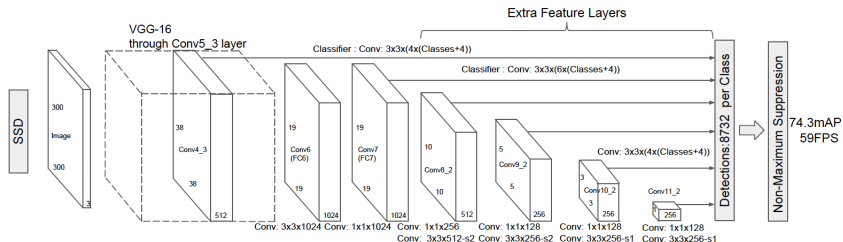
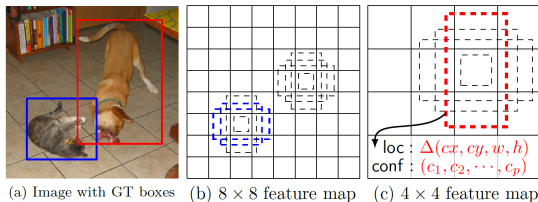
In the previous example, $\beta = -1$

When $\beta > 0$:

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

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
- ▶ $l \equiv (l^{cx}, l^{cy}, l^w, l^h)$: 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:

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

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

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

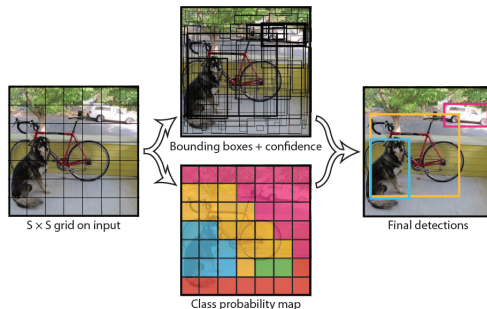
- ▶ $\mathcal{C}_{\text{loc}}(x, l, g)$:

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

- ▶ 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)

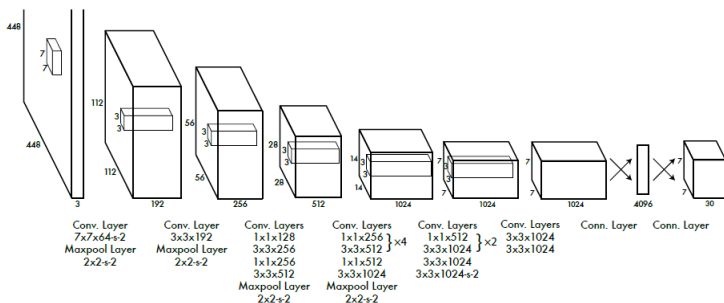
Redmon, et. al., (2016) *You Only Look Once: Unified, Real-Time Object Detection*, ICCV

- ▶ objective function:

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

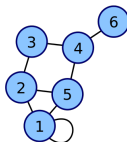
- ▶ $\mathbf{1}_{i,j}^{\text{obj}}$: j -th bounding box predictor in cell i is responsible for that prediction
- ▶ $\mathbf{1}_i^{\text{obj}}$: if object appears in cell i

CNN for detection: You Only Look Once (YOLO)



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

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

$$L^{\mathcal{G}} = D - A$$

- ▶ D is the degree matrix and A is the adjacency matrix of the graph.

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

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

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

About Graph Laplacian

- ▶ For a twice differentiable function f on the euclidean space, the Laplacian of f , Δf is **div(grad(f))** (divergence of the gradient of f)
- ▶ consider Laplacian in two dimensions:

$$\Delta f(x, y) = \nabla \cdot \nabla f(x, y) = \frac{d^2 f(x, y)}{dx^2} + \frac{d^2 f(x, y)}{dy^2}$$

- ▶ knowing that in 1-d, second derivative is approximated by:

$$f''(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

- ▶ then we can see the approximation of

$$\begin{aligned} \Delta f(x, y) &= \lim_{t \rightarrow 0} \frac{f(x+h, y) + f(x-h, y) + f(x, y+h) + f(x, y-h) - 4f(x, y)}{h^2} \\ &\approx \frac{f(x+h, y) + f(x-h, y) + f(x, y+h) + f(x, y-h) - 4f(x, y)}{h^2} \end{aligned}$$

- ▶ this means that $\Delta f(x, y)$ is a **local gradient averaging operator**, is zero at a smooth image

- ▶ Consider a function (or vector) $\mathbf{f}$ over vertices $\mathcal{V}$, $\mathbf{f} : \mathcal{V} \rightarrow \mathbb{R}$. This function is indexed by vertices, say v_i is the node value for i^{th} node, i.e., $\mathbf{f}$ is a vector, for example:

$$\mathbf{f} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

- ▶ then the product $\nabla \mathbf{f}$ is indexed by edges $\mathcal{E}$:
- ▶ each element of $\nabla \mathbf{f}$ is the difference between two end points of an edge e i.e. $v_i - v_j$:

$$\nabla \mathbf{f} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} v_1 - v_2 \\ v_2 - v_3 \end{bmatrix}$$

- ▶ Matrix ∇ acts like a difference or gradient operator on $\mathcal{G}$

- ▶ Consider a real function $\mathbf{g}$ over edges $\mathcal{E}$, $\mathbf{g} : \mathcal{E} \rightarrow \mathbb{R}$ indexed by edges.
- ▶ then, the product $\nabla^T \mathbf{g}^T$ is a vector indexed by $\mathcal{V}$.
- ▶ Value at i^{th} vertex:

$$(\mathbf{g}\nabla)_i = \sum_{e \text{ exit node } i} g_e - \sum_{e \text{ enters node } i} g_e$$

- ▶ $(\mathbf{g}\nabla)_i$ is the net outbound flow on the vertex v which is **divergence**

combine together

- ▶ Consider the quantity $\nabla^T \nabla \mathbf{f}$
- ▶ this is divergence of the gradient.
- ▶ the matrix $L^{\mathcal{G}} = \nabla^T \nabla$ is the **Graph Laplacian**:

$$\begin{aligned}\nabla^T \nabla &= \begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & 1 & -1 \end{bmatrix} \\ \Rightarrow \nabla^T \nabla \mathbf{f} &= \begin{bmatrix} 1 & -1 & 0 \\ 1 & 2 & -1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} v_1 - v_2 \\ 2v_2 + v_1 - v_3 \\ v_2 - v_3 \end{bmatrix}\end{aligned}$$

- ▶ for each i, j element of $L^{\mathcal{G}}$:

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

- ▶ easy to see that $L^{\mathcal{G}} = D - A$:
- ▶ Laplacian is positive semidefinite:

$$\mathbf{f}^T \nabla^T \nabla \mathbf{f} = \|\nabla \mathbf{f}\|^2 = \sum_{(i,j) \in \mathcal{E}} (\mathbf{f}_i - \mathbf{f}_j)^2$$

Spectral Decomposition

- ▶ A Laplacian is written as:

$$L^{\mathcal{G}} \phi_k = \lambda_k \phi_k$$

- ▶ eigenvectors:

$$\langle \phi_k, \phi_{k'} \rangle = \delta_{k,k'}$$

- ▶ eigenvalues are non-negative:

$$0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$$

- ▶ eigendecomposition of graph Laplacian:

$$L^{\mathcal{G}} = \Phi^{\top} \Lambda \Phi$$

where:

$$\Phi = [\phi_1, \dots, \phi_n] \quad \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$$

- **traditional** Fourier series of a function:

$$f(x) = \sum_{k \geq 0} \underbrace{\frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) \exp^{-ikt} dt}_{\hat{f}_k = \langle \mathbf{f}, \exp^{-ikx} \rangle \text{ coefficient}} \underbrace{\exp^{-ikx}}_{\text{Fourier basis}}$$

- **graph** Fourier series: for the i^{th} element:

$$f_i = \sum_{k=1}^n \underbrace{\langle f, \phi_k \rangle}_{\hat{f}_k \text{ coefficient}} \underbrace{\phi_{k,i}}_{\text{graph Fourier basis}}$$

in matrix notation:

$$\hat{\mathbf{f}} = \Phi^{\top} \mathbf{f} \quad \text{and} \quad \mathbf{f} = \Phi^{\top} \hat{\mathbf{f}}$$

Convolution in Euclidean space

- convolution theorem:

$$\widehat{\mathbf{f} \circledast \mathbf{g}} = \hat{\mathbf{f}} \odot \hat{\mathbf{g}}$$

- $\mathbf{f} = (f_1, \dots, f_n)^\top$ and $\mathbf{g} = (g_1, \dots, g_n)^\top$

$$\begin{aligned} (\mathbf{f} \circledast \mathbf{g})_i &= \sum_k g(i - k) \bmod n \cdot f_m \\ \Rightarrow \mathbf{f} \circledast \mathbf{g} &= \underbrace{\begin{bmatrix} g_1 & g_2 & \dots & \dots & g_n \\ g_n & g_1 & g_2 & \dots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \dots & g_1 & g_2 \\ g_2 & g_3 & \dots & \dots & g_1 \end{bmatrix}}_{\text{Circulant matrix}} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} \\ &= \Phi \underbrace{\begin{bmatrix} \hat{g}_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \hat{g}_n \end{bmatrix}}_{\Phi^\top \mathbf{g}} \Phi^\top \mathbf{f} \\ &= \Phi(\Phi^\top \mathbf{g} \odot \Phi^\top \mathbf{f}) \end{aligned}$$

- spectral convolution:

$$\begin{aligned}\mathbf{f} \circledast \mathbf{g} &= \Phi(\Phi^\top \mathbf{g} \odot \Phi^\top \mathbf{f}) \\ &= \underbrace{\Phi \text{diag}(\hat{g}_1, \dots, \hat{g}_n) \Phi^\top}_G \mathbf{f}\end{aligned}$$

What is missing?

- ▶ CNN has a pooling step
- ▶ GraphCNN has Graph downsampling(**coarsening**) step