ECE 5582 Computer Vision

Lec 18: CNN based Image Analysis II - Classification & Identification

Zhu Li
Dept of CSEE, UMKC
Office: FH560E, Email: lizhu@umkc.edu, Ph: x 2346.
http://l.web.umkc.edu/lizhu
Outline

- Recap of Lec 17
- Loss Functions
  - Classification - SoftMax loss networks
  - Identification - Triplet loss networks
- Training of CNN
- Summary
We can generate successive convolution features into higher level of representation: (notice w/o padding, shrinking)

- This gives us low level to high level features.

- Deeper feature, has larger receptive field, i.e., how many pixels it derives from.
A landmark work:
- Conv layers generate w x h x k feature maps
- FC layers map features to vectors

How is label prediction done from final fc6 84-dimensional feature?
- SoftMax loss function
Classification - kNN search

- Find the k-nearest neighbors, and use majority rule to assign label

- not a differentiable process
Linear Classifier

- Cut a line in Feature (2x2 pixels) Space

\[ f(x, W) = Wx + b \]

- How do we define a Loss Function to select the best label?
  - \( \text{max}(f(x, W)) \)?
Linear Classifier Loss Function

- Loss Function:
  - $f(x, W)$ with 4 pixel input and 3 labels give us this score matrix:

  \[
  \begin{array}{ccc}
  x_1 & x_2 & x_3 \\
  \text{cat} & 3.2 & 1.3 & 2.2 \\
  \text{car} & 5.1 & 4.9 & 2.5 \\
  \text{frog} & -1.7 & 2.0 & -3.1 \\
  \end{array}
  \]

  - How do we define a loss function such that the best label is predicted from minimizing a (differentiable) loss function?
Given the labeled data set, classifier performance is evaluated by the loss in correctly recognizing labels.

Given a dataset of examples
\[ \{(x_i, y_i)\}_{i=1}^{N} \]

Where \( x_i \) is image and \( y_i \) is (integer) label.

Loss over the dataset is a sum of loss over examples:
\[ L = \frac{1}{N} \sum_{i} L_i(f(x_i, W), y_i) \]
For classification loss function, mapping the scores to a probability function:

- Network observing pixels in $x$
- Output score $s = f(x; W)$

**Softmax Classifier** (Multinomial Logistic Regression)

Scores = unnormalized log probabilities of the classes.

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

where $s = f(x_i; W)$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = - \log P(Y = y_i | X = x_i)$$

<table>
<thead>
<tr>
<th>Class</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
Soft Max Loss Function

- Softmax function highlights the largest value, suppressing others.
- Softmax is a differentiable function.
Softmax Loss in operation

- Take pixels in and output softmax prob:
  - the prediction is achieved by selecting the highest prob prediction

\[
k^* = \arg \max_k -\log \left( \frac{e^{S_k}}{\sum_j e^{S_j}} \right)
\]
Wining solutions for ImageNet in 2012

- Similar framework to LeCun’98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data ($10^6$ vs. $10^3$ images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week

A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
Using CNN for Image Classification

- Fixed input size: 224x224x3
- Averaging
- Fully connected layer Fc7
  - $d = 4096$
- Softmax Layer
- Image Ids

Data set augmentation via *rand cropping*

AlexNet
VGG-Net

- The deeper, the better

- Key design choices:
  - 3x3 conv. Kernels
    - very small
  - conv. stride 1
    - no loss of information

- Other details:
  - Rectification (ReLU) non-linearity
  - 5 max-pool layers (x2 reduction)
  - no normalization
  - 3 fully-connected (FC) layers

- MatConvNet:
  - has pretrained models, VGG16, VGG19

FC-4096 is an embedding like Laplacian Face
VGG-Net

- Why 3x3 layers?
  - Stacked conv. layers have a large receptive field
  - two 3x3 layers – 5x5 receptive field
  - three 3x3 layers – 7x7 receptive field

- More non-linearity
  - Less parameters to learn: 3x3 filters
  - ~140M per net
### ResNet

- **Can we just increase the #layer?**

- **How can we train very deep network?**
  - Residual learning

<table>
<thead>
<tr>
<th>method</th>
<th>top-5 err. (test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG [41] (ILSVRC’14)</td>
<td>7.32</td>
</tr>
<tr>
<td>GoogLeNet [44] (ILSVRC’14)</td>
<td>6.66</td>
</tr>
<tr>
<td>VGG [41] (v5)</td>
<td>6.8</td>
</tr>
<tr>
<td>PReLU-net [13]</td>
<td>4.94</td>
</tr>
<tr>
<td>BN-inception [16]</td>
<td>4.82</td>
</tr>
<tr>
<td><strong>ResNet (ILSVRC’15)</strong></td>
<td><strong>3.57</strong></td>
</tr>
</tbody>
</table>
DenseNet

- Shorter connections (like ResNet) help
- Why not just connect them all?
Progress on ImageNet

- Deeper networks can give wider feature sets in modeling
  - Image level label prediction has been relatively mature
  - Rivaling human performance

<table>
<thead>
<tr>
<th>Year</th>
<th>Model</th>
<th>Top5 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>AlexNet</td>
<td>16.4</td>
</tr>
<tr>
<td>2013</td>
<td>ZF</td>
<td>11.7</td>
</tr>
<tr>
<td>2014</td>
<td>VGG</td>
<td>7.3</td>
</tr>
<tr>
<td>2014</td>
<td>GoogLeNet</td>
<td>6.7</td>
</tr>
<tr>
<td>2015</td>
<td>ResNet</td>
<td>3.57</td>
</tr>
<tr>
<td>2016</td>
<td>GoogLeNet-v4</td>
<td>3.08</td>
</tr>
</tbody>
</table>

ImageNet Image Classification Top5 Error
Loss Function at Pixel Level

- For tasks like denoising, super resolution, the ground truth is labeled at pixel level, how do we define loss function in this case?

![Low res input](image1)

![Bilinear scaling](image2)

![Desired high res image](image3)
VDSR Work [Prof. Kyungmu Lee, SNU]

- Residual learning with pixel loss in L2 norm

Exploits very deep CNN to SR
- 20 convolutional layers (41x41 receptive field)
- 64 channels, 3x3 filters in each convolutional layer
- Skip connection to learn residual only
- No dimension reduction such as pooling
Pixel Level Loss Function

Given an image patch in the input side, the residual is pixel level loss

- a bicubic upsampled image is prediction,
- the residual to be-learn is the difference between the ground truth \( \{y_j\} \) and predicted image.

\[
L = \sum_j (x_j - y_j)^2 \\
x_j = f(u_j; W)
\]
Identification Problem

- Face recognition

- Landmark recognition

- Limited samples (N) vs large number of labels (M)

<table>
<thead>
<tr>
<th>No.</th>
<th>Aim</th>
<th># Persons</th>
<th>Total # images</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Labeled Faces In the Wild</td>
<td>5,749</td>
<td>13,233</td>
</tr>
<tr>
<td>2</td>
<td>WDRef</td>
<td>2995</td>
<td>99,773</td>
</tr>
<tr>
<td>3</td>
<td>Celeb Faces</td>
<td>10177</td>
<td>202,599</td>
</tr>
<tr>
<td>4</td>
<td>Ours</td>
<td>2622</td>
<td>1,635,159</td>
</tr>
<tr>
<td>5</td>
<td>Facebook</td>
<td>4030</td>
<td>4.4M</td>
</tr>
<tr>
<td>6</td>
<td>Google</td>
<td>8M</td>
<td>200M</td>
</tr>
</tbody>
</table>
One (a few) Shot Learning

- Learn a distance metric/feature for retrieval from very few samples
- Learn what is differentiating among different objects
Triplet loss - Identification Problem

- When the sample per label, i.e, n/m is small, not enough training sample to move SoftMax gradient

FaceNet: A Unified Embedding for Face Recognition and Clustering, CVPR 2015
FaceNet

- A very deep triplet loss network
  - achieves the state of art in recognition performance

FaceNet – Deep Learning

- 22 layers:
  - 11 convolutions
  - 3 normalizations
  - 4 max-pooling
  - 1 concatenation
  - 3 fully-connected
- 140 million parameters

<table>
<thead>
<tr>
<th>layer</th>
<th>size-in</th>
<th>size-out</th>
<th>kernel</th>
<th>param</th>
<th>FLPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>220×220×3</td>
<td>110×110×64</td>
<td>7×7×3, 2</td>
<td>9K</td>
<td>115M</td>
</tr>
<tr>
<td>pool1</td>
<td>110×110×64</td>
<td>55×55×64</td>
<td>3×3×64, 2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>rnorm1</td>
<td>55×55×64</td>
<td>55×55×64</td>
<td>1×1×64, 1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>conv2a</td>
<td>55×55×64</td>
<td>55×55×64</td>
<td>3×3×64, 1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>conv2</td>
<td>55×55×64</td>
<td>55×55×192</td>
<td>3×3×192, 2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>rnorm2</td>
<td>55×55×192</td>
<td>55×55×192</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>pool2</td>
<td>55×55×192</td>
<td>28×28×192</td>
<td>3×3×192, 2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>conv3a</td>
<td>28×28×192</td>
<td>28×28×384</td>
<td>3×3×192, 1</td>
<td>37K</td>
<td>29M</td>
</tr>
<tr>
<td>pool3</td>
<td>28×28×384</td>
<td>14×14×384</td>
<td>3×3×192, 1</td>
<td>664K</td>
<td>521M</td>
</tr>
<tr>
<td>conv4a</td>
<td>14×14×384</td>
<td>14×14×384</td>
<td>1×1×384, 1</td>
<td>0</td>
<td>29M</td>
</tr>
<tr>
<td>conv4</td>
<td>14×14×384</td>
<td>14×14×256</td>
<td>3×3×384, 1</td>
<td>0</td>
<td>29M</td>
</tr>
<tr>
<td>conv5a</td>
<td>14×14×256</td>
<td>14×14×256</td>
<td>3×3×384, 1</td>
<td>0</td>
<td>29M</td>
</tr>
<tr>
<td>conv5</td>
<td>14×14×256</td>
<td>14×14×256</td>
<td>3×3×256, 1</td>
<td>590K</td>
<td>116M</td>
</tr>
<tr>
<td>conv6a</td>
<td>14×14×256</td>
<td>14×14×256</td>
<td>3×3×256, 1</td>
<td>0</td>
<td>13M</td>
</tr>
<tr>
<td>conv6</td>
<td>14×14×256</td>
<td>14×14×256</td>
<td>3×3×256, 1</td>
<td>590K</td>
<td>116M</td>
</tr>
<tr>
<td>pool4</td>
<td>14×14×256</td>
<td>7×7×256</td>
<td>3×3×256, 2</td>
<td>0</td>
<td>0.5M</td>
</tr>
<tr>
<td>concat</td>
<td>7×7×256</td>
<td>7×7×256</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>fc1</td>
<td>7×7×256</td>
<td>1×32×128</td>
<td>maxout p=2</td>
<td>103M</td>
<td>103M</td>
</tr>
<tr>
<td>fc2</td>
<td>1×32×128</td>
<td>1×32×128</td>
<td>maxout p=2</td>
<td>34M</td>
<td>34M</td>
</tr>
<tr>
<td>fc7128</td>
<td>1×32×128</td>
<td>1×1×128</td>
<td></td>
<td>524K</td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td>1×1×128</td>
<td>1×1×128</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

total  140M  1.6B

~0.73 sec per image @2.2GHZ CPU
Crop and scaling
Triplet loss
0.9963±0.09
For our project, it is a classification problem in its original form
- $n=15 \times 700 = 10500$ images
- $m=15$ categories

Use pre-trained VGG16 network

For the final conv feature $512 \times 7 \times 7$, treat them as $n=512$ features of $d=49$ dimensions, and train GMM models for each class to a set of $k_d=[8\ 16\ 32]$, $n_c=[24\ 48\ 64]$.

Optimizing the component turn on and off logics (See SCFV) to improve performance
For our project, it is a classification problem in its original form
- \( n = 15 \times 700 = 10500 \) images
- \( m = 15 \) categories
- use AlexNet to obtain a baseline

Label splitting via Embedding
- Same label “airport”, may have very different visual appearance
- Embedding with pre-trained AlexNet FC + PCA/LDA (\( kd = m - 1 = 14 \))
  build a kd-tree to assign secondary labels

Triplet loss replacing Softmax with AlexNet
- Triplet quality control with embedding + kdtree
- Compare results with SoftMax

This requires training of the network, total worth 40pts (1+0.25)
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ Recap of Lec 15</td>
</tr>
<tr>
<td>☐ Loss Functions</td>
</tr>
<tr>
<td>☐ Training of CNN</td>
</tr>
<tr>
<td>☐ Summary</td>
</tr>
</tbody>
</table>
Loss function gradient driven network weights update

- A generic CNN inference function with \( L \) layers:
  \[
  f_W(x) = f_L(...(f_2(f_1(x; W_1); W_2)...; W_L)
  \]

- Network weights to be learned
  \[
  W = (W_1, W_2, ..., W_L)
  \]

- Empirical Loss Function from a batch of training samples \( \{x_i, y_i\} \):
  \[
  L(W) = \frac{1}{n} \sum_i l(y_i, f_W(x_i))
  \]

- Gradient Descent:
  \[
  W^{(t+1)} = W^t - \eta_t \frac{\partial L(W^t)}{\partial W}
  \]
How to find best weights in the network?

- Backpropagation

**Key Idea: Wiggle To Decrease Loss**

Let's say we want to decrease the loss by adjusting $W_{i,j}^1$. We could consider a very small $\epsilon = 1e-6$ and compute:

$$L(x, y; \theta)$$

$$L(x, y; \theta \setminus W_{i,j}^1, W_{i,j}^1 + \epsilon)$$
Gradients propagates backwards

Backward Propagation

Given $\frac{\partial L}{\partial o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial W^3}$$

$$\frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2}$$
Backprop

- Gradient back prop

### Backward Propagation

Given $\frac{\partial L}{\partial h^2}$, we can compute now:

$$
\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial W^2} \\
\frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial h^1}
$$
First Layer:

Backward Propagation

Given $\frac{\partial L}{\partial h^1}$ we can compute now:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial h^1} \frac{\partial h^1}{\partial W^1}$$
Scalar Example

Consider the following network

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Scalar Backprop example

- Gradients analytical form:

- \( q = (x + y) \):
  \[
  \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1,
  \]

- \( f = qz \):
  \[
  \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q,
  \]

  \[
  \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} = \frac{\partial f}{\partial q} \cdot 1 = \frac{\partial f}{\partial q}.
  \]

  \[
  \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} = \frac{\partial f}{\partial q} \cdot 1 = \frac{\partial f}{\partial q}.
  \]
If we change the weights, what will loss function change?
Consider the simple network

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]
Consider the vector input network

A vectorized example: \[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \]

\[
\begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\rightarrow
\begin{bmatrix}
0.22 \\
0.26
\end{bmatrix}
\rightarrow
L2
\rightarrow
0.116
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

how to compute gradient?
Gradient Propagation

- Vector case

\[
\frac{\partial f}{\partial q_i} = 2q_i
\]

A vectorized example: 

\[
f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2
\]

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\frac{\partial f}{\partial q_i} = 2q_i
\]

\[
\nabla_q f = 2q
\]
Gradient Propagation

- Vector case

\[
\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\
= \sum_k (2q_k) (1_k = i x_j) \\
= 2q_i x_j
\]

A vectorized example: 
\[
f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W_i \cdot x)_i^2
\]

\[
\nabla_W f = 2q \cdot x^T
\]

\[
q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}
\]

\[
f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2
\]
Stochastic Gradient Descent (SGD)

- Given a CNN network, how do we find out the best weights for the network?

- Each training points \( \{x_i, y_i\} \) will give us a loss function and its gradient, how we use this to find the optimal solution?
  - gradients from training data set usually highly redundant, no dependence
  - so instead randomly group the gradients together to form minibatch to drive the gradients and updates.
SGD

- Convex optimization problem

![Diagram showing SGD with convex optimization problem]

- True negative gradient direction
- Original W
SGD

- SGD with minibatch:
  - group sample gradients together to find gradient direction

True negative gradient direction

original W

W_1

W_2
SGD

- Minibatch driven gradient movement
  - works okay though noisy
  - optimization strategy on how to form minibatch
  - biggest puzzle, convexity assumption

True gradients in blue
minibatch gradients in red

original W

W_1
Momentum Method

- Gradient momentum
  
  If the error surface is a tilted plane, the ball reaches a terminal velocity.
  - If the momentum is close to 1, this is much faster than simple gradient descent.

\[
v(\infty) = \frac{1}{1-\alpha} \left( -\varepsilon \frac{\partial E}{\partial w} \right)
\]

- At the beginning of learning there may be very large gradients.
  - So it pays to use a small momentum (e.g. 0.5).
  - Once the large gradients have disappeared and the weights are stuck in a ravine the momentum can be smoothly raised to its final value (e.g. 0.9 or even 0.99)

- This allows us to learn at a rate that would cause divergent oscillations without the momentum.
Momentum Method

- Keep a memory of gradient updates, and use that to derive momentum
- Use momentum to update network weights

\[ v(t) = \alpha v(t-1) - \varepsilon \frac{\partial E}{\partial w}(t) \]

The effect of the gradient is to increment the previous velocity. The velocity also decays by \( \alpha \) which is slightly less than 1.

\[ \Delta w(t) = v(t) \]

The weight change is equal to the current velocity.

\[ = \alpha v(t-1) - \varepsilon \frac{\partial E}{\partial w}(t) \]

\[ = \alpha \Delta w(t-1) - \varepsilon \frac{\partial E}{\partial w}(t) \]

The weight change can be expressed in terms of the previous weight change and the current gradient.
Nestrov Momentum

- Use gradients history to drive big step directions, and then followed by a correction with gradient from current minibatch
- Conjugate method?

- **First** make a big jump in the direction of the previous accumulated gradient.
- **Then** measure the gradient where you end up and make a correction.

brown vector = jump,  red vector = correction,  green vector = accumulated gradient

blue vectors = standard momentum
Optimization methods in Deep Learning

- More details in Prof. Jorge Nocedal's online coverage:
  - https://www.youtube.com/watch?v=dY249vF0Pps

- A more comprehensive treatment of this topic:
  - Guanghui Lan, Georgia Tech:

Guanghui (George) Lan

Lectures on Optimization Methods for Machine Learning
February 27, 2019
Different applications have different loss function, e.g. Softmax for classification, Triplet loss for identification, MSE for pixel level loss.

Loss function need to be differentiable, i.e., have analytical solution to recover gradient.

Network training is basically a numerical optimization problem with familiar tools like gradient, conjugate gradient, newtonian methods.

What works for CNN training: SGD and variations.

Rich software resource and tools.