CENG 501
Deep Learning
Week 10

Sinan Kalkan
Disadvantages of MLPs: **Curse of Dimensionality**

- Number of required samples for obtaining small error increases exponentially with input dimensions
- Too many parameters

Illustration of the curse of dimensionality: in order to approximate a Lipschitz-continuous function composed of Gaussian kernels placed in the quadrants of a d-dimensional unit hypercube (blue) with error $\epsilon$, one requires $\Theta(1/\epsilon^d)$ samples (red points).

Disadvantages of MLPs: **Equivariance**

- Vectorizing an image breaks patterns in consecutive pixels.
  - Shifting one pixel means a whole new vector
  - Makes learning more difficult
  - Requires more data to generalize

![Figure: http://cs231n.github.io/linear-classify/](http://cs231n.github.io/linear-classify/)
CNNs vs. MLPs: **Curse of Dimensionality**

When things go deep, an output may depend on all or most of the input:

---

Example multi-dimensional convolution

Input:

\[
\begin{bmatrix}
a & b & c & d \\
e & f & g & h \\
i & j & k & l \\
\end{bmatrix}
\]

Kernel:

\[
\begin{bmatrix}
w & x \\
y & z \\
\end{bmatrix}
\]

Output:

\[
\begin{bmatrix}
aw + bx + ey + fz \\
bw + cx + fy + gz \\
ew + fx + iy + jz \\
fw + gx + jy + kz \\
gew + hx + ky + lz \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
3_0 & 3_1 & 2_2 & 1_0 & 0 \\
0_2 & 0_2 & 1_0 & 3_1 & 1 \\
3_0 & 1_1 & 2_2 & 2_3 & 3 \\
2_0 & 0_0 & 2_2 & 2 & 2 \\
2_0 & 0_0 & 0_0 & 0_1 & 1 \\
\end{bmatrix}
\]

https://github.com/vdumoulin/conv_arithmetic


Sinan Kalkan
CNN layers

- Stages of CNN:
  - Convolution (in parallel) to produce pre-synaptic activations
  - Detector: Non-linear function
  - Pooling: A summary of a neighborhood

- Pooling of a rectangular region:
  - Max
  - Average
  - L2 norm
  - Weighted average acc. to the distance to the center
  - ...

Connectivity in CNN

Local: The behavior of a neuron does not change other than being restricted to a subspace of the input.

- Each neuron is connected to slice of the previous layer
- A layer is actually a volume having a certain width x height and depth (or channel)
- A neuron is connected to a subspace of width x height but to all channels (depth)
- Example: CIFAR-10
  - Input: 32 x 32 x 3 (3 for RGB channels)
  - A neuron in the next layer with receptive field size 5x5 has input from a volume of 5x5x3.
Important parameters

• Depth (number of channels)
  – We will have more neurons getting input from the same receptive field
  – This is similar to the hidden neurons with connections to the same input
  – These neurons learn to become selective to the presence of different signals in the same receptive field

• Stride
  – The amount of space between neighboring receptive fields
  – If it is small, RFs overlap more
  – If it is big, RFs overlap less

• How to handle the boundaries?
  i. Option 1: Don’t process the boundaries. Only process pixels on which convolution window can be placed fully.
  ii. Option 2: Zero-pad the input so that convolution can be performed at the boundary pixels.
Padding illustration

- Only convolution layers are shown.
- Top: no padding $\Rightarrow$ layers shrink in size.
- Bottom: zero padding $\Rightarrow$ layers keep their size fixed.

Figure 9.11: The effect of zero padding on network size. Consider a convolutional network with a kernel of width six at every layer. In this example, do not use any pooling, so only the convolution operation itself shrinks the network size. Top) In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not even move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive and some shrinking is inevitable in this kind of architecture. Bottom) By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

Size of the next layer

• Along a dimension:
  – \( W \): Size of the input
  – \( F \): Size of the receptive field
  – \( S \): Stride
  – \( P \): Amount of zero-padding

• Then: the number of neurons as the output of a convolution layer:
  \[
  \frac{W - F + 2P}{S} + 1
  \]

• If this number is not an integer, your strides are incorrect and your neurons cannot tile nicely to cover the input volume
Today

• Convolutional Neural Networks
  – Convolution types in CNNs
  – Other CNN Operations: pooling, nonlinearity, normalization, FC
  – CNN Architectures
  – Backpropagation in a CNN
  – Transfer learning
  – Visualizing and understanding CNNs

• Next week:
  – Widely-used CNN architectures
  – CNN Applications

<table>
<thead>
<tr>
<th>Week</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 10 (17 May)</td>
<td>CNNs</td>
</tr>
<tr>
<td>Week 11 (24 May)</td>
<td>CNNs</td>
</tr>
<tr>
<td>Week 12 (31 May)</td>
<td>RNNs</td>
</tr>
<tr>
<td>Week 13 (7 June)</td>
<td>RNNs</td>
</tr>
<tr>
<td>Week 14 (14 June)</td>
<td>RNNs + Attention</td>
</tr>
<tr>
<td>Week 15 (21 June)</td>
<td>Modern Models</td>
</tr>
</tbody>
</table>

These slides available at: [https://user.ceng.metu.edu.tr/~skalkan/DL/week_10.pdf](https://user.ceng.metu.edu.tr/~skalkan/DL/week_10.pdf)
Administrative Issues

• Programming assignment 1
• Take-Home Exam 1

• Office Hour:
  – Every Tuesday, 21:00

• Project paper selection
  – https://docs.google.com/spreadsheets/d/1tzPHq_Vgu6gCwNyXJHGvqeA6p
gU67H0nKYjfqkisWfKc/edit?usp=sharing
  – Deadline: 19th of April
Types of Convolution: Unshared convolution

• In some cases, sharing the weights do not make sense
  – When?

• Different parts of the input might require different types of features

• In such a case, we just have a network with local connectivity

• E.g., a face.
  – Features are not repeated across the space.
Types of Convolution:
Dilated (Atrous) Convolution

Purpose: Increase effective receptive field size without increasing parameters.

Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) $F_1$ is produced from $F_0$ by a 1-dilated convolution; each element in $F_1$ has a receptive field of $3 \times 3$. (b) $F_2$ is produced from $F_1$ by a 2-dilated convolution; each element in $F_2$ has a receptive field of $7 \times 7$. (c) $F_3$ is produced from $F_2$ by a 4-dilated convolution; each element in $F_3$ has a receptive field of $15 \times 15$. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.
Types of Convolution: **Transposed Convolution**

**Purpose:** Increasing layer width+height (upsampling).

---

**The size of the output:**

- **Regular convolution:** \( O = \left( \frac{W - F + 2 \times P}{S} \right) + 1 \)
- **Transpose convolution:** \( W = (O - 1) \times S + F - 2 \times P \)

---

Fig. 13.10.1 Transposed convolution layer with a 2 × 2 kernel.

Figure: [https://d2l.ai/chapter_computer-vision/transposed-conv.html](https://d2l.ai/chapter_computer-vision/transposed-conv.html)
Types of Convolution: Transposed Convolution

https://github.com/vdumoulin/conv_arithmetic
Purpose: Work with 3D data, e.g. learn spatial + temporal representations for videos.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: 1x1 Convolution

Purpose: Reduce number of channels.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: Separable Convolution

Purpose: Reduce number of parameters and multiplications.

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 \\
\end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}
\]

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution:
Depth-wise Separable Convolution

Purpose: Reduce number of parameters and multiplications.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: Group Convolution

AlexNet
Types of Convolution:

Group Convolution

**Purpose:** Reduce number of parameters and multiplications.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution:

Group Convolution

• Benefits:
  – Efficiency in training (distribute groups to different GPUs)
  – Decrease in # of parameters as the # of groups increases
  – Better performance?

Figure: https://blog.yani.ai/filter-group-tutorial/

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution:

Deformable Convolution


**Purpose:** Flexible receptive field.

Figure 1: Illustration of the sampling locations in $3 \times 3$ standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.

Figure 2: Illustration of $3 \times 3$ deformable convolution.
Convolution demos & tutorials

- https://github.com/vdumoulin/conv_arithmetic
- https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
OPERATIONS IN A CNN:
POOLING
Pooling

- Apply an operation on the “detector” results to combine or to summarize the answers of a set of units.
  - Applied to each channel (depth slice) independently
  - The operation has to be differentiable of course.
- Alternatives:
  - Maximum
  - Sum
  - Average
  - Weighted average with distance from the value of the center pixel
  - L2 norm
  - Second-order statistics?
  - …
- Different problems may perform better with different pooling methods
- Pooling can be overlapping or non-overlapping

Remember the motivation for CNNs:
S (simple) cells: local feature extraction.
C (complex) cells: provide tolerance to deformation, e.g. shift.
Pooling

• Example
  – Pooling layer with filters of size 2x2
  – With stride = 2
  – Discards 75% of the activations
  – Depth dimension remains unchanged

• Max pooling with F=3, S=2 or F=2, S=2 are quite common.
  – Pooling with bigger receptive field sizes can be destructive

• Avg pooling is an obsolete choice. Max pooling is shown to work better in practice.


http://cs231n.github.io/convolutional-networks/
Pooling

• Pooling provides invariance to small translation.

• If you pool over different convolution operators, you can gain invariance to different transformations.

Pooling can downsample

- Especially needed when to produce an output with fixed-length on varying length input.

- If you want to use the network on images of varying size, you can arrange this with pooling (with the help of convolutional layers)

CNNs without pooling

• “Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.”

http://cs231n.github.io/convolutional-networks/

Summary: Convolution & pooling

• Provide strong bias on the model and the solution
• They directly affect the overall performance of the system
OPERATIONS IN A CNN:
NONLINEARITY
Non-linearity

• Sigmoid
• Tanh
• ReLU and its variants
  – The common choice
  – Faster
  – Easier (in backpropagation etc.)
  – Avoids saturation issues
• ...

Spring 2021
Sinan Kalkan
OPERATIONS IN A CNN:
NORMALIZATION
• From Krizhevsky et al. (2012):

generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel $i$ at position $(x,y)$ and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

$$b^i_{x,y} = a^i_{x,y} / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

where the sum runs over $n$ “adjacent” kernel maps at the same spatial position, and $N$ is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants $k$, $n$, $\alpha$, and $\beta$ are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$. We
Normalization

https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7
OPERATIONS IN A CNN:
FULLY CONNECTED LAYER
Fully-connected layer

- At the top of the network for mapping the feature responses to output labels
- Full connectivity
- Can be many layers
- Various activation functions can be used
CNN ARCHITECTURES
A Blueprint for CNNs

INPUT $\rightarrow$ $[[\text{CONV} \rightarrow \text{RELU}] \ast N \rightarrow \text{POOL}?] \ast M \rightarrow [\text{FC} \rightarrow \text{RELU}] \ast K \rightarrow \text{FC}$

where the $\ast$ indicates repetition, and the $\text{POOL}$? indicates an optional pooling layer. Moreover, $N \geq 0$ (and usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (and usually $K < 3$). For example, here are some common ConvNet architectures you may see that follow this pattern:

- **INPUT $\rightarrow$ FC**, implements a linear classifier. Here $N = M = K = 0$.
- **INPUT $\rightarrow$ CONV $\rightarrow$ RELU $\rightarrow$ FC**
- **INPUT $\rightarrow$ $[\text{CONV} \rightarrow \text{RELU} \rightarrow \text{POOL}] \ast 2 \rightarrow \text{FC} \rightarrow \text{RELU} \rightarrow \text{FC}**. Here we see that there is a single CONV layer between every POOL layer.
- **INPUT $\rightarrow$ $[\text{CONV} \rightarrow \text{RELU} \rightarrow \text{CONV} \rightarrow \text{RELU} \rightarrow \text{POOL}] \ast 3 \rightarrow [\text{FC} \rightarrow \text{RELU}] \ast 2 \rightarrow \text{FC}**. Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

http://cs231n.github.io/convolutional-networks/
Fully Convolutional Networks (FCNs)

- Fully-connected layers limit the input size
- Use convolution, especially 1x1 convolution to reduce channels and layer size

Figure 2. Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end dense learning.


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.
Demo

http://scs.ryerson.ca/~aharley/vis/conv/
General rules of thumb:

The input layer

• The size of the input layer should be divisible by 2 many times
  – Hopefully a power of 2
• E.g.,
  – 32 (e.g. CIFAR-10),
  – 64,
  – 96 (e.g. STL-10), or
  – 224 (e.g. common ImageNet ConvNets),
  – 384, and 512 etc.
General rules of thumb:  
The conv layer  

• Small filters with stride 1  
• Usually zero-padding applied to keep the input size unchanged  
• In general, for a certain $F$, if you choose  
  \[ P = \frac{(F - 1)}{2}, \]
  the input size is preserved (for $S=1$):  
  \[ \frac{W - F + 2P}{S} + 1 \]

• Number of filters:  
  – A convolution channel is more expensive compared to fully-connected layer.  
  – We should keep this as small as possible.
General rules of thumb:
The pooling layer

• Commonly,
  – F=2 with S=2
  – Or: F=3 with S=2

• Bigger F is very destructive
Taking care of downsampling

• At some point(s) in the network, we need to reduce the size
• If conv layers do not downsize, then only pooling layers take care of downsampling
• If conv layers also downsize, you need to be careful about strides etc. so that
  (i) the dimension requirements of all layers are satisfied and
  (ii) all layers tile up properly.
• S=1 seems to work well in practice
• However, for bigger input volumes, you may try bigger strides
TRAINING A CNN

Fig: http://www.robots.ox.ac.uk/~vgg/practicals/cnn/
Feed-forward through convolution

\[
a_i^l = \sigma(\text{net}_i^l)
\]

\[
\text{net}_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}
\]

For example:

\[
\text{net}_1^1 = w_1 a_1^{l-1} + w_2 a_2^{l-1} + w_3 a_3^{l-1}
\]
Backpropagation through convolution

Feedforward:
\[ a_i^l = \sigma (\text{net}_i^l) \]
\[ \text{net}_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j}^{l-1} \]

Gradient wrt. weights:
\[
\frac{\partial L}{\partial w_k} = \sum_i \frac{\partial L}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_k} + \sum_i \frac{\partial L}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_k} \frac{\partial a_i^l}{\partial \text{net}_i^l} \frac{\partial \text{net}_i^l}{\partial w_k}
\]
Backpropagation through convolution

Feedforward:
\[ a_i^l = \sigma(\text{net}_i^l) \]
\[ \text{net}_i^l = \sum_{j=1}^F w_j \cdot a_{i+j}^{l-1} \]

Gradient wrt. input layer:
\[ \frac{\partial L}{\partial a_{3}^{l-1}} =? \]
\[ = \frac{\partial L}{\partial a_1^l} \frac{\partial a_1^l}{\partial \text{net}_1^l} + \frac{\partial L}{\partial a_2^l} \frac{\partial a_2^l}{\partial \text{net}_2^l} \frac{\partial \text{net}_2^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_2^l} \frac{\partial a_2^l}{\partial \text{net}_1^l} \frac{\partial \text{net}_1^l}{\partial a_3^{l-1}} \]
\[ = \frac{\partial L}{\partial \text{net}_1^l} w_3 + \frac{\partial L}{\partial \text{net}_2^l} w_2 + \frac{\partial L}{\partial \text{net}_3^l} w_1 \]

In general:
\[ \frac{\partial L}{\partial a_i^{l-1}} = \sum_{j=1}^F \frac{\partial L}{\partial \text{net}_{i-j+1}^l} w_j \]

This is also convolution!
Feed-forward through pooling

\[ a_i^l = \max\{a_{i+j-1}^{l-1}\}_{j=1}^F \]

For example:
\[ \text{net}_1^l = \max\{a_1^{l-1}, a_2^{l-1}, a_3^{l-1}\} \]
Backpropagation through pooling

Feedforward:
\[ a_i^l = \max\{a_{i+j-1}^{l-1}\}_{j=1}^F \]

Using derivative of max:
\[
\frac{\partial L}{\partial a_i^{l-1}} = \frac{\partial L}{\partial \text{net}_k^l} \frac{\partial \text{net}_k^l}{\partial a_i^{l-1}}
\]
\[
= \begin{cases} 
\frac{\partial L}{\partial \text{net}_k^l}, & a_i^{l-1} \text{ is max} \\
0, & \text{otherwise}
\end{cases}
\]

This requires that we save the index of the max activation (sometimes also called the switches) so that gradient “routing” is handled efficiently during backpropagation.
Backpropagation

- Backpropagation through non-linearity and fully-connected layers are straight-forward
How to initialize the weights?

• Option 1: randomly
  – E.g. using He initialization (check Week 7 slides)
  – This has been shown to work nicely in the literature

• Option 2:
  – Train/obtain the “filters” elsewhere and use them as the weights
  – Unsupervised pre-training using image patches (windows)
  – Avoids full feedforward and backward pass, allows the search to start from a better position
  – You may even skip training the convolutional layers

CONVOLUTIONAL CLUSTERING FOR UNSUPERVISED LEARNING

Ayseulg Dundar, Jonghoon Jin, and Eugenio Culurciello
Purdue University, West Lafayette, IN 47907, USA
{adundar,jhjin,euge}@purdue.edu

3.1 LEARNING FILTERS WITH K-MEANS

Our method for learning filters is based on the k-means algorithm. The classic k-means algorithm finds cluster centroids that minimize the distance between points in the Euclidean space. In this context, the points are randomly extracted image patches and the centroids are the filters that will be used to encode images. From this perspective, k-means algorithm learns a dictionary $D \in \mathbb{R}^{n \times k}$ from the data vector $w^{(i)} \in \mathbb{R}^n$ for $i = 1, 2, ..., m$. The algorithm finds the dictionary as follows:

$$s_j^{(i)} := \begin{cases} D^{(j)^T} w^{(i)} & \text{if } j = \text{argmax}_l |D^{(l)^T} w^{(i)}|, \\ 0 & \text{otherwise}, \end{cases}$$

$$D := W S^T + D,$$

$$D^{(j)} := \frac{D^{(j)}}{||D^{(j)}||_2},$$

where $s^{(i)} \in \mathbb{R}^k$ is the code vector associated with the input $w^{(i)}$, and $D^{(j)}$ is the $j$'th column of the dictionary $D$. The matrices $W \in \mathbb{R}^{n \times m}$ and $S \in \mathbb{R}^{k \times m}$ have the columns $w^{(i)}$ and $s^{(i)}$, respectively. $w^{(i)}$'s are randomly extracted patches from input images that have the same dimension as the dictionary vectors, $D^{(j)}$. 

---

Table 3: Classification error on MNIST.

(a) Algorithms that learn the filters unsupervised.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>600</th>
<th>1000</th>
<th>3000</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhao et al. (2015) (auto-encoder)</td>
<td>8.8%</td>
<td>8.2%</td>
<td>1.78%</td>
<td>1.43%</td>
</tr>
<tr>
<td>Rifai et al. (2011) (contractive auto-encoder)</td>
<td>6.3%</td>
<td>4.77%</td>
<td>3.22%</td>
<td>1.14%</td>
</tr>
<tr>
<td>This work (2 layers + multi class)</td>
<td>2.8%</td>
<td>2.5%</td>
<td>1.4%</td>
<td>0.5%</td>
</tr>
</tbody>
</table>

(b) Supervised and semi-supervised algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>600</th>
<th>1000</th>
<th>3000</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeCun et al. (1998) (convnet)</td>
<td>7.68%</td>
<td>6.43%</td>
<td>3.85%</td>
<td></td>
</tr>
<tr>
<td>Lan (2013) (pseudo label)</td>
<td>5.63%</td>
<td>3.46%</td>
<td>2.69%</td>
<td></td>
</tr>
<tr>
<td>Zhao et al. (2015) (semi-supervised auto-encoder)</td>
<td>3.91%</td>
<td>2.83%</td>
<td>2.15%</td>
<td>0.71%</td>
</tr>
<tr>
<td>Kingma et al. (2014) (generative model)</td>
<td>2.59%</td>
<td>2.40%</td>
<td>2.18%</td>
<td>0.96%</td>
</tr>
<tr>
<td>Rasmus et al. (2015) (semi-supervised ladder)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.16%</td>
</tr>
</tbody>
</table>