# Homework 2: Convolutional Neural Networks (CNNs)

Daryoush Safe

October 2025

## 1 Theoretical Problems

### Problem 1

(a) **Parameter Sharing:** A single **filter (or kernel)** is applied across various regions of the input, significantly decreasing the parameter count compared to fully connected (dense) networks.
The filter slides (strides) over the input, computing the dot product at each position to produce a **feature map**. That means the **same set of weights** is reused (shared) for all positions

(b)
- (i) CNNs are very suitable. Because the wolf may appear at any position, orientation, or scale in the image (depending on the drone's altitude and camera angle), CNNs can effectively detect it due to translation invariance and local receptive fields. Moreover, using a Fully Connected Network (FCN) would require an enormous number of parameters and would fail to generalize well to spatial transformations.
- (ii) CNNs are suitable. Here, the task is detect a specific text (e.g., "Welcome") anywhere in the audio stream. Therefore it is required to be translation invariance. Using an FCN on raw waveform or fixed windows would require huge models and massive negative sampling across all possible positions — impractical.
- (iii) Partially suitable. A 2D CNN captures spatial patterns per frame, but recognizing an action also requires understanding temporal changes across frames. Hence, CNNs are usually combined with temporal models such as 3D CNNs, RNNs, or Transformers to capture motion dynamics.

(c) In tasks involving downsampling, we learn to detect increasingly important features in each layer while becoming invariant to shifts and other transformations. However, to reconstruct an image from downsampled features, we must return to the original spatial dimensions—this requires upsampling. Upsampling is achieved in two main ways:

- Non-learnable methods (e.g., unpooling variants like max unpooling, nearest-neighbor, or bilinear interpolation).
- Learnable methods, primarily transpose convolution.

In transpose convolution, a single value in the low-resolution feature map is mapped to a patch in the higher-resolution output using learnable weights. This makes the upsampling process trainable—the model learns how to upsample effectively.

## Problem 2

(a)
- **Invariance:** The output remains the same when the input undergoes a transformation.
  Mathematically:
  $$f(T(x)) = f(x)$$
- **Equivariance:** The output varies predictably when the input undergoes a transformation.
  Mathematically:
  $$f(T(x)) = T(f(x))$$

(b) The layer most responsible for creating local translation invariance in a CNN is the pooling layer (especially max pooling). Pooling (or using a convolution with stride $> 1$) reduces spatial resolution while preserving the presence of features rather than their exact positions. This means small local shifts in the input image will not significantly affect the output.
However, the model becomes only locally translation invariant, not fully invariant. Deeper layers, downsampling, and finally global average pooling or fully connected layers (caused flatten feature-map) help achieve a full translation invariance at the final stage.

(c) Standard CNNs are not naturally rotation-equivariant or rotation-invariant. Because convolution filters slide in fixed directions (horizontal/vertical), not rotated ones. If an object in the image rotates, the spatial pattern that the filter detects no longer matches as before.
E.g. Suppose that a CNN has a filter that detects a horizontal edge. If you rotate the image 90°, that edge becomes vertical. The horizontal-edge filter will now output nothing and the vertical-edge filter (if it exists) might respond instead.
Data augmentation with rotated images can bring rotation invariance.

(d) In many problems the location of a pattern is not important. Only the presence of the pattern matters. Need a simple network that will fire regardless of the location of the pattern.

(e)
- **Pose Estimation:** This involves determining the exact position and orientation of objects or body parts.
- **Facial landmark detection:** Translation invariance would make it impossible to distinguish left eye from right eye or align features anatomically.
- **Medical image analysis for anatomical localization**

## Problem 3

(a) **Traditional Methods and Their Costs**

(i) In a standard, sequential CNN such as VGG, there are two main strategies to increase the receptive field:

- **Stacking more convolutional layers:** As the network becomes deeper, each neuron in higher layers receives input from a larger region of the original image. This happens because each convolution builds upon the receptive field of the previous layer.
- **Using spatial downsampling (Pooling or Strided Convolutions):** Operations such as max pooling or using stride $> 1$ reduce the spatial dimensions of feature maps while maintaining their depth. This process effectively increases the receptive field of subsequent neurons, allowing them to capture more global features of the input.

(ii) Using large filters such as $15 \times 15$ is inefficient because the number of parameters and computations grows quadratically with filter size. For a layer with $C$ input and output channels:

$$\text{Parameters} = (15 \times 15 \times C) \times C = 225C^2$$

This large number of parameters increases both memory consumption and computational cost, leading to slower training and higher overfitting risk.

In contrast, using smaller filters across multiple layers can achieve a similar receptive field of while significantly reducing the number of parameters and introducing more nonlinear transformations between layers:

- one $9 \times 9$ filter: $(9 \times 9 \times C) \times C = 81C^2$
- 3-stacked $3 \times 3$ filters: $(3 \times 3 \times C) \times 3 \times C = 27C^2$

(iii) Although stacking many small filters is parameter-efficient, excessively deep networks face several optimization challenges. Beyond a certain depth, simply adding more layers may reduce training and test accuracy, as the network becomes harder to optimize effectively. As gradients are backpropagated through many layers, they can diminish, causing the model to stop learning.

(b) **Clever Solutions in Modern Architectures**

(i) **The Inception module** is designed to capture visual features at multiple scales simultaneously. It does this by processing the same input through parallel convolutional paths that use filters of different sizes (e.g., 1×1, 3×3, and 5×5). Each filter size corresponds to a different receptive field, allowing the network to detect both local fine details (small filters) and more global structures (large filters).

To make sure all feature maps can be concatenated along the channel dimension, padding is applied so that all outputs have the same spatial dimensions (height and width). The outputs from each parallel branch are then concatenated depth-wise, forming a rich, multi-scale feature representation.

However, using multiple filters directly would make the module computationally expensive because of the high number of parameters. To solve this, the Inception module introduces 1×1 convolutions before applying larger filters. These 1×1 convolutions act as bottlenecks that reduce the number of input channels (i.e., perform dimensionality reduction), thus lowering both memory usage and computational cost, while also adding extra nonlinearity to improve feature learning.

(ii) **A dilated (or atrous) convolution** is a variant of the standard convolution that introduces gaps (holes) between filter elements, controlled by a parameter called the dilation rate (r).

- When $r = 1$, the convolution is standard (no gaps).
- When $r > 1$, the filter elements are spaced apart, effectively expanding the receptive field without adding extra parameters or increasing computation.

Consider a 3×3 filter:

- When $r = 1$, the filter covers a 3×3 patch of input pixels.
- When $r = 2$, each filter element skips 1 pixel in both directions — meaning the effective receptive field becomes:

$$\text{Effective filter size} = 3 + (3-1) \times (r-1) = 3 + 2 \times 1 = 5$$

$$\begin{array}{ccccc} \# & \cdot & \# & \cdot & \# \\ . & . & . & . & . \\ \# & \cdot & \# & \cdot & \# \\ . & . & . & . & . \\ \# & \cdot & \# & \cdot & \# \end{array} \quad \text{where } \cdot = \text{skipped pixel}$$

The receptive field increases (acts like a $5 \times 5$ filter).

## Problem 4

(a) Notation:

- $C_{in}$, $C_{out}$: input / output channels
- $k_h$, $k_w$: kernel size
- $s_h$, $s_w$: stride size
- $p$: padding (*same*: output size = input size; *valid*: 0 padding)
- $d$: dilation ratio
- $H_{in}$, $W_{in}$: input feature-map size for the layer
- $H_{out}$, $W_{out}$: output feature-map size for the layer

Input dimension after pooling layer:

$$H_{out} = \lfloor \frac{H_{in} - p}{s} \rfloor + 1$$
$$W_{out} = \lfloor \frac{W_{in} - p}{s} \rfloor + 1$$
$$C_{in} = C_{out}$$

Input dimension after convolution layer:

$$H_{out} = \lfloor \frac{H_{in} + 2p - \text{Effective}(k_h)}{s} \rfloor + 1$$
$$W_{out} = \lfloor \frac{W_{in} + 2p - \text{Effective}(k_w)}{s} \rfloor + 1$$
$$C_{in} = C_{out}$$

Effective receptive field after dilation:

$$\text{Effective Receptive Field w.r.t previous layer} = k + (k-1) \cdot (d-1)$$

Closed-Form Expression (for $N$ layers):

$$R_N = 1 + \sum_{i=1}^{N} (k_i - 1) \cdot d_i \cdot \prod_{j=1}^{i-1} s_j$$
$$R_i = R_{i-1} + (k_i - 1) \times d_i \times j_{t-1}$$

Where:

$$j_t = j_{t-1} \times s_i$$

Parameters count in convolution layers:

$$(k_h \cdot k_w \cdot C_{in} + 1) \cdot C_{out}$$

Parameters count in batch normalization layers *(running mean and running variance for each output channel)*:

$$2 \times C_{out}$$

Multiply-Accumulate operations (MACs):

$$H_{out} \cdot W_{out} \cdot C_{out} \cdot (k_h \cdot k_w \cdot C_{in})$$

**Solution for Network Architecture**

- Layer1
  (i) Output size: $256 \times 256 \times 32$
  (ii) Parameters: $(7 \times 7 \times 3 + 1) \times 32$
  (iii) MACs: $(256 \times 256 \times 32) \times (7 \times 7 \times 3)$
  (iv) ERF: $1 + (7 - 1) \times 1 \times 1 = 7$
- BN1
  (i) Output size: $256 \times 256 \times 32$
  (ii) Parameters: $2 \times 32$
  (iii) MACs: 0
- Layer2
  (i) $H_{out} = \lfloor \frac{256-5}{2} \rfloor + 1 = 126$
  (ii) Output size: $126 \times 126 \times 64$
  (iii) Parameters: $(5 \times 5 \times 32 + 1) \times 64$
  (iv) MACs: $(126 \times 126 \times 64) \times (5 \times 5 \times 32)$
  (v) ERF: $7 + (5 - 1) \times 1 \times 1 = 11$
- BN2
  (i) Output size: $126 \times 126 \times 64$
  (ii) Parameters: $2 \times 64$
  (iii) MACs: 0
- Layer3
  (i) $H_{out} = \lfloor \frac{126-2}{2} \rfloor + 1 = 63$
  (ii) Output size: $63 \times 63 \times 64$
  (iii) Parameters: 0
  (iv) MACs: 0
  (v) ERF: $11 + (2 - 1) \times 1 \times 2 = 13$
- Layer4
  (i) $H_{out} = \lfloor \frac{63-5}{1} \rfloor + 1 = 59$
  (ii) Output size: $59 \times 59 \times 128$
  (iii) Parameters: $(3 \times 3 \times 64 + 1) \times 128$
  (iv) MACs: $(59 \times 59 \times 128) \times (3 \times 3 \times 64)$
  (v) ERF: $13 + (3 - 1) \times 2 \times 4 = 29$
- BN4
  (i) Output size: $59 \times 59 \times 128$
  (ii) Parameters: $2 \times 128$
  (iii) MACs: 0
- Layer5
  (i) $H_{out} = \lfloor \frac{59-3}{1} \rfloor + 1 = 57$
  (ii) Output size: $57 \times 57 \times 128$

(iii) Parameters: $(3 \times 3 \times 128 + 1) \times 128$
(iv) MACs: $(57 \times 57 \times 128) \times (3 \times 3 \times 128)$
(v) ERF: $29 + (3-1) \times 1 \times 4 = 37$

- BN5
  (i) Output size: $57 \times 57 \times 128$
  (ii) Parameters: $2 \times 128$
  (iii) MACs: 0
- Layer6
  (i) $H_{out} = \lfloor \frac{57-2}{2} \rfloor + 1 = 28$
  (ii) Output size: $28 \times 28 \times 128$
  (iii) Parameters: 0
  (iv) MACs: 0
  (v) ERF: $37 + (2-1) \times 1 \times 4 = 41$
- Layer7
  (i) $H_{out} = \lfloor \frac{28-3}{1} \rfloor + 1 = 26$
  (ii) Output size: $26 \times 26 \times 256$
  (iii) Parameters: $(3 \times 3 \times 128 + 1) \times 256$
  (iv) MACs: $(26 \times 26 \times 256) \times (3 \times 3 \times 128)$
  (v) ERF: $41 + (3-1) \times 1 \times 8 = 57$
- BN5
  (i) Output size: $26 \times 26 \times 256$
  (ii) Parameters: $2 \times 256$
  (iii) MACs: 0
- Layer8
  (i) $H_{out} = \lfloor \frac{26-2}{2} \rfloor + 1 = 13$
  (ii) Output size: $13 \times 13 \times 256$
  (iii) Parameters: 0
  (iv) MACs: 0
  (v) ERF: $57 + (2-1) \times 1 \times 8 = 65$
- FC1
  (i) Parameters: $43264 \times 1024 + 1024$
  (ii) MACs: $43264 \times 1024$
- FC2
  (i) Parameters: $1024 \times 1024 + 1024$
  (ii) MACs: $1024 \times 1024$
- FC3
  (i) Parameters: $1024 \times 10 + 10$
  (ii) MACs: $1024 \times 10$