HW2

Feras Dwere 214225021 Mahmoud Abade 206773756

December 2025

1 Question 3: Translational Properties of CNNs (18 Points)

Question: Convolutional neural networks are often motivated by geometric properties of images, in particular the fact that semantic content typically does not depend on exact pixel alignment. Throughout this question, refer to Figure 1, which displays three spatially shifted versions of the same digit image.

(a) Provide concise and formal definitions of translational equivariance and translational invariance, expressed using short function notation. Your answer should clearly distinguish between these two properties.

(b) Identify which components of a convolutional neural network equipped with max-pooling exhibit equivariance, and which components introduce invariance. Support your answer by referring to the computational roles of convolution and pooling.

(c) Consider the three shifted images in Figure 1. Discuss whether a CNN with standard max-pooling (e.g., 2×2 or 3×3 pooling with stride ¿ 1) will necessarily produce identical internal activations or class scores for all three versions. Relate your explanation to receptive field growth, discretization effects introduced by pooling stride, and the concepts defined in part (a).

Answer:

Convolutional Neural Networks (CNNs) are fundamentally motivated by the geometric structure of image data. In many visual recognition tasks, the semantic meaning of an object is independent of its exact spatial location within the image. As a result, CNNs are designed to exploit translation-related symmetries in order to efficiently learn meaningful representations. Two central concepts in this context are *translational equivariance* and *translational invariance*. Although closely related, these properties describe fundamentally different behaviors and play distinct roles within CNN architectures.

1.1 Part (a): Definitions of Translational Equivariance and Translational Invariance

1.1.1 Translation Operator

Let $x : \mathbb{Z}^2 \to \mathbb{R}$ denote a discrete image defined on a two-dimensional grid. We define the translation operator T_Δ, parameterized by a displacement vector $\Delta = (\Delta_x, \Delta_y) \in \mathbb{Z}^2$, as

$$(T_\Delta x)(u) = x(u - \Delta),$$

where $u \in \mathbb{Z}^2$ indexes pixel locations. This operator shifts the image spatially without altering its internal structure.

1.1.2 Translational Equivariance

A function f, representing a neural network layer or a composition of layers, is said to be *translationally equivariant* if translating the input results in an equivalent translation of the output:

$$f(T_\Delta x) = T_\Delta f(x) \quad \forall x, \Delta.$$

This property implies that the network preserves spatial relationships: when a pattern in the input is shifted, the corresponding activation in the output feature map is shifted by the same amount. Importantly, equivariance does not remove spatial information; rather, it ensures a consistent and predictable response to translations. Standard convolution operations inherently satisfy translational equivariance due to weight sharing and local receptive fields.

1.1.3 Translational Invariance

In contrast, a function f is said to be *translationally invariant* if the output remains unchanged under translations of the input:

$$f(T_\Delta x) = f(x) \quad \forall x, \Delta.$$

Translational invariance implies that the network output depends solely on the presence of a feature, not on its spatial location. This property is particularly desirable in classification tasks, where the goal is to assign the same label to an object regardless of where it appears in the image. In CNNs, invariance is typically achieved through spatial aggregation mechanisms such as pooling or global averaging, which deliberately discard precise positional information.

1.1.4 Conceptual Distinction

The distinction between equivariance and invariance is crucial:

- **Translational equivariance** preserves spatial structure and is essential for hierarchical feature extraction.

- **Translational invariance** removes spatial dependence and supports robust decision-making at the output level.

Modern CNN architectures leverage equivariance in early layers to build rich spatial representations, while progressively introducing invariance in deeper layers to achieve stable and semantically meaningful predictions.

1.2 Part (b): Equivariance and Invariance in CNN Components

A Convolutional Neural Network is composed of multiple computational components, each contributing differently to the translational behavior of the network. In particular, convolutional layers and pooling layers play distinct and complementary roles with respect to translational equivariance and translational invariance.

1.2.1 Convolutional Layers and Translational Equivariance

Convolutional layers are inherently *translationally equivariant.* This property follows directly from the definition of discrete convolution and the principle of weight sharing.

Let x be an input image and k a convolution kernel. The convolution operation is defined as

$$(f(x))(u) = (x * k)(u) = \sum_v x(v)\, k(u - v).$$

Applying a translation T_Δ to the input yields

$$f(T_\Delta x)(u) = \sum_v x(v - \Delta)\, k(u - v).$$

By a change of variables, this expression simplifies to

$$f(T_\Delta x)(u) = (T_\Delta f(x))(u),$$

which confirms that convolution satisfies

$$f(T_\Delta x) = T_\Delta f(x).$$

Intuitively, this means that a convolutional filter detects the same pattern regardless of its spatial location, and the resulting feature activation appears at the corresponding shifted position in the output feature map. Importantly, the spatial arrangement of features is preserved, enabling CNNs to build hierarchical representations across layers.

1.2.2 Pooling Layers and Translational Invariance

Pooling layers, particularly max-pooling, are introduced to reduce spatial resolution and increase robustness to small input translations. Unlike convolution, pooling operations tend to introduce *translational invariance.*

A pooling operation aggregates information over a local neighborhood. For example, max-pooling computes

$$y(u) = \max_{v \in \mathcal{N}(u)} x(v),$$

where $\mathcal{N}(u)$ denotes a local spatial region. As a result, small translations of the input that do not move features outside the pooling window often lead to identical pooled outputs.

However, it is important to note that standard max-pooling with stride greater than one does not provide perfect invariance. Instead, it introduces *approximate invariance* by discarding precise spatial information within pooling regions. Larger translations or shifts across pooling boundaries may still change the output.

1.2.3 Combined Effect in CNN Architectures

In practice, CNNs exploit a deliberate interplay between equivariance and invariance:

- **Convolutional layers** preserve translational equivariance, maintaining spatial structure across feature maps.
- **Pooling layers** progressively reduce spatial sensitivity, introducing robustness and partial translational invariance.

This design allows CNNs to first capture detailed spatial patterns and then gradually abstract away exact location information, enabling effective recognition and classification across varying object positions.

1.3 Part (c): Effect of Spatial Shifts Under Max-Pooling

We now consider the three spatially shifted instances of the same digit shown in Figure 1 and analyze whether a CNN equipped with standard max-pooling necessarily produces identical internal activations or final class scores for all three inputs.

1.3.1 Non-Ideal Translational Invariance in Practice

Although convolutional layers are translationally equivariant, a CNN with standard max-pooling does *not* guarantee identical internal activations or outputs for translated versions of the same image. This is due to discretization effects introduced by pooling operations with stride greater than one.

Max-pooling aggregates local activations within fixed, non-overlapping windows. When an input image is translated by an amount that is not an integer multiple of the pooling stride, the alignment between feature activations and pooling windows changes. Consequently, different activations may be selected as maxima, leading to distinct pooled outputs.

1.3.2 Role of Pooling Stride and Discretization

Let the pooling window have size $k \times k$ and stride $s > 1$. Translations smaller than s pixels may cause a feature to move across pooling boundaries, resulting in different pooled representations even though the underlying semantic content remains unchanged.

This discretization breaks exact translational equivariance:

$$f(T_\Delta x) \neq T_\Delta f(x),$$

and also prevents strict translational invariance:

$$f(T_\Delta x) \neq f(x),$$

for general translations Δ.

Thus, max-pooling introduces only *approximate* translational invariance, limited to shifts that remain within pooling regions.

1.3.3 Receptive Field Growth and Deep Architectures

As the network depth increases, receptive fields grow, allowing neurons in deeper layers to integrate information over larger spatial extents. This increases robustness to translations at the semantic level. However, the loss of precise spatial alignment caused by pooling remains irreversible, and early misalignments may propagate through the network.

As a result, while deeper layers may produce similar class scores for moderately shifted inputs, there is no theoretical guarantee that internal feature maps or final outputs will be identical across all spatial shifts.

1.3.4 Connection to Equivariance and Invariance

The observed behavior directly reflects the distinction introduced in Section 3.1:

- Convolution preserves *translational equivariance.*
- Max-pooling trades equivariance for *partial translational invariance.*

Therefore, a CNN with standard max-pooling is neither fully translationally equivariant nor strictly translationally invariant. Instead, it achieves a practical compromise that improves robustness while sacrificing exact spatial consistency.

1.3.5 Conclusion

In summary, the three shifted images in Figure 1 will not necessarily produce identical internal activations or class scores in a CNN with standard max-pooling. Differences arise from pooling stride discretization and receptive field alignment, despite the semantic equivalence of the inputs.

2 Question 4: Representation Efficiency and Inductive Bias (12 Points)

Question: Modern CNN architectures leverage locality and parameter sharing, whereas fully connected networks treat each input dimension independently.

(a) Compare the number of learnable parameters required to process a 28×28×3 input using:

  (i) a convolutional layer employing three filters of size 2 × 2 × 3 with valid padding, and

  (ii) a fully connected layer that reads the same input (i.e., takes all 28 × 28 × 3 pixel values as its input units and maps them to a single hidden layer of your choice).

  Explain why one of these approaches is more parameter-efficient, and briefly comment on how this efficiency relates to overfitting.

(b) Again, in reference to Figure 1, explain why weight sharing enables convolutional networks to generalize more robustly to spatial shifts than fully connected networks. Your answer should articulate the inductive bias embedded in convolutional design (for example, the assumption that similar patterns can appear at multiple spatial locations).

Answer:

2.1 Part (a): Parameter comparison: convolution vs. fully connected

We compare the number of learnable parameters needed to process an input of size 28 × 28 × 3.

2.1.1 (i) Convolutional layer (3 filters of size $2\times2\times3$, valid padding).

Each convolutional filter spans the full input depth (3 channels). Therefore, the number of weights per filter is

$$2 \cdot 2 \cdot 3 = 12.$$

If a bias term is included (the common convention), each filter has $12 + 1 = 13$ learnable parameters. With three filters:

$$\#\text{params}_{\text{conv}} = 3 \cdot (12 + 1) = 39.$$

(If biases are omitted, then $\#\text{params}_{\text{conv}} = 3 \cdot 12 = 36$.)

2.1.2 (ii) Fully connected layer (reads all pixels into one hidden layer).

A fully connected (dense) layer treats the input as a vector. The input dimensionality is

$$28 \cdot 28 \cdot 3 = 2352.$$

Let the hidden layer have H units (chosen by the designer). Then the weight matrix has size $2352 \times H$, and there is typically one bias per hidden unit. Hence:

$$\#\text{params}_{\text{FC}} = 2352H + H = H(2352 + 1) = 2353H.$$

For example, if $H = 100$, then $\#\text{params}_{\text{FC}} = 2353 \cdot 100 = 235{,}300$.

2.1.3 Why convolution is more parameter-efficient and the relation to overfitting.

The convolutional layer is far more parameter-efficient because it uses two key structural assumptions:

- **Locality:** each output depends only on a small spatial neighborhood (here 2×2) rather than all 28×28 locations.
- **Weight sharing:** the same filter parameters are reused at every spatial location, so the parameter count does *not* grow with the image width/height.

In contrast, a fully connected layer assigns independent weights to every input dimension, so its parameter count scales linearly with the input size and the chosen hidden width H.

Regarding overfitting: models with substantially more parameters can fit training data more easily (higher capacity), which increases the risk of overfitting when data is limited. Convolution reduces the number of free parameters and imposes an inductive bias consistent with images, often improving generalization compared to a dense layer with an equally expressive but much larger parameterization.

2.2 Part (b): Weight sharing, inductive bias, and robustness to spatial shifts

Referring to Figure 1, we observe multiple spatially shifted instances of the same digit. A convolutional neural network generalizes more robustly to such shifts than a fully connected network primarily due to *weight sharing*, which encodes a strong and appropriate inductive bias.

In a convolutional layer, a single filter is applied identically across all spatial locations. This means that the same set of weights is responsible for detecting a particular local pattern regardless of where it appears in the image. As a result, once a filter has learned to detect a feature (e.g., an edge or stroke), that knowledge automatically transfers to all spatial positions. In Figure 1, this implies that the same learned filters respond similarly to the digit regardless of its location.

This design reflects an inductive bias that is well matched to natural images: namely, the assumption that meaningful local patterns can occur at multiple spatial locations and should be treated equivalently. By enforcing this assumption through weight sharing, CNNs reduce the number of parameters while simultaneously improving generalization to unseen spatial configurations.

In contrast, a fully connected network assigns independent weights to each input dimension. Consequently, the same local pattern appearing at different spatial locations corresponds to different input units and different parameters. Without explicit exposure to all possible shifted versions during training, a fully connected network has no structural reason to treat these patterns as equivalent, leading to poorer generalization under spatial shifts.

Therefore, weight sharing not only improves parameter efficiency but also embeds a translation-aware inductive bias that enables convolutional networks to generalize more robustly to spatially shifted inputs, as illustrated by the examples in Figure 1.