Lecture 12: Feature Visualization and Network Inversion

Areej Alasiry

CIS 6217 – Computer Vision for Data Representation

College of Computer Science, King Khalid University

Network Inversion

- A deep learning interpretability technique that **reconstructs an input image** (or synthesizes one) from **intermediate layer activations**.

How it works:

- Forward Pass: Pass the image through a trained CNN to extract features at a chosen layer.
- Inverse Network: A separate, usually symmetrical upconvolution/deconvolution network (UpconvNet) takes these feature maps as input.
- Reconstruction: The inverse network generates an image by minimizing $MSE(x, \hat{x})$ between the reconstructed image x^x^ and the original input x.

Objective

- **It Reveals**
  - **Information Loss vs Abstraction:**
    - Early layer inversion → sharp details, edges, textures
    - Deeper layer inversion → semantic structure (object parts, category cues)
  - **Interpretability:**
    Shows what features (textures, shapes, semantics) the network encodes at each depth.
  - **Privacy Risks:**
    Demonstrates that **private or identifiable details** can be reconstructed from feature vectors.

Feature Visualization via Deconvolution / Network Inversion

- A **deconvolutional network** (also called “DeconvNet” / Up-ConvNet) is a reversed version of a convolutional network layer stack.
- It uses **unpooling (reverse pooling), rectified activation inversion, and deconvolution (transpose convolution)** to map feature-maps back into image space.
- By feeding a **feature map** from a hidden layer into the deconvnet, one obtains a **reconstructed (visualized) image pattern** — revealing what that layer “sees.”

It reveals:

- **Interpretability:** Shows what features (edges, textures, shapes, object-parts) activate neurons in each layer. Helps understand internal representations — not just outputs.
- **Hierarchy of abstraction:** Early layers reconstruct **fine details / textures**, deeper layers reconstruct **abstracted semantics / object structure** — revealing what is preserved vs. lost across layers. **Model diagnostics:** Helps detect dead filters or inactive neurons (no activation → blank reconstructions), indicating possible issues in training (bad weights, poor initialization).
- **Privacy concerns:** Since feature-maps can be inverted to approximate original input, this raises risk: feature embeddings may leak sensitive image content

Architecture

- **Forward pass:** Input image → CNN → obtain activation maps at chosen layer.
- **Prepare feature map:** Retain only the selected activations (e.g., one filter, or top N filters), zero-out others.
- **Inverse network (DeconvNet / UpconvNet):**
  - **Unpooling:** Using the saved "switches" from max-pooling to restore spatial layout.
  - **Inverse activation:** Reverse ReLU / nonlinearity handling.
  - **Transpose convolution:** Use flipped filters (transpose of original conv filters) to map features back to pixel space.
- **Reconstructed image:** Results show the pattern(s) the neuron/filter responds to — used for visualization and interpretation

Adversarial Examples

- Adversarial examples are **inputs intentionally perturbed by small, often imperceptible noise** that cause a neural network to make **incorrect or high-confidence wrong predictions**, even though the modified image looks unchanged to humans.

Why they Occur?

- Neural networks learn **high-dimensional decision boundaries** that can be manipulated.
- Small perturbations aligned with the model's **gradient directions** can dramatically change model output.
- Model relies on **non-robust features** that are predictive but brittle.

What They Show?

- **Vulnerability:** Even state-of-the-art CNNs can be fooled with tiny changes invisible to humans.
- **Lack of robustness:** High accuracy does not imply model stability.
- **Feature misalignment:** Networks rely on patterns humans do not perceive.
- **Transferability:** An adversarial example crafted for one model can fool different models.

References

- Guide to CNNs for CV – Khan et al. (2018)
- Deep Learning with Python – Chollet (2018)
- Deep Learning in Computer Vision – Awad & Hassaballah (2020)

- Deep Learning for Vision Systems by Mohamed Elgendy (2020)