Defense Demo

A Comparative Visualization of Classical and Deep Learning Approaches to Digital Image Analysis

Companion Project to the Bachelor Thesis
The Role of Deep Learning in Digital Image Analysis

Fabjan Elezi
Faculty of Economics, Business Informatics
Universiteti Metropolitan Tirana

June 2026

Abstract

This document accompanies a browser-based interactive demonstration built to illustrate the central question of my bachelor thesis: how deep learning has changed the way computers analyze images. The demo places two approaches side by side. On one side, three classical feature-extraction techniques — grayscale conversion, gradient-based edge detection, and Histogram of Oriented Gradients — are computed directly in the browser. On the other side, the same input image is sent to a convolutional vision model that returns its top five predicted labels. Both processes run in real time on any image the viewer drops onto the page.

The purpose of the demo is not to outperform either approach but to make the contrast between them visible. The classical pipeline shows what a human engineer once had to design by hand. The deep learning side shows what a model learns to extract on its own when given enough labeled data.

1. Introduction

Image analysis has changed more in the last decade than in the four decades preceding it. Before 2012, building an image classifier meant carefully designing features — measurable properties of an image such as edge orientations, color histograms, or texture statistics — and feeding those features into a classical machine learning model such as a Support Vector Machine. The success of the system depended heavily on the skill of the engineer who chose the features.

Deep learning replaced that workflow. Convolutional Neural Networks learn their own features directly from labeled examples, without any human deciding in advance what to measure. The thesis this project accompanies explores why that change happened, what its consequences have been, and what the current limits are. The demo gives the examiner something concrete to interact with while we discuss it.

2. Technology Stack

The application is intentionally lightweight. It runs entirely in a modern web browser, requires no installation by the viewer, and uses only standard web technologies.

Component	Technology	Role
Frontend framework	React 19 (functional components, hooks)	Manages UI state and the drag-and-drop interaction.
Build tool	Vite	Bundles the source, serves the development build, produces the pro
Image processing	HTML5 Canvas API	All classical feature extraction (grayscale, edge map, HOG) runs as
File handling	FileReader API	Reads a dropped image into a base64 data URL for both display an
Deep learning inference	Vision model REST API	Returns the top-5 classification result for each uploaded image.
Styling	Inline JavaScript style objects	No external CSS framework; all styling is colocated with the compon

Language	JavaScript (JSX)	Single-file component — no TypeScript build step required.

No native libraries, no Python backend, no GPU. Everything that needs to be visual runs locally on the user's machine; the only network call is the classification request.

3. System Architecture

The application is a single React component, *DefenseDemo*, that owns all state and renders four logical regions of the UI: a header, an input panel with the drop zone, a results panel with the top-5 predictions, and a visualization panel showing the four classical canvases.

When the user drops an image, the following sequence runs:

• **Step 1 — File ingestion.** The browser fires a *drop* event. The handler extracts the File object and passes it to the *processImage* function.
• **Step 2 — Base64 conversion.** A FileReader reads the file and emits a base64-encoded data URL.
• **Step 3 — Local visualization.** The data URL is loaded into an Image element. Once the image fires its *onload* event, the function draws it into the original canvas, runs the grayscale conversion, then reuses the grayscale array to compute the HOG and edge-map canvases.
• **Step 4 — Remote inference.** In parallel with the drawing, the base64 string is sent to the vision model API. When the response arrives, the top-5 list is parsed from JSON and stored in React state.
• **Step 5 — Render.** React re-renders the prediction bars and the explanatory text, completing the pipeline.

4. Implementation Details

4.1 Classical feature extraction

Three classical techniques are implemented from first principles:

Grayscale conversion. Each pixel's red, green, and blue channels are combined into a single luminance value using the ITU-R BT.601 weights (0.299, 0.587, 0.114). The result is stored in a Float32Array, which is later reused by both the edge and HOG extractors. Reusing the same array avoids three separate grayscale passes and keeps the demo responsive even for large images.

Edge detection. A simplified gradient operator computes horizontal and vertical intensity differences at every pixel. The edge strength at each location is the magnitude of the resulting gradient vector. The output is rendered as a bright-on-dark canvas where strong edges appear white.

Histogram of Oriented Gradients (HOG). The image is divided into 8×8 pixel cells. In each cell, the average gradient direction is computed using *atan2* on the summed horizontal and vertical gradients. A short line is then drawn through the center of the cell, oriented along the dominant edge direction and faded according to the gradient magnitude. The result resembles a pencil sketch that captures the underlying shape structure of the image — exactly the

representation that HOG uses numerically when paired with a Support Vector Machine.

4.2 Deep learning classification

The deep learning side is implemented as a single asynchronous function, *classifyImage*. It builds a POST request to the vision model endpoint, attaches the base64-encoded image, and instructs the model to return a strict JSON array of five (label, confidence) pairs ordered by confidence.

Running the model behind an API rather than locally is a deliberate choice. Pre-trained vision networks are hundreds of megabytes; downloading and loading one in the browser introduces a delay that harms the live demo experience. The remote endpoint returns results in roughly one to three seconds and requires no setup on the viewer's side.

4.3 User interface

The UI uses a dark muted palette with gold accents — chosen to look presentable in a defense setting rather than playful. The layout is a CSS Grid with the upload area on the left and the prediction list on the right, sized to fill the viewport without scrolling. Below the main row, a secondary section displays the four feature visualizations and a short explanatory paragraph that updates as soon as a prediction arrives.

All accessibility basics are respected: the file input is keyboard reachable through the drop zone's click handler, drag and drop is supplemented by a file picker fallback, and color is not the only indicator of state (a textual label and a pulsing dot also report loading status).

5. How It Works in Plain Terms

From the viewer's perspective the application has only one thing to do. The viewer drops or selects an image. Within a second, four small grayscale-style canvases appear, each showing a different classical interpretation of the same image. Within another second or two, a list of five predicted labels appears on the right with a confidence percentage and a horizontal bar. A short paragraph at the bottom names the most confident label and reminds the viewer what the HOG canvas next to it represents.

Underneath, the demonstration is doing two completely different kinds of work at the same time. On the local machine, JavaScript is scanning the pixels of the image and applying small mathematical operations that have been known to computer vision researchers since the 1980s and 1990s. On a remote server, a neural network trained on millions of labeled images is running a much larger, learned set of operations to produce its best guess at what is in the image. The two outputs land on the screen at almost the same moment, and the contrast between them is the entire point of the demo.

6. Results and Reflection

The demo achieves what it set out to do. It runs at interactive speed in a standard browser; it works on any image type the browser can read; it produces visually clear contrasts between hand-crafted and learned features; and it requires no environment setup from anyone viewing

it. It can be opened on a laptop in front of a defense committee and used live without preparation.

Quantitative benchmarks on standard datasets (CIFAR-10, ImageNet) consistently show deep learning outperforming HOG plus SVM by a wide margin — figures often cited are around 54 percent for HOG plus SVM versus 76 percent or higher for a moderately sized CNN on CIFAR-10, with even larger gaps on harder datasets. These numbers are reported in the thesis in detail; the demo itself does not evaluate accuracy, only visualizes the difference in approach.

7. Conclusion

The demo distills a single comparison into something the audience can interact with. It shows what classical computer vision actually computes when it looks at an image — gradients, edges, oriented histograms — and places that next to a modern learned classifier's output on the same image. For a defense focused on the role of deep learning in image analysis, that side-by-side is more persuasive than any benchmark table can be, because it makes the difference between engineering and learning concretely visible.

8. References

• Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. CVPR 2005.

• Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. NeurIPS 2012.

• He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. CVPR 2016.

• Howard, A. G. et al. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861.

• ITU-R Recommendation BT.601. Studio encoding parameters of digital television.

• Mozilla Developer Network. Canvas API, FileReader API, and Drag and Drop API documentation.