Identifying Eye Diseases Using CNN

By Zuleyka, Jose & Lidzy

Computer vision—the science of teaching machines to “see” and interpret images—has fueled innovations from autonomous vehicles to smartphone face recognition. A cornerstone of this field is the Convolutional Neural Network (CNN), a deep-learning architecture uniquely suited to extracting visual patterns. In this article, we first review the key ideas behind neural networks and CNNs and how they work, explain why PyTorch is the best framework for building them, and then dive into our real-world case study: automatic detection of ten eye diseases using color fundus photography.

From Neurons to Deep Networks

At its simplest, a neural network is a cascade of interconnected units—neurons—that each perform two steps. First, they compute a weighted sum of their inputs plus a bias term:

Then they apply a nonlinear activation function (such as ReLU or sigmoid) to z, allowing the network to model complex relationships.

- **Inputs** might be pixel colors in an image.
- **Weights** are learned multipliers that tune how influential each input is.
- **Bias** shifts the activation threshold, helping neurons activate even with small input signals.

Neurons are organized into layers. The **input layer** ingests raw features, **hidden layers** learn progressively more abstract representations (edges, textures, shapes), and the **output layer** delivers class probabilities. Multiple hidden layers enable hierarchical learning, where simple features combine into ever more sophisticated patterns.

Why Convolutional Neural Networks?

Standard (fully connected) networks treat each pixel as an independent feature, flattening an image into a long vector. CNNs preserve two-dimensional structure by introducing two operations:

1. **Convolutional layers** apply small, trainable filters (kernels) that slide over the image to detect local features—think of them as tiny pattern detectors that identify edges, color transitions, or textures.
2. **Pooling layers** (e.g., 2×2 max-pool) downsample feature maps, retaining only the strongest signals and reducing both spatial size and computation.

By sharing filter weights across the image, CNNs dramatically reduce the number of parameters, gain shift-tolerance (the ability to recognize a feature even if it moves slightly), and excel at building complex visual hierarchies.

Why PyTorch?

PyTorch has become the de facto standard in research and industry because it speeds up your code by using GPU acceleration behind the scenes, allowing you to focus on building and training your models while it handles the performance improvements. GPU (Graphics Processing Unit) is better for handling lots of small, repetitive calculations (like the ones needed for training a machine learning model). CPU (Central Processing Unit) is slower when it comes to many repetitive tasks.

Leading AI labs—from OpenAI to Tesla—rely on PyTorch for rapid prototyping and production deployment.

Case Study: Detecting Eye Disease with CNNs

Motivation & Dataset
Millions suffer preventable vision loss because eye diseases go undetected until too late. In many underserved regions, specialists are scarce. Our solution uses automatic, early screening powered by CNNs to flag potential problems for further clinical review.

We assembled a dataset of 16,242 color fundus images spanning ten classes: nine disease categories (Diabetic Retinopathy, Glaucoma, Macular Scar, Optic Disc Edema, Central Serous Chorioretinopathy, Retinal Detachment, Retinitis Pigmentosa, Myopia, Pterygium) plus healthy controls. We split the data 80% for training, 10% for validation, and 10% for testing.

Because some classes were under-represented, we applied random oversampling on the training set only—duplicating minority-class images until all classes reached equal counts. This ensures our model learns to recognize all diseases rather than biasing toward the most common.

Creating the Model

Preprocessing

Before feeding images into our network, we organized every retinal scan and its diagnosis into a single Pandas DataFrame—each row stores the file path and its associated disease label. This table makes it easy to shuffle, split, and inspect the data programmatically.

For the training set, we then applied a rich sequence of image transforms to teach the model to generalize beyond the exact pictures it sees. First, each image is resized to 128×128 pixels so that all inputs share the same dimensions. Finally, we convert the result to a PyTorch tensor and normalize each color channel to have zero mean and unit variance (using the standard ImageNet statistics). For validation and testing, we simplify this pipeline—images are only resized, converted to tensors, and normalized—to produce consistent, unaugmented inputs for fair evaluation.

Hyperparameters

We trained with the Adam optimizer and an initial learning rate of 0.001, a sweet spot that balances stability with convergence speed. A batch size of 32 strikes a good compromise between GPU memory limits and the quality of our gradient estimates. Training runs for up to ten full passes through the data (epochs), but we guard against overfitting by stopping early if the validation loss fails to improve for five consecutive epochs.

Building the CNN Model

Our `EyeDiseaseCNN` follows a classic three-block convolutional design. The first block takes the 3-channel input and applies 32 learned 3×3 filters, each followed by a ReLU activation; a 2×2 max-pooling layer then halves the spatial resolution from 128×128 to 64×64. The second block repeats this pattern with 64 filters, reducing the feature maps to 32×32. The third block uses 128 filters and pools down to 16×16. At this stage, we have 128 feature maps of size 16×16 each—32,768 activations in total. We flatten that volume into a single 32,768-dimensional vector and pass it through a fully connected layer of 256 neurons (with ReLU and 50 % dropout to prevent overfitting). A final dense layer with 10 outputs produces the raw class scores, which our loss function (cross-entropy) and softmax interpret as probabilities for the ten disease categories.

Training & Performance

- **Validation accuracy** climbed from ~66 % → ~89 % over ten epochs.
- **Validation loss** dropped from ~0.96 → ~0.34.
- **Final test accuracy**: ~89 %.
- **Confusion insights**: Distinct conditions (e.g. Pterygium) achieve near-perfect detection; subtler ones (early Glaucoma vs. healthy retina) remain challenging.

Conclusion

This pipeline demonstrates how PyTorch-built CNNs—powered by robust preprocessing, balanced sampling, and best-practice training—can deliver high-accuracy screening for eye diseases. By combining deep-learning fundamentals with a healthcare application, we move closer to preventing avoidable vision loss worldwide.

—

Feel free to connect or reach out if you have questions!

Link to GitHub: https://github.com/zuleykaben/Exploring-Convolutional-Neural-Networks