Version January 5, 2025
Preprint typeset using LaTeX style openjournal v. 09/06/15

A NEURAL NETWORK'S TALE OF TAILS

Parth Sinha
Oxford Brookes University
Version January 5, 2025

Abstract

This project explores the design and implementation of a feedforward neural network for binary image classification, focusing on distinguishing between images of cats and dogs. Using a compact architecture, the network processes 64×64 RGB images through progressively smaller hidden layers, culminating in a single output neuron for classification. The implementation incorporates essential preprocessing steps, including normalization and standardization, ensuring optimal input preparation for training. Forward propagation utilizes sigmoid activations and He initialization to maintain stability, while backpropagation efficiently computes gradients using the chain rule. The training process employs mini-batch gradient descent with momentum and an exponentially decaying learning rate to achieve convergence. Binary cross-entropy loss serves as the guiding metric, enabling the model to refine its decision-making. The system is designed with modular, object-oriented principles for scalability and clarity, while techniques such as OpenCV integration ensure robust data handling. Although the current implementation is foundational, it lays the groundwork for incorporating advanced strategies like batch normalization and dropout to enhance performance. This work demonstrates the principles of neural networks and their practical application to real-world classification tasks, blending theoretical depth with hands-on AI development.

Subject headings: Artificial intelligence, neural networks, binary classification, image processing, deep learning, and binary cross-entropy loss

1. INTRODUCTION

As a last-year computer science student with a passion for AI and machine learning, I have been diving deep into the fascinating world of neural networks. And what better way to learn than by teaching a machine to do the impossible: decide whether a blurry picture is a majestic canine or an aloof feline? This journey into binary image classification is not just a technical exercise; it is a battle of logic versus the whims of fur and whiskers. With a neural network armed with 12,288 input neurons, a layer cake of computations, and a mission to classify cats and dogs, this project combines the charm of AI with the thrill of watching algorithms learn. Spoiler: no treats were required to train this model, but plenty of mathematical snacks were consumed.

2. EXPLAINATION

This implementation showcases a feedforward neural network designed for binary image classification, specifically distinguishing between images of dogs and cats. At its core, the network processes input images by passing them through a series of layers, transforming the raw pixel data into meaningful patterns, and ultimately outputting a classification decision. The architecture begins with an input layer capable of handling images with dimensions of 64×64 pixels and three color channels (RGB). This translates to a total of $12,288$ input neurons ($64 \times 64 \times 3$), which serve as the starting point for the network computations. These inputs are passed through multiple hidden layers of decreasing size: $256, 128, 64$, and 32 neurons. Progressive reduction in size enables the network to extract and compress meaningful features while ignoring irrelevant information. At the end of this pipeline, a single output neuron provides the final classification. If the output value is greater than 0.5, the image is classified as a dog, whereas a value of 0.5 or less indicates a cat.

To ensure the network receives input data in a format that facilitates effective learning, a preprocessing pipeline is applied to the images. The first step is normalization, which scales all pixel values to the range [0, 1] by dividing each pixel value by 255. This ensures consistency in the scale of input data. The normalized data is then standardized by centering it around a mean (μ) and scaling it by the standard deviation (σ). This step ensures that the inputs are not only on a similar scale but also have a consistent distribution, which is critical for the stable and efficient training of the network.

Once the input images have been preprocessed, they are passed through the network via a process known as forward propagation. For each layer, the network computes a weighted sum of the inputs from the previous layer, adds a bias

sinhaparth555@gmail.com

term, and applies an activation function. Mathematically, this process can be described as,

$$z^{[l]} = W^{[l]}a^{[l-1]} \; + \; b^{[l]}$$

where $W^{[l]}$ represents he weight matrix for the layer, $a^{[l-1]}$ represents the activation output of the previous layer, and $b^{[l]}$ is the bias vector. The activation function applied to $z^{[l]}$ is the sigmoid function, defined as $\sigma(z) = \frac{1}{1+e^{-z}}$. This function squashes the output into the range [0, 1], making it suitable for binary classification tasks. To ensure the weights are initialized effectively, the network employs He initialization. In this method, weights are sampled from a normal distribution with a mean of 0 and a variance of $\sqrt{\frac{2}{n^{[l-1]}}}$, where $n^{[l-1]}$ is the number of neurons in the previous layer. This initialization prevents the variance of activations from diminishing or exploding as they propagate through the network.

To evaluate the performance of the network, binary cross-entropy loss is used. This loss function is specifically designed for binary classification tasks and quantifies the difference between the predicted probability $\hat{y}$ and the true label (y). The loss is given by the formula:

$$L = -[y \; \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

By penalizing incorrect predictions, the loss function guides the network toward minimizing errors during training.

Training the network involves backpropagation, a process that computes the gradients of the loss function with respect to each parameter. For the output layer, the error is calculated as:

$$dz^{[L]} = \hat{y} - y$$

where $\hat{y}$ is the predicted output and y is the true label. Gradients for the weights and biases are then computed as:

$$dW^{[L]} = dz^{[L]}a^{[L-1]^T} \text{ and } db^{[L]} = dz^{[L]}$$

For hidden layers, the error is propagated backward using the chain rule:

$$dz^{[l]} = W^{[l+1]^T}dz^{[l+1]} \odot \sigma'(z^{[l]})$$

where $\sigma'(x) = \sigma(x)(1-\sigma(x))$ is the derivative of the sigmoid activation function. These gradients are used to update the parameters during the optimization step.

To optimize the parameters, the network employs mini-batch gradient descent with momentum. This method updates each parameter θ using the formula $v = \beta v - \alpha\nabla J(\theta)$ and $\theta = \theta + v$, where β is the momentum coefficient (set to 0.9), α is the learning rate and $\nabla J(\theta)$ represents the gradient of the loss function with respect to θ. Momentum helps accelerate convergence by smoothing out oscillations, especially in areas with high curvature. The learning rate decays exponentially over time according to the formula $\alpha_t = \alpha_0 \cdot 0.95^t$ where t is the epoch number. This decay ensures that the learning rate decreases as training progresses, allowing the network to fine-tune its parameters. The implementation processes training data in mini-batches of 32 images, which strikes a balance between computational efficiency and stability in gradient updates. The data loading mechanism is robust, leveraging OpenCV to validate image formats and directory structures. The code-base is structured using an object-oriented approach, with separate classes handling different responsibilities. The NeuralNetwork class manages the overall architecture and training loop, while the Layer class handles computations and parameter updates for individual layers.

While this implementation is designed for clarity and educational value, there is potential for enhancements. Techniques such as batch normalization could address internal covariate shifts, dropout regularization could reduce overfitting, and advanced architectures like residual connections could improve performance. Nevertheless, this implementation provides a strong foundation for understanding the principles of deep learning and binary image classification.

3. CONCLUSION

The training results reveal a gradual improvement in accuracy across epochs, starting at 56.6% in the first epoch and reaching 60.8% by the fifth. The most significant progress occurs between epochs 1 and 2, after which the accuracy gains begin to plateau—a common trend in neural network training. While the network performs slightly better than random chance (50% for binary classification), the overall accuracy of around 60% suggests there is significant room for improvement. Potential enhancements could include increasing the number of training epochs, fine-tuning the learning rate, introducing additional regularization techniques, deepening or widening the network architecture, or implementing data augmentation to enrich the training dataset.

APPENDIX

GitHub Repository

The code for this project is available on GitHub at:
https://github.com/sinhaparth5/cat-dogs-classifier

Kaggle Dataset

The dataset used in this project can be found at:
https://www.kaggle.com/datasets/tongpython/cat-and-dog