# Developing an Image Binary Classifier

Created: 02/02/21 by Tom Lever
Updated: 02/03/21 by Tom Lever


## Abstract

An image binary classifier classifies images as belonging to one category or another. One image binary classifier is an elementary neural network that takes in a column vector representing a flattened, normalized image and outputs a category index. This paper describes the sequential flow of this image binary classifier. This paper subsequently discusses creating a larger neural network; the value in improving deep neural networks of and the process of convolution; and applying professional deep learning frameworks and building Convolutional Neural Networks from scratch.


## 1. Introduction: An Extremely Simple Neural Network

An extremely simple neural network is used to estimate a $y$-coordinate (e.g., $5$) given an $x$-coordinate (e.g., $2$) along a line (e.g., $y(x) = 2x + 1$). The neural network will be trained based on a given column vector $\vec{x}$ of $x$-coordinates (e.g., a column vector of twenty uniformly distributed values between $0$ and $1$ inclusive) and a given column vector $\vec{y}$ of $y$-coordinates (e.g., $\vec{y} = 2\vec{x} + 1$). The neural network will involve one hidden fully connected layer with one neuron with one linear function with parameters weight $w$ and bias $b$ (e.g., $\hat{\vec{y}}(x) = w\vec{x} + b$). The weight $w$ will be initialized to a number drawn randomly from a normal distribution and the bias $b$ will be initialized to $0$. The weight $w$ and the bias $b$ will be honed during training. Training consists of many iterations. During one iteration, the neural network will forward propagate the column vector of training examples $\vec{x}$ through its fully connected layer to generate a column vector of predicted values $\hat{\vec{y}}$. A cost $J$ (e.g., mean squared error) will be calculated based on the column vector of ground-truth values $\vec{y}$ and the column vector of predicted values $\hat{\vec{y}}$. A partial derivative $\partial J/\partial w$ of the cost $J$ with respect to the weight $w$ will be calculated; a partial derivative $\partial J/\partial b$ of the cost $J$ with respect to the bias $b$ will be calculated. During gradient descent, the partial derivative $\partial J/\partial w$, scaled by the hyperparameter learning rate $\alpha$ (e.g., $0.1$), will be taken from the weight $w$; the partial derivative $\partial J/\partial b$, scaled by the learning rate $\alpha$, will be taken from the bias $b$. During testing, the neural network will forward propagate one $x$-coordinate and will provide an estimate of the associated $y$-coordinate.

## 2. Flattened, Normalized Images

An image in NCHW format is a $c \times h \times w$ tensor, where $c$, $h$, and $w$ are number of channels (e.g., 3, for red, green, and blue), image height (e.g., 608 pixels), and image width (e.g., 608 pixels). The elements $x_{k,y,x}$ of the image are

$$x_{1,1,1}, x_{1,1,2}, \ldots, x_{1,1,w}, x_{1,2,1}, x_{1,2,2}, \ldots, x_{1,h,w}, x_{2,1,1}, x_{2,1,2}, \ldots, x_{k,y,x}.$$

A flattened image is a column vector representing an image. The flattened image for the image above is a column vector of the elements above in the order above.

A flattened, normalized image is a flattened image with all elements between 0 and 1 inclusive.

## 3. Image Binary Classifier

An image binary classifier classifies images as belonging to one category or another. One image binary classifier is an elementary neural network that takes in a column vector representing a flattened, normalized image and outputs a confidence that the image belongs to one category.

The neural network will be trained based on a given matrix of flattened, normalized images

$$\tilde{x} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(n)} \\ x_2^{(1)} & x_1^{(2)} & \ldots & x_2^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ x_m^{(1)} & x_m^{(2)} & \ldots & x_m^{(n)} \end{bmatrix},$$

where $m$ is the number of elements in each flattened, normalized image and $n$ is the number of images. The neural network will also be trained on a given row vector of ground-truth values

$$\vec{y} = [y^{(1)} \quad y^{(2)} \quad \ldots \quad y^{(n)}],$$

each equal to category index 0 or category index 1. The neural network will involve one fully connected layer with one neuron with the linear function

$$\vec{z}(\tilde{x}) = \vec{w}^T \tilde{x} + b,$$

where parameter bias $b$ is a scalar, parameter weight vector $\vec{w}$ is a $m \times 1$ column vector, and $\vec{z}$ is a $1 \times n$ row vector. The bias $b$ will be initialized to 0; the elements of weight vector $\vec{w}$ will be drawn randomly from a normal distribution and scaled by $\sqrt{1/m}$. The neuron will also involve the logistic activation function

$$\hat{\vec{y}} = \frac{1}{1 + \exp(-\vec{z})}$$

The elements of the weight vector $\vec{w}$ and the bias $b$ will be honed during training. Training consists of many iterations. During one iteration, the neural network will forward propagate the matrix of flattened, normalized images $\tilde{x}$ through its fully connected layer to generate a row vector of predicted values $\hat{\vec{y}}$.

The binary cross-entropy cost $J$ will be calculated according to the formula

$$J = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\ln\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right)\ln\left(1 - \hat{y}^{(i)}\right)\right]$$

A vector

$$\overrightarrow{\frac{\partial J}{\partial\left(w_g\right)}}$$

of the partial derivatives of the cost $J$ with respect to the elements $w_g$ of the weight vector $\vec{w}$ will be calculated. $g$ ranges from 1 to $m$ inclusive. A partial derivative $\partial J/\partial b$ of the cost $J$ with respect to the bias $b$ will be calculated.

During gradient descent, the weight vector $\vec{w}$ will be updated according to the formula

$$\vec{w} \leftarrow w - \alpha\overrightarrow{\frac{\partial J}{\partial\left(w_g\right)}}$$

The bias will be updated according to the formula

$$b \leftarrow b - \alpha\frac{\partial J}{\partial b}$$

During testing, the neural network will forward propagate one flattened, normalized image and will provide a confidence that the image belongs to one category. To arrive at the neural network's prediction of the class index of the image, the confidence is rounded.

## 4. Batch Normalization

For large, varied images, one or more elements of the weight vector $\vec{w}$ vector and/or the bias $b$ become negative enough to drive one or more predicted values in $\hat{\vec{y}}$ to 0 and the cost $J$ to $-\infty$. A solution (besides introducing further neurons and/or fully connected layers and/or convolutional layers), is to introduce batch normalization to the neuron in the fully connected layer of our elementary neural network. The neuron will involve the functions

$$\vec{z}(\vec{x}) = \vec{w}^T\vec{x} + b, \vec{z} \in \mathbb{R}_{1\times n}$$

$$\mu = \sum_{k=1}^{n} z^{(k)}, \mu \in \mathbb{R}$$

$$v = \sum_{k=1}^{n}\left(z^{(k)} - \mu\right)^2, v \in \mathbb{R}$$

$$\varepsilon \in \mathbb{R}\ (e.g., \varepsilon = 1 \times 10^{-6})$$

$$\vec{z}_{norm}(\vec{z}) = \frac{\vec{z} - \mu}{\sqrt{v + \varepsilon}}, \vec{z} \in \mathbb{R}_{1\times n}$$

$$\gamma \in \mathbb{R}$$

$$\beta \in \mathbb{R}$$

$$\vec{z}_{SON}(\vec{z}_{norm}) = \gamma\vec{z}_{norm} + \beta, \vec{z} \in \mathbb{R}_{1\times n}$$
$$\vec{\hat{y}} = \frac{1}{1 + \exp\left(-\vec{z}_{SON}\right)}$$

The bias $b$ will be initialized to 0; the elements of weight vector $\vec{w}$ will be drawn randomly from a normal distribution and scaled by $\sqrt{1/m}$. The parameter scale $\gamma$ and parameter offset $\beta$ will be initialized as $\gamma = \sqrt{v_0 + \varepsilon}$, and $\beta = \mu_0$, where $v_0$ and $\mu_0$ are determined by forward propagating the matrix of flattened, normalized images through the neural network when the neural network is configured with initial weight vector $\vec{w}_0$ and initial bias $b_0$.

## 5. Adding Fully Connected Layers and Adding Neurons to a Fully Connected Layer

To add to a neural network a fully connected layer after the $j$th layer, the output of the fully connected layer $\tilde{a}^{\{j\}}$ will be forward propagated into the new fully connected layer as input $\tilde{x}^{\{j+1\}}$.

To add an $i$th neuron to the $j$th fully connected layer, the input $\tilde{x}^{\{j\}}$ will be forward propagated into all existing neurons and the new neuron. The outputs of each neuron $\tilde{a}_{[i]}^{\{j\}}$ will be stacked into layer output $\tilde{a}^{\{j\}}$. For example, layer output $\tilde{a}^{\{j\}}$ may be the matrix created by stacking neuron output vectors $\vec{a}_{[i]}^{\{j\}}$. Or, layer output $\tilde{a}^{\{j\}}$ may the three-dimensional tensor created by stacking neuron output matrices $\vec{a}_{[i]}^{\{j\}}$.

A simple image binary classifier may involve a binary cross-entropy cost $J$ and gradient descent, a hyperparameter learning rate $\alpha$, and an architecture with two fully connected layers each with two neurons each with a linear function, a parameter weight vector of weights $\left(w_g\right)_{[i]}^{\{j\}}$, and a parameter bias $b_{[i]}^{\{j\}}$; batch normalization, a parameter scale $\gamma_{[i]}^{\{j\}}$, and a parameter offset $\beta_{[i]}^{\{j\}}$; and a logistic activation function. During gradient descent, each parameter will be updated based on the partial derivative of the cost $J$ with respect to the parameter.

## 6. Convolution

Using two-dimensional convolution in lieu of linear functions in neurons of many-layer (i.e., deep) neural networks reduces training time and memory requirements, and improves testing accuracy and resilience to warped images. Practical convolution is identical to theoretical cross-correlation.

One-dimensional convolution is a process of "placing" a vector window / filter / kernel $\vec{w}$ over a input vector $\vec{x}$ at one end of $\vec{x}$, performing a basic convolutional operation involving the filter $\vec{w}$ and the covered sub-vector of $\vec{x}$ (designated $\vec{c}$), placing the scalar output of the basic convolution operation at the beginning of an output vector $\vec{z}$, shifting the filter $\vec{w}$ one element toward the other end of the input vector $\vec{x}$, performing the basic convolution operation, and so on. The output of the one-dimensional basic convolution operation involving filter $\vec{w}$ and sub-vector $\vec{c}$ is the product of the row vector $\vec{w}$ and the column vector $\vec{c}$. One-dimensional convolutional can occur with two- or three-dimensional tensors, as long as the shifting is in one direction.

Two-dimensional convolution is a process of placing a matrix filter $\tilde{w}$ over an input matrix $\tilde{x}$ at the upper left corner of $\tilde{x}$, performing a basic convolution operation involving the filter $\tilde{w}$ and the covered sub-matrix of $\tilde{x}$ (designated $\tilde{c}$), placing the scalar output of the basic convolution operation at the upper left corner of output matrix $\tilde{z}$, shifting the filter $\tilde{w}$ one element toward the right edge of the input matrix $\tilde{x}$, performing the basic convolution operation, and so on. After a basic convolution operation has occurred for the filter $\tilde{w}$'s right-most position, the filter $\tilde{w}$ is returned to the left-most region of the input matrix $\tilde{x}$ and is shifted down one element toward the bottom edge of the input matrix $\tilde{x}$, at which point a basic convolution operation is performed. After this basic convolution operation has been performed, the filter $\tilde{w}$ is shifted right one element, a basic convolution operation is performed, and so on. Two-dimensional convolution can occur with three-dimensional tensors, as long as the shifting is in two directions. This is relevant to passing a $(c = 3) \times h' \times w'$ filter $\tilde{w}$ over a $(c = 3) \times h \times w$ input tensor $\tilde{x}$.

## 7. Next Steps

To apply an image binary classifier, it is recommended that the darknet deep-learning framework / YOLOv4 two-dimensional Convolutional Neural Networks be used. To build an image binary classifier, it is suggested that the PyTorch deep-learning framework be used to construct a many-layer two-dimensional Convolutional Neural Network. To build an image binary classifier from scratch, it is suggested that CUDA C++ and NVIDIA's cuDNN library be used to construct a two-dimensional Convolutional Neural Network, if cuDNN is error-free and compatible with existing hardware.

## 8. Conclusion

In this paper, the sequential flow of an elementary image binary classifier involving a binary cross-entropy cost and gradient descent, a learning rate, and an architecture with one fully connected layer of one neuron with a linear function, a weight vector, a bias; batch normalization, a scale, and an offset; and a logistic activation function was described. A similar image binary classifier with two fully connected layers each with two neurons was described. The value in improving deep neural networks of and the process of convolution was described. Applying a professional deep-learning framework and a two-dimensional Convolutional Neural Network to image binary classification, building an image binary classifier using the PyTorch deep-learning framework and two-dimensional Convolutional Neural Networks, and building an image binary classifier and two-dimensional Convolutional Neural Networks from scratch were discussed.