Image Anomaly Detection with Neural Networks

Ross Churchley[1], Omeiza Olumoye[2], Pawan Patel[3], Jonathan Smith[4], Parker Williams[5], Ruiyu Yang[3], Jesse Berwald[6], and Janet Keel[6]

[1]Simon Fraser University
[2]University of Texas
[3]Indiana University
[4]Acadia University
[5]University of California
[6]Target Corporation

September 11, 2015

1 Introduction

Neural networks are a statistical learning paradigm inspired by the structure of the neurons in the human brain. The brain learns by forming and strengthening connections between neurons when exposed to various stimuli. Analogously, neural networks "learn" by adjusting connecting maps between computational layers when exposed to data according to some objective function.

Interest in neural networks has periodically waxed and waned since the development of the backpropagation algorithm in the mid-1970s [12]. Recently, computational improvements have caused a resurgence in neural networks: modern GPU processing is now highly optimized for 2-dimensional convolution, allowing for much deeper networks. Architectural developments have also driven success in this field; competitions have seen dramatic new records set for accuracy and speed using algorithms that are less complex than the previous record holders. New and notable results include [6, 5]. GoogleNet has also enjoyed recent success [11] using networks inspired by models for the visual cortex, and their creation of a special "inception layer" is similar to the network-in-network approach by Lin et al. [9].

Many open questions and technical challenges still surround convolutional neural networks. It is often unclear what situations a new technique or structure will apply to and the circumstances under which it will fail. Still, the current approaches suggest that as-yet-undiscovered techniques could lead to sparser, more accurate networks. Known theoretical bounds have been documented by Arora et al. [1].

2 Convolutional Neural Networks

A *convolutional neural network (CNN)* is a sequence of computational layers, each "feeding forward" into the next. Throughout this section, we will assume the input for a network is an $n \times n$ array representing an image. A layer can be of several different types:

- A *convolution layer* applies m different $k \times k$ learned convolution kernels to the image, producing a group of m $(n-k+1) \times (n-k+1)$ images. Over the course of the learning process, the kernels in the first convolution layer will typically specialize to extract low-level features like edges or corners of an image (see Figure 1). Deeper convolution layers will recognize patterns of these features in order to extract higher-level features.

- A *max-pooling layer* is a deterministic step that reduces the dimension of an image by a factor of k. It simply divides an input image into $k \times k$ sub-blocks and takes the maximal pixel value in each block.

- A *fully-connected layer* discards the positional information of the input image and treats it as an n^2-dimensional vector x. It then applies a learned affine transformation $Wx + b$ to aggregate information from the individual feature maps discovered by the convolutional layers.

Networks typically end with at least one fully connected layer, with the output dimension set to the number of categories one wishes to divide the images into.

After each layer, an activation function is called. This corresponds biologically to the activation potential of a neuron firing, but also serves a purpose of rescaling the data layer to layer. As we will see, the choice of activation function can be vital and should be made within the context of the data.

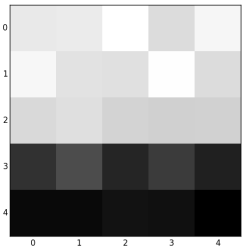

Figure 1: A 5×5 convolution kernel which has learned to detect thick edges.

Today, the most popular activation function for deep neural networks is the *rectifier* or *rectified linear unit (ReLU)* [8]

$$f(x) = \max\{0, x\}.$$

In this paper, we also consider the so-called *leaky rectifier* [10]

$$f(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0. \end{cases}.$$

Other activations that can be found in the literature include $\tanh(x)$ and the sigmoid function $(1 + e^{-x})^{-1}$. The *softmax* activation function

$$\phi(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

is often added to the fully-connected layer at the end of the network, so that its output can be interpreted as the network's confidence, expressed as a probability, that the image falls into the jth category

To make the network "learn", one must specify an objective function for it to minimize. In this report, we use the *cross-entropy* function

$$L = -t \log(p) - (1 - t) \log(1 - p),$$

where $p, (1 - p)$ is the output of the network and $t, (1 - t) \in \{0, 1\}$ indicate the "true distribution".

The learning process uses a backpropagation algorithm to adjust the parameters given a labeled training dataset. Layer by layer, the gradient of the

objective function L is computed with respect to the learned parameters W at that layer, and the parameters are adjusted in that direction, i.e.

$$W_{i+1} = W_i - \epsilon \nabla L(W_i)$$

where ϵ is a fixed number called the *learning rate* chosen in advance. Choosing ϵ for a particular dataset and network is also an art, and can qualitatively change the learning progress.

3 Anomalous images

Retailers currently manage very large numbers of digital photographs, typically within a very narrow scope. Nearly all images are subject to some form of image manipulation, and because of the sheer number of images, some errors are inevitable. Leveraging the focused nature of the data (photos of similar objects, already categorized) it is hoped that it may be feasible for a convolutional neural network to learn to detect anomalous images.

Image manipulation errors, however, may come in many forms. Some high-profile cases have three models with seven hands between them due to bad photo compositing; more commonly, overzealous smudging, erasing, or stretching leads to models with bizarre proportions, artificial smoothness, or missing parts.

Given the wide range of potential anomalies — and lack of a labeled set of "anomalous" pictures at the scale we need for neural network training — we need to focus on a particular type of error which is easily reproduced programmatically. In this report, we consider the problem of detecting white rectangles added to an image, simulating the type of error where portions of an image are damaged or deleted.

4 Methodology

We built several convolutional neural networks using the Lasagne Python library [4], which is built on top of Theano [2, 3]. Each network was trained up to four datasets using the GPU architecture on the Minnesota Supercomputing Institute's Mesabi compute cluster. The four datasets used in this paper are as follows:

- Binary (10000 images). Solid black backgrounds.

Figure 2: Examples of images from the Binary, Solid, Flickr, and Target datasets

- Solid (10000 images). Solid backgrounds of varying brightnesses.
- Flickr (30000 images). Photos from Flickr, chosen from several of the most popular tags (e.g. "canon", "nikon", "travel").
- Target (131784 images). Target's complete catalogue of product images, each consisting of a product and/or model against a white background.

Each image was resized to 64×64 pixels and converted to greyscale. To each image, a white 16×16 square was added with probability $\frac{1}{2}$ in a position chosen uniformly at random (among positions having the square entirely inside the image). The resulting greyscale image was then read in as a 64×64 array and normalized so each pixel took values in $[0, 1]$.

Each dataset was split into three parts: a *training set* (80% of the images) was used for network training; a *validation set* (10%) used to measure the performance of the network during the training process; and a *test set* (10%) used to measure the performance of the network after it has finished learning. In Section 5, we report for each network the average value of the objective function on the test set (the *test loss*) and the proportion of images in the test set correctly identified by the network (the *test accuracy*).

The learning process for each network and dataset was run for 100 *epochs* (full passes through the training set) using the Nesterov momentum update scheme with a learning rate of 0.001 and momentum of 0.9. In Section 8 we plot for each network the loss and accuracy on the validation set at each epoch in the learning process.

5 Comparison of network architectures

5.1 Logistic regression

The most basic neural network is the logistic regression, consisting of a single fully-connected layer with the softmax activation function.

Dataset	Test Loss	Accuracy (%)
Binary	0.0007	100.0%
Solid	0.8464	74.9%
Flickr	1.0338	55.0%
Target	—	—

As a simple and standard model, linear regression is a useful baseline to compare our convolutional networks with. It does well on the Binary dataset, as one might expect: applying a roughly-uniform W is similar to computing the average brightness of pixels in the image, from which the presence or absence of a white square on a black background can easily be determined. But this strategy does not fare as well when the brightness of the background can vary, as it does in the Solid dataset. And on the noisy Flickr dataset, this simple model is not much better than random.

Although logistic regression outperforms coin-flipping, the training process does not produce significantly better results than a randomly-initialized W (see Figures 3, 4, and 5) In fact, single-layer networks are inherently incapable of detecting translation-invariant features like shapes; if asked to do so, the best they can do is a crude heuristic like average brightness.

5.2 Multi-layer CNN with rectifier activations

To recognize shapes, we add convolutional layers to the beginning of the network. Convolutional layers are designed to recognize translation-invariant features (whose size depends on the size of the convolution filters and the number of layers). Our network is based on the LeNet architecture [7], and features two convolutional layers, each consisting of eight learned 5×5 convolutional filters and a deterministic 2×2 max pooling step. The convolutional layers are installed with the widely-used rectifier activation function

$$f(x) = \max\{0, x\}.$$

The output of the second max pooling step is sent to a final fully-connected layer with a softmax activation; as with linear regression, the network is trained to minimize cross-entropy.

Dataset	Test Loss	Accuracy (%)
Binary	0.0000	100.0%
Solid	0.0001	100.0%
Flickr	0.6931	50.2%
Target	—	—

This convolutional neural network gives promising results on the Binary and Solid datasets (see Figures 6 and 7). Unlike the previous subsection's single-layer network, it reaches 100% accuracy on the Solid dataset almost immediately. However, it falls into a curious failure state on the Flickr dataset (see Figure 8). At some point in the learning process, the learned parameters enter a domain where the rectifier activation outputs 0 for every input image. The network therefore outputs the same answer for every input (the value based on the sign of the bias parameter learned in the fully-connected layers). Ideally, the learning process would move on to better parameters, but the gradient of the rectifier is also zero on this domain, and the parameters do not, in fact, escape.

To escape the black hole of 50% accuracy, we must replace rectify with a different activation function — preferably one with similar advantages in speed and accuracy that has been observed for the rectifier in the literature.

5.3 Multi-layer CNN with leaky-rectifier activation

Our best-performing neural network uses the same LeNet-inspired archetecture as in the previous subsection, with one change: the leaky rectifier

$$f(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0. \end{cases}$$

is used as the activation function for the convolution layers instead of the non-leaky version.

The leaky rectifier performs almost identically to its non-leaky counterpart on the Binary and Solid datasets (see Figures 9 and 10). It also does not fall into the same "black hole" and manages to achieve better-than-random performance on the Flickr and Target datasets.

Dataset	Test Loss	Accuracy (%)
Binary	0.0000	100.0%
Solid	0.0006	100.0%
Flickr	0.6724	63.3%
Target	0.6258	65.1%

6 Conclusion

While we did not achieve the accuracy needed for industry use, we succeeded in creating a neural network that performs better than random chance at finding white rectangles added to images. Our network is 65% accurate even on the difficult Target dataset, where unaltered images may include predominantly white backgrounds.

Model	Binary	Solid	Flickr	Target	Time
Logistic regression	100%	75%	55%	—	1.00
CNN w/ rectify	100%	100%	50%	—	2.94
CNN w/ leaky-rectify	100%	100%	63%	65%	2.96

Table 1: Compare the performance of the three models with respect to accuracy (final % correct on test dataset) and computation time (seconds per epoch per 1000 images).

Previous work has observed the performance of the standard and leaky rectifiers as being nearly identical when used as an activation function in convolutional neural networks [10]. We have demonstrated a case in which they are qualitatively different: the standard rectifier traps the learning process in a "black hole of 50% accuracy" while the leaky rectifier allows the network to learn.

Due to time constraints, we have only studied simple architectures in this report. Future work should focus on adapting network in network structures similar to Lin et al. [9] and GoogLeNet [11] in ways that might exploit features of Target's data. Finally, we hope that new structures and techniques might be discovered, such as methods to compensate for unevenly-sized categories, to tackle the problem of detecting more general anomalies.

7 Acknowledgements

We would like to deeply thank Jesse Berwald and Janet Keel from the Target Corporation for their enthusiasm and encouragement. We would also like to thank the University of Minnesota for their hospitality in hosting us, the Institute for Mathematics and its Applications for organizing such collaborative and exciting event, the Minnesota Supercomputing Institute for their resources and technical support and the Pacific Institute for the Mathematical Sciences for its support of some of our members.

References

[1] Sanjeev Arora, Aditya Bhaskara, Rong Ge, Tengyu Ma (October 2013) Provable Bounds for Learning Some Deep Representations. *arXiv:1310.6343*

[2] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio. (2012) Theano: new features and speed improvements. *NIPS deep learning workshop.*

[3] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. Theano: A CPU and GPU Math Expression Compiler. *Proceedings of the Python for Scientific Computing Conference (SciPy)* 2010. June 30–July 3, Austin, TX

[4] Sander Dieleman et al. (2015). Lasagne: First release. *GitHub.* doi:10.5281/zenodo.27878

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik (October 2014) Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv:1311.2524*

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems* 25, 1097–1105.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

[8] Yann LeCun, Yoshua Bengio, Geoffrey Hinton (2015). Deep learning. *Nature* 521: 436–444.

[9] Min Lin, Qiang Chen, Shuicheng Yan (March 2014) Network In Network. *arXiv:1312.4400*

[10] Andrew Maas, Awni Hannun, Andrew Ng (2014). Rectifier nonlinearities improve neural network acoustic models. `http://web.stanford.edu/~awni/papers/relu_hybrid_icml2013_final.pdf`

[11] C. Szegedy, W. Liu, Y. Jia, et al. Going Deeper with Convolutions. (September 2014) *arXiv:1409.4842*

[12] P. Werbos (1974). Beyond regression: New tools for prediction and analysis in the behavioural sciences. *PhD dissertation, Committee on Appl. Math., Harvard University.*

8 Appendix: Training process charts

In this section, we plot for each experiment the *validation accuracy* and *validation loss* at each epoch in the learning process. The validation accuracy measures the proportion of images in the validation set for which the network is more than 50% confident in the correct answer. The validation loss is the average value of the objective function minimized during the learning process. As explained in earlier sections, we always use the cross-entropy function

$$L = -t \log(p) - (1 - t) \log(1 - p),$$

where $p, (1 - p) \in [0, 1]$ represent the network's confidence in the presence or absence of a white square in the image and $t, (1 - t) \in \{0, 1\}$ indicate the correct answer. This quantity is always non-negative and is typically small (say, below 2) unless the network is very confident in an incorrect answer.

8.1 Logistic regression

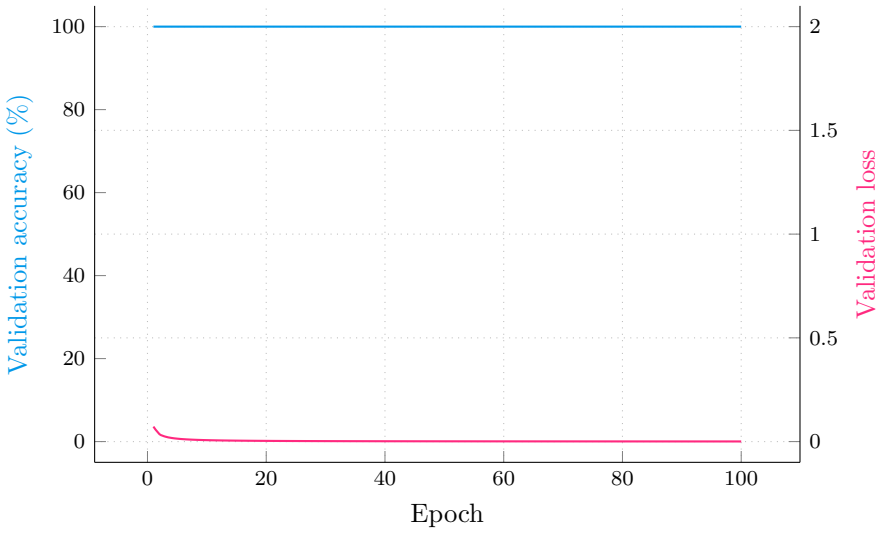

Figure 3: Learning progress of logistic regression on the binary data set

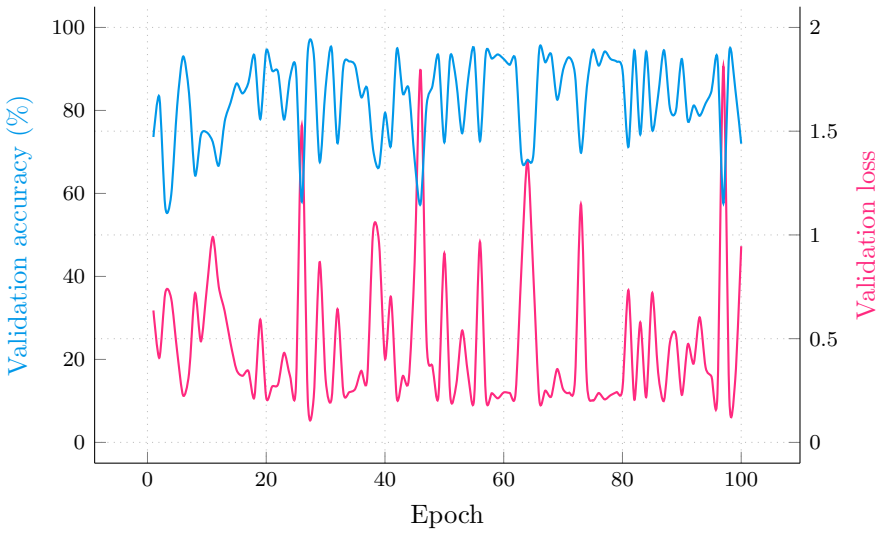

Figure 4: Learning progress of logistic regression on the Solid greyscale data set

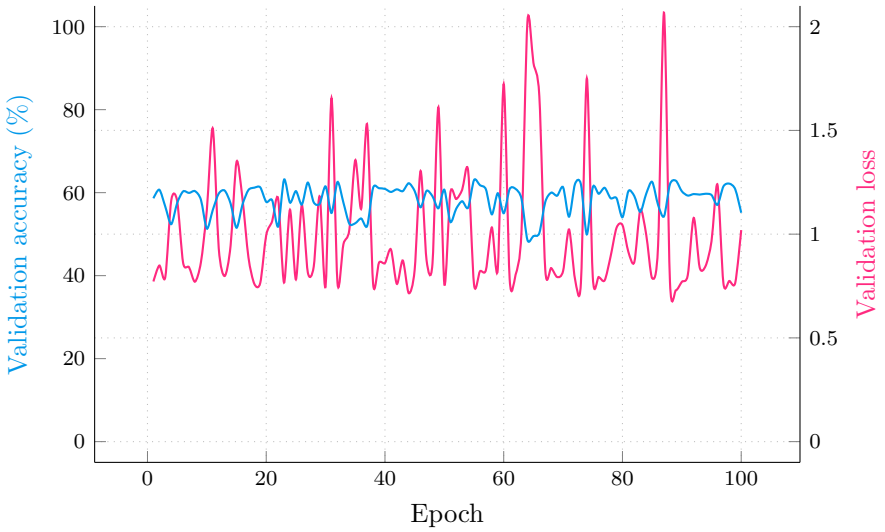

Figure 5: Learning progress of logistic regression on the Flickr data set

8.2 Multi-layer CNN with rectifier activations

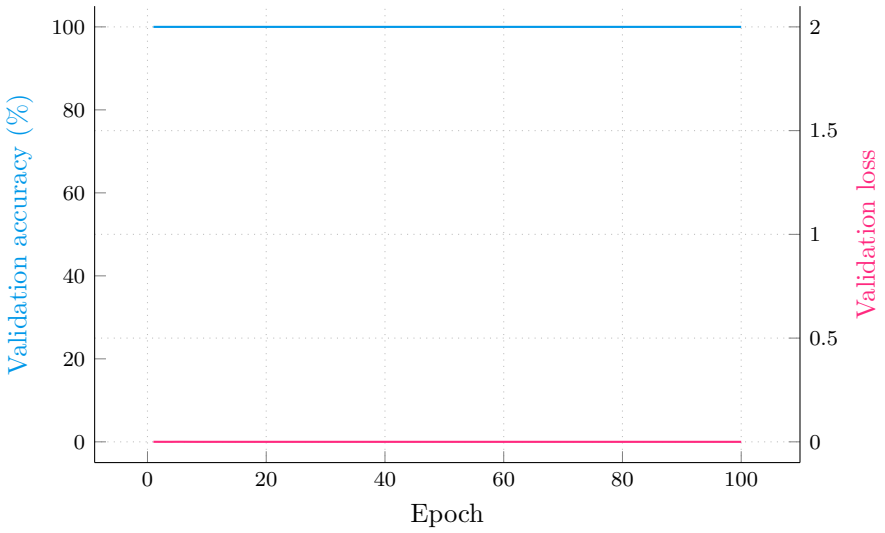

Figure 6: Learning progress of the CNN with rectifier activations on the binary data set

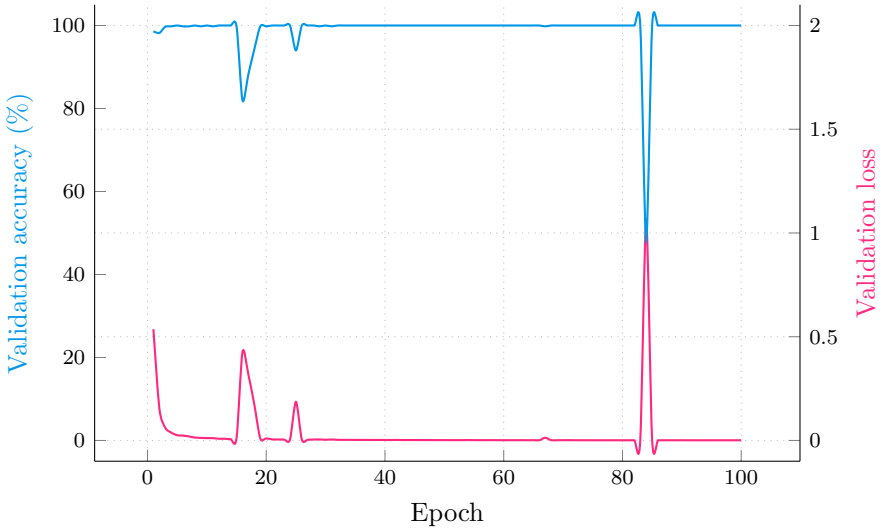

Figure 7: Learning progress of the CNN with rectifier activations on the Solid greyscale data set

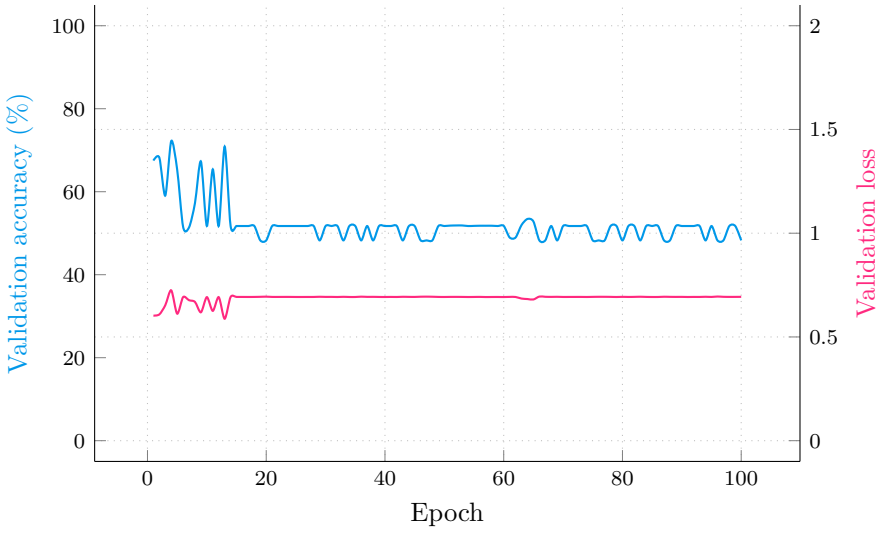

Figure 8: Learning progress of the CNN with rectifier activations on the Flickr data set

8.3 Multi-layer CNN with leaky-rectifier activations

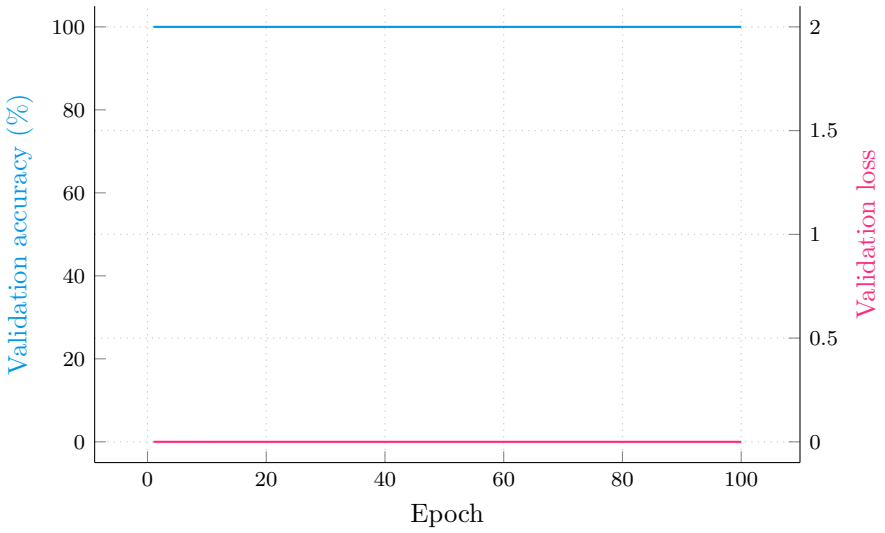

Figure 9: Learning progress of the CNN with leaky rectifier activations on the binary data set

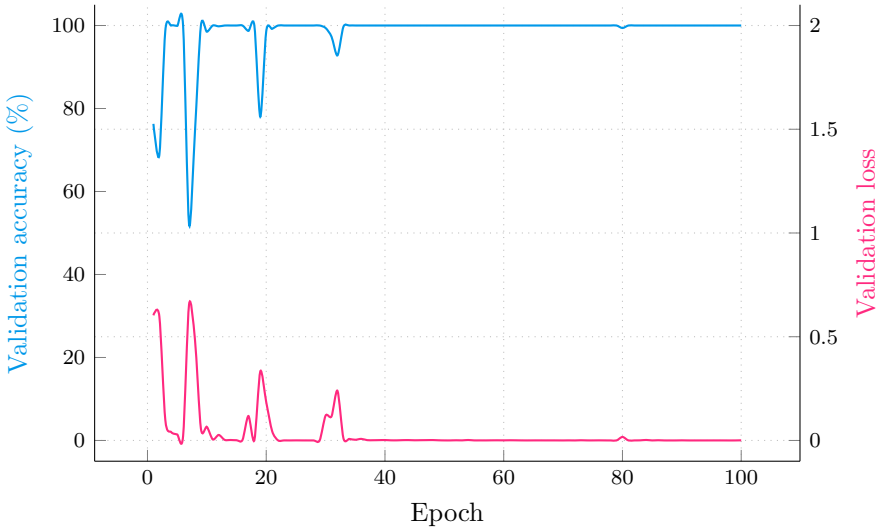

Figure 10: Learning progress of the CNN with leaky rectifier activations on the Solid greyscale data set

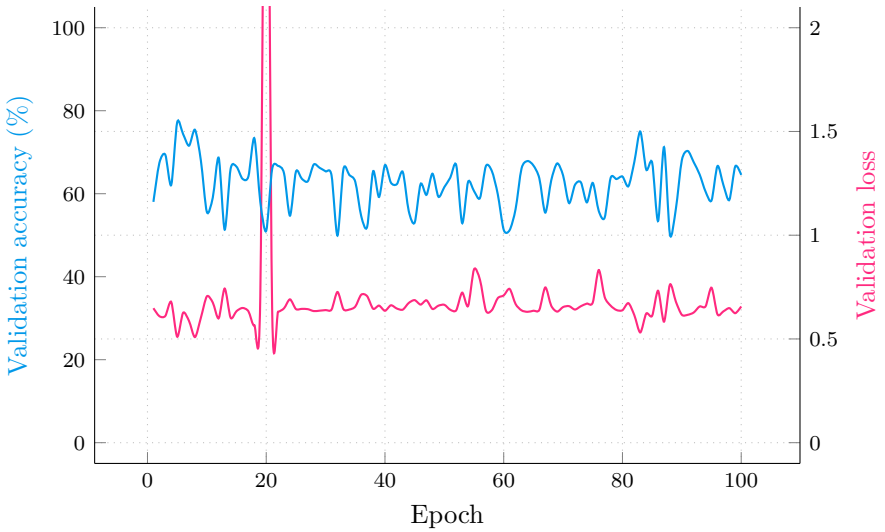

Figure 11: Learning progress of the CNN with leaky rectifier activations on the Flickr data set

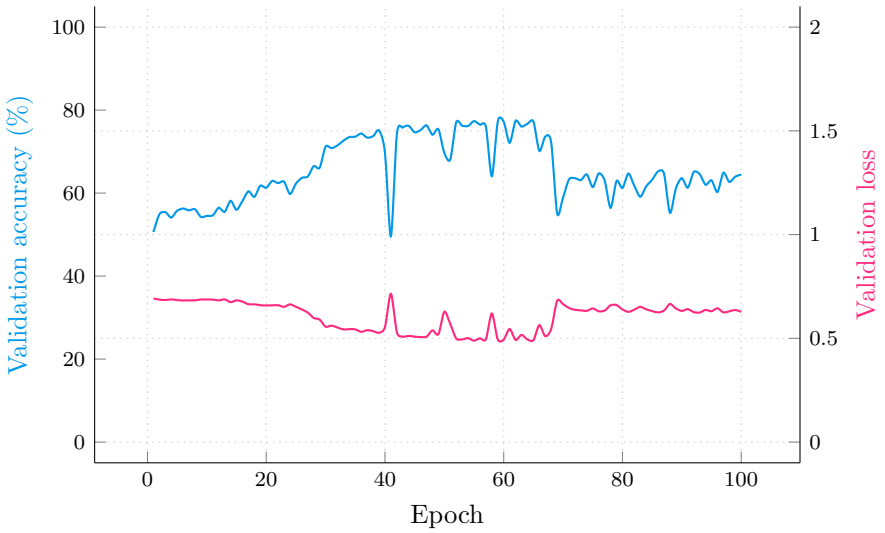

Figure 12: Learning progress of the CNN with leaky rectifier activations on the Target data set