AlexNet-Lite

Erik Putizer, Nicklas Andie and Maximilian Stumpf

Abstract

AlexNet-Lite, a computationally efficient convolutional neural network for digit recognition, is presented. The model reduces parameter counts while maintaining strong classification performance by leveraging batch normalization and dropout regularization. Experiments on the Street View House Numbers (SVHN) dataset demonstrate 85.82% test accuracy, proving effective for resource-constrained image classification tasks.

1 Introduction

In recent years, image classification tasks have become increasingly prominent, driven by advancements in deep learning methods and the availability of large, annotated datasets. Among these datasets, the **Street View House Numbers (SVHN)** dataset provides a compelling benchmark for digit recognition in real-world images. SVHN consists of digitized house numbers extracted from Google Street View imagery, posing challenges such as varying lighting conditions, occlusions, and diverse backgrounds. This paper focuses on leveraging the SVHN dataset to address the task of digit classification, a fundamental problem in computer vision with applications in automated address recognition and other digit-based tasks.
To tackle this problem, we propose **AlexNet-Lite**, a computationally efficient variant of the classical AlexNet architecture. AlexNet-Lite reduces parameter counts and computational demands while maintaining strong classification performance, making it well-suited for datasets like SVHN.

2 Background

A Convolutional Neural Network (CNN) is a special type of deep learning model that works well with classifying and extracting features from images. The formal architecture consists of three main types of layers that the input data propagates through; convolutional layers, pooling layers, and fully connected layers. In the following, we want to describe each of these layers in greater scientific and mathematical detail.[1]

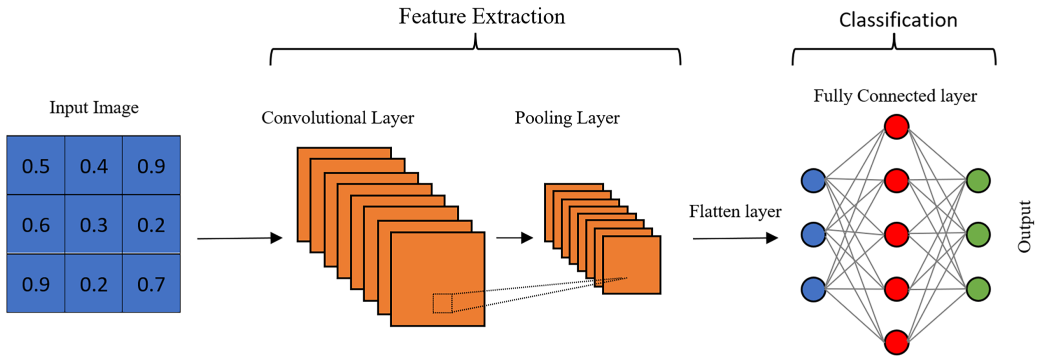


Figure 1: The architecture of a CNN

2.1 Input image

In the CNN architecture, the dimensionality of the input and transformed data plays an important role. As observed in Figure 1, the input data is typically in the form of pixel values from a colored image, which consists of three color channels (red, green, and blue). Although our model is designed to handle RGB images with three color channels, for simplicity of explanation, we assume that the number of color channels is set to 1. This allows us to represent the input as a single matrix instead of a tensor, simplifying the mathematical description of convolution without too much loss of generality.
Let m_1 be the pixel height of the input image and m_2 be the pixel width of the image. Under this assumption, the image can be interpreted as a matrix $I \in \mathbb{M}_{m_1 \times m_2}(\mathbb{N})$, where each element holds a pixel value between 0 and 255, representing the brightness of the corresponding pixel. While the methods described assume a single-channel image for clarity, they are directly extendable to multi-channel RGB images ($m_c = 3$) by considering the convolutional operations across all channels.

2.2 Convolutional layer

The main purpose of the convolutional layer is to apply a collection of learnable filters (or kernels) to the input image. These kernels slide over the input image, generating feature maps that highlight various patterns or features, such as edges, textures, or shapes. This is done with the convolution operation hence the layer's name. Hopefully by choosing fitting kernels, we are able to extract useful features from the images and encode it in a feature map matrix. Typically kernels are 3×3 or 5×5 matrices, but we will consider general kernels with height n_1 and width n_2. Therefore, a general kernel can be interpreted as a matrix $K \in \mathbb{M}_{n_1 \times n_2}(\mathbb{R})$. Mathematically, the feature map, F, is the convolution between the input image and the kernel, that is[2]

$$F[i,j] = (I * K)_{[i,j]} = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} K_{[x,y]} I_{[i-x,j-y]}$$

which result in a matrix $F \in \mathbb{M}_{(m_1-n_1+1)\times(m_2-n_2+1)}(\mathbb{R})$. Note that we can rewrite the convolution because $K_{[x,y]}$ is undefined for certain values, as K only has size $n_1 \times n_2$. Note secondly, that the convolution formula can be parameterized in various ways, but we have chosen to stick with standard literature parametrization. Since focus is mostly on the center of the image during the convolution, we can add columns and rows of zeroes on each side of the input matrix/image, which is called padding. We denote by p the amount of borders around the input before applying the convolution, but set it to 0 here. After the convolution operation a feature map is generated as

$$\underbrace{\begin{array}{|c|c|c|c|c|}\hline 1&0&1&1&0\\\hline 1&0&1&0&0\\\hline 1&0&1&0&1\\\hline 0&1&0&0&1\\\hline 1&0&0&1&1\\\hline\end{array}}_{\textbf{Image}} \quad * \quad \underbrace{\begin{array}{|c|c|c|}\hline 1&0&1\\\hline 0&1&0\\\hline 1&0&1\\\hline\end{array}}_{\textbf{Kernel}} \quad = \quad \underbrace{\begin{array}{|c|c|c|}\hline 4&2&3\\\hline 2&2&2\\\hline 4&1&3\\\hline\end{array}}_{\textbf{Feature map}}$$

Table 1: Illustration of convolution between a 5×5 image and a 3×3 kernel, which results in a 3×3 feature map

shown in table 1. If for some reason we wish to add a bias term to the feature map we are able to do this by adding a matrix $b \in \mathbb{M}_{(m_1-n_1+1)\times(m_2-n_2+1)}(\mathbb{R})$. This results in an activation, upon which we can use an activation function e.g. ReLU, which yields the output from the convolutional layer. To sum up mathematically, we get that $a = F + b = I * K + b$ and the final output from the convolutional layer

$$y = \sigma(a) = \sigma(I * K + b) \in \mathbb{M}_{(m_1-n_1+1)\times(m_2-n_2+1)}(\mathbb{R})$$

An example of an activation function on the feature map from table 1 is seen below in table 2

$$\underbrace{\begin{array}{|c|c|c|}\hline 4&2&3\\\hline 2&2&2\\\hline 4&1&3\\\hline\end{array}}_{\textbf{Feature map}} + \underbrace{\begin{array}{|c|c|c|}\hline -2&1&-2\\\hline -4&2&-4\\\hline 1&-3&2\\\hline\end{array}}_{\textbf{Bias}} = \underbrace{\begin{array}{|c|c|c|}\hline 2&3&1\\\hline -2&4&-2\\\hline 5&-2&5\\\hline\end{array}}_{\textbf{Activation}} \xrightarrow{\text{ReLU}} \underbrace{\begin{array}{|c|c|c|}\hline 2&3&1\\\hline 0&4&0\\\hline 5&0&5\\\hline\end{array}}_{\textbf{Output}}$$

Table 2: Illustration of adding a bias, the resulting activation and the final output

2.3 Pooling layer

The main purpose of the pooling layer is to reduce the dimensions of the output from the convolutional layer while preserving important features. This effectively reduces the computational load and the risk of overfitting. The pooling operation is typically performed by a 2×2 kernel such as a max-pooling kernel (that tries to keep the strongest feature in the region) or an average pooling kernel (with less aggressive down sampling) that strides over the output matrix $y \in \mathbb{M}_{(m_1-n_1+1)\times(m_2-n_2+1)}(\mathbb{R})$. The stride, s, is defined as the pixel step size with which the kernel moves across the input image. Typically a 2×2 kernel uses a stride of 2. The pooling operation results in a matrix P. An example of which is shown below in figure 2

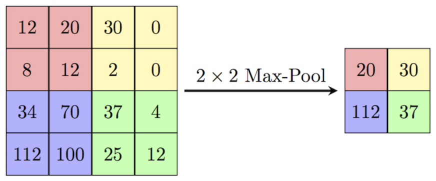


Figure 2: A 2×2 max-pool kernel with a stride of 2 on a 4×4 output from the convolutional layer

2.3.1 Flatten layer

The last part of the feature extraction consists of the flatten layer, which flattens the pooling matrix P from the pooling layer into a vector. This concludes the feature extraction of a simple CNN and the vectorized matrix will then be passed onto the classification layer starting with the fully-connected layer.

2.4 Fully-connected layer

The fully-connected layer functions as a standard feed-forward fully connected neural network. Its primary purpose is to aggregate and process the extracted features from previous layers (such as convolutional or pooling layers) into a compact representation for tasks like classification or regression.

Let x_i denote the i-th coordinate of the vectorized input matrix, and let D represent the dimensionality of the input vector P. The fully-connected layer transforms this input by computing weighted sums followed by a nonlinear activation. Specifically, the values are passed from the input layer to the first hidden layer with M neurons as follows:[3]

$$y_j(P) = f\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right),$$

for $j = 1, \ldots, M$, where $w_{ji}^{(1)}$ are the weights connecting input neuron i to hidden neuron j, $w_{j0}^{(1)}$ is the bias term for neuron j and f is a differentiable nonlinear activation function, such as ReLU, sigmoid, or tanh. This transformation maps the input into a new representation defined by the M neurons of the hidden layer.

The above process is repeated layer by layer, where each subsequent layer applies its own weights and biases to the output of the previous layer. For the l-th layer with N neurons, the output is given by:

$$y_k^{(l)} = f\left(\sum_{j=1}^{M} w_{kj}^{(l)} y_j^{(l-1)} + w_{k0}^{(l)}\right),$$

for $k = 1, \ldots, N$, where $w_{kj}^{(l)}$ are the weights for the l-th layer, $y_j^{(l-1)}$ is the output from the previous layer and $w_{k0}^{(l)}$ is the bias term for the current layer.

The last layer, or output layer, aggregates information and produces the final predictions. For classification problems, the sigmoid or the softmax activation function is typically used to output probabilities across classes.

3 Dataset

This project uses the **Street View House Numbers (SVHN)** dataset[4], a widely-used benchmark for digit recognition in natural scene images. SVHN contains labeled house numbers from Google Street View images, with bounding box annotations for each digit. The training subset includes 73,257 labeled images, which we processed for use in a digit classification task.
The dataset includes bounding box annotations in the `digitStruct.mat` file, detailing the position and size of each digit in an image. During preprocessing, each bounding box was extracted, cropped, resized to 32×32 pixels, and normalized to the $[0, 1]$ range:

$$\mathbf{I}' = \frac{\mathbf{I}_{\text{crop}}}{255},$$

where $\mathbf{I}_{\text{crop}}$ is the cropped digit tensor. Labels were adjusted such that digits labeled as `10` in the dataset were remapped to `0` for classification purposes. Labels were then one-hot encoded to facilitate multi-class classification.
To prepare the data for training, the dataset was split into training (60%), validation (20%), and test (20%) subsets using stratified sampling to preserve class distribution. The processed images were saved in a hierarchical structure:

- `train/`, `val/`, `test/` folders for the splits.
- Within each split, subfolders were created for each digit class (0–9), with images stored as `.png` files.

Images with invalid bounding boxes or unreadable files were automatically excluded during processing. Unlike in other setups, we did not implement additional data augmentation techniques (e.g., random rotations, cropping, or brightness adjustments), as the dataset already provided a sufficient number of images for training and evaluation. The SVHN dataset is inherently diverse, featuring digits captured in real-world scenarios with significant variation in lighting, angles, and backgrounds. This natural variability effectively acts as implicit augmentation, reducing the need for synthetic transformations and allowing the model to generalize well without additional preprocessing steps.

4 Model

4.1 Architecture

Our proposed **AlexNet-Lite** is a computationally efficient variant inspired by the classical AlexNet architecture[5]. It is designed to reduce parameter counts and computational demands while maintaining robust performance on image classification tasks. The model consists of five convolutional layers and three fully connected layers, incorporating batch normalization and dropout for enhanced regularization and improved generalization. Compared to the original AlexNet, AlexNet-Lite includes several architectural adjustments to improve efficiency while retaining key design principles.

The convolutional portion of the model comprises five layers, each using smaller 3×3 filters to reduce computational complexity. The number of filters in each layer was determined using Random Search over a predefined range to identify an optimal balance between model complexity and

performance. The first layer applies 4 filters with a stride of 1 and `same` padding, followed by batch normalization, ReLU activation, dropout with a probability of 0.2, and max pooling with a 2×2 filter and stride of 2. Similarly, the second convolutional layer applies 8 filters with the same configuration of batch normalization, ReLU activation, dropout, and max pooling. The third and fourth convolutional layers each use 16 filters with batch normalization, ReLU activation, and dropout (with $p = 0.3$), but no pooling is applied. The fifth layer uses 8 filters, followed by batch normalization, ReLU activation, dropout, and max pooling with a 2×2 filter and stride of 2. This configuration ensures a progressive reduction in spatial dimensions while maintaining rich feature representations.
The fully connected layers are designed for efficiency and regularization. The first dense layer consists of 64 neurons with ReLU activation and dropout ($p = 0.5$), followed by a second dense layer with 32 neurons and the same activation and dropout configuration. The final layer, which serves as the output, includes 10 neurons corresponding to the number of target classes, with softmax activation for multi-class classification. Mathematically, the transformation through the fully connected layers can be expressed as:

$$\mathbf{y} = \text{softmax}(\mathbf{W}_3(\text{ReLU}(\mathbf{W}_2(\text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)) + \mathbf{b}_3)),$$

where $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{W}_3$ are the weight matrices, and $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{b}_3$ are the biases for the respective layers.

Regularization plays a crucial role in AlexNet-Lite. Dropout is consistently applied after every convolutional and fully connected layer, helping to mitigate overfitting in a dataset like SVHN, which, despite its size, contains some redundancy due to repeated digits in similar contexts. It is defined as:

$$\text{Dropout}(\mathbf{X}, p) = \mathbf{X} \odot \mathbf{M}, \quad \mathbf{M} \sim \text{Bernoulli}(1 - p),$$

where $\mathbf{M}$ is a binary mask sampled from a Bernoulli distribution with probability $1 - p$. Additionally, batch normalization is applied after each convolutional layer, stabilizing training and accelerating convergence by normalizing intermediate feature maps.

Compared to the original AlexNet, AlexNet-Lite dramatically reduces the number of filters and neurons in each layer, adopts smaller kernel sizes, and eliminates computationally intensive operations such as Local Response Normalization (LRN). These changes make AlexNet-Lite more efficient for smaller-scale datasets like SVHN, where the focus is on extracting localized patterns in a computationally lightweight manner. The use of Random Search to optimize the number of filters further enhances the model's efficiency, ensuring it balances performance and computational demands. By leveraging dropout and batch normalization as lightweight and effective regularization techniques, the model achieves strong performance while remaining computationally efficient for real-world digit classification tasks.

4.2 Training Process

4.2.1 Optimizer and Loss Function

The model is trained using the Stochastic Gradient Descent (SGD) optimizer with Nesterov acceleration[6], momentum of 0.9, and a cosine decay learning rate schedule. The momentum term accelerates convergence by dampening oscillations during gradient updates, which are computed as:

$$v_t = \mu v_{t-1} - \eta(t)\nabla\mathcal{L}(\mathbf{W}),$$
$$\mathbf{W} = \mathbf{W} + v_t,$$

where μ is the momentum parameter, v_t is the velocity vector, $\eta(t)$ is the learning rate at time step t, $\mathcal{L}$ is the loss function, and $\nabla\mathcal{L}(\mathbf{W})$ is the gradient of the loss function with respect to the weights. The cosine decay schedule[7] determines the learning rate dynamically as:

$$\eta(t) = \alpha + \frac{1}{2}(\eta_{\text{init}} - \alpha)(1 - \cos(\pi t/T)),$$

where η_{init} is the initial learning rate, T is the total number of decay steps, and α is the final learning rate. In this setup, $\eta_{\text{init}} = 0.01$, $\alpha = 0.0001$, and $T = 10000$. This schedule is chosen with $T = 10000$, slightly larger than the total training steps, to ensure a smoother and gradual decay of the learning rate, allowing for more stable updates and avoiding premature reduction in the early stages of training.

The loss function used is categorical cross-entropy, appropriate for multi-class classification tasks, and is defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log\hat{y}_{i,c},$$

where N is the batch size, C is the number of classes, $y_{i,c}$ is the ground truth label, and $\hat{y}_{i,c}$ is the predicted probability for class c. This loss function penalizes incorrect predictions while emphasizing correct ones, ensuring the model optimizes for accurate classification.

4.2.2 Model Training and Validation

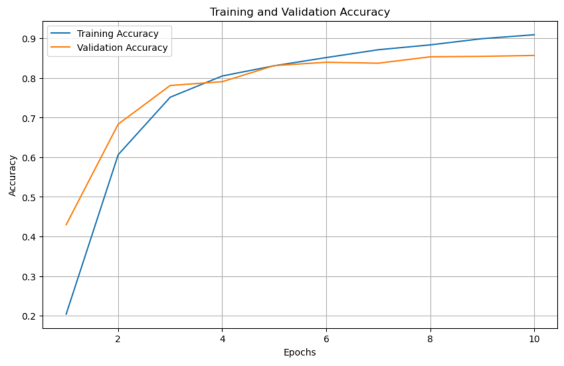
The model is trained for 10 epochs using data generators for efficient batch processing. Separate generators are used for training, validation, and testing datasets, ensuring that no data leakage occurs between the splits. Each generator processes the data in batches of 64 images and uses categorical labels for multi-class classification.

Training is performed using the preprocessed training dataset, and validation is carried out using a separate validation set. The validation set monitors the model's performance and adjust weights during training. The `ModelCheckpoint` callback saves the best model weights based on validation accuracy, ensuring that the best-performing model is retained. To prevent overfitting, an `EarlyStopping` callback monitors validation loss and halts training if no improvement is observed for 3 consecutive epochs, restoring the best weights at the end.

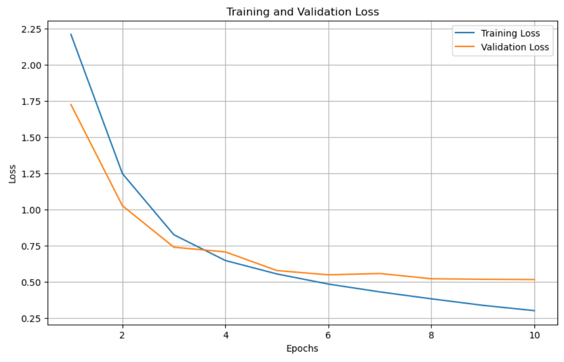
After training, the model's final weights are saved to disk for future use. The model is then evaluated on a completely independent test set to assess its generalization performance. The evaluation uses metrics such as test loss and accuracy to measure how well the model performs on unseen data. The training process, including loss and accuracy values for both the training and validation sets, is stored in the training history for further analysis and visualization.

5 Results

The AlexNet-Lite model was trained on the SVHN dataset over just 10 epochs, achieving a training accuracy of 91.16% and a training loss of 0.2955 by the final epoch. Despite the relatively low number of training epochs, the validation accuracy improved steadily, reaching 85.71%, while the validation loss reduced to 0.5156 3. These results indicate that the model was able to effectively learn meaningful patterns from the training data and generalize well to the validation set. The performance gap between training and validation accuracy suggests slight overfitting, but it remains within an acceptable range. On the independent test set, the model achieved an accuracy of 85.82% and a test loss of 0.4978, demonstrating its ability to generalize effectively to unseen data. The smooth convergence of the training and validation loss curves further supports the stability of the training process. While the model performed strongly given the modest training duration, additional training epochs or techniques like data augmentation and stronger regularization might further enhance its performance. Overall, the AlexNet-Lite architecture demonstrated a compelling balance between computational efficiency and strong predictive accuracy, achieving robust results with minimal training.



(a) Training and Validation Accuracy



(b) Training and Validation Loss

Figure 3: Performance visualizations

6 Conclusion

This project successfully demonstrated the effectiveness of AlexNet-Lite, a lightweight variant of the classic AlexNet architecture, for digit classification on the SVHN dataset. The model achieved strong generalization with minimal training, underscoring its computational efficiency and practical applicability. While some signs of overfitting were observed, this could be addressed with additional regularization or data augmentation. Overall, AlexNet-Lite strikes a balance between performance and efficiency, offering a promising solution for resource-constrained image classification tasks.

References

[1] Ahmad Alsaleh and Cahit Perkgoz. A space and time efficient convolutional neural network for age group estimation from facial images. *PeerJ Computer Science*, 9:e1395, 2023.

[2] Farina Tariq. Breaking down the mathematics behind cnn models: A comprehensive guide, 2023.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[4] Stanford University. Street view house numbers (svhn) dataset, 2024. Accessed: 2024-11-23.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[6] Yurii Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[7] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations (ICLR)*, 2017. Preprint available at arXiv:1608.03983.