RBE549 HW0 - Alohomora

Amrit Krishna Dayanand
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts, 01609
Email: adayanand@wpi.edu

I. Phase 1 - Shake My Boundary

A. Filter Banks

A filter is a basic, discrete computation that can be applied to an image to change its information, by either reducing or increasing its dimensionality, and changing the phase and magnitude of the image data itself. A collection of such filters, or filter-bank, provides valuable information in the form of an N-dimensional filter response. Filter-banks can be applied using convolution or, more generally, the cross-correlation function:

$$Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v]I[i+u, j+v] \tag{1}$$

In the context of Pb-lite, we are interested in filters that can provide low-level information about the image, like detecting changes in texture, brightness and color at different scales and orientations. This section covers three low-level filter-banks: oriented Derivative-of-Gaussian, Leung-Malik, and Gabor.

1) Oriented Derivative-of-Gaussian (DoG) Filter Bank: The derivative of a Gaussian filter forms a simple edge-detector changing from low to high-intensity across one image axis. By rotating this filter, intensity change across an arbitrary axis can be found. Furthermore, applying the filter across different scales results in finer or coarser filtering.

This filter is implemented by convolving a simple Sobel kernel, an approximation of the first-derivative, and a Gaussian kernel. An example of a sobel kernel in the x-axis is shown below.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

A DoG filter bank across 2 scales and 16 scales was generated to contribute to the Pb-lite output.

2) Leung-Malik Filter Bank: The Leung-Malik filter bank is a multi-scale, multi-orientation filter bank consisting of first and second order DoG filters at 6 orientations and 3 scales, 8 Laplacian of Gaussian filters, and 4 Gaussian filters.

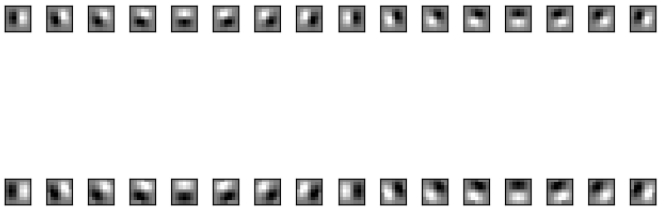

Fig. 1. Derivative-of-Gaussian filter bank over scales [1, $\sqrt{2}$] and 16 orientations

The DoG filters are elongated along the y-axis. Thus its filter response would scrutinize edges better than a regular DoG filter, which has a more circular profile, which would lead to a fuzzier response. The second order DoG filter is distinguished by a low intensity strip straddled by high intensity strips on either side.

The Gaussian filter is a low-pass filter, commonly used for blurring images. This is useful for smoothing out an input image before computing derivatives, which would otherwise amplify noise.

The Laplacian of Gaussian (LoG) filter is commonly used for multiscale blob detection and contour detection. Since the Gaussian is a low-pass filter, and the Laplacian is a high-pass filter, the LoG filter acts as a band-pass filter. An efficient way to implement this is as the difference of two Gaussian filters of slightly different scale. In this paper, the difference in scales occurs at a factor of $\sqrt{2}$.

A small and large version of the Leung-Malik filter were implemented. These occur at the basic scales of $\sigma = [1, \sqrt{2}, 2, 2\sqrt{2}]$, and $\sigma = [\sqrt{2}, 2, 2\sqrt{2}, 4]$, respectively.

3) Gabor Filter Bank: The Gabor filter is inspired by the human visual system. It is implemented by modulating a Gaussian kernel by a 2-D sinusoidal wave. This filter is useful for identifying textures in an image of a particular frequency. Generating a filter bank across scales approximates a change in frequency, and applying a rotation to the filter allows multi-orientation texture detection.

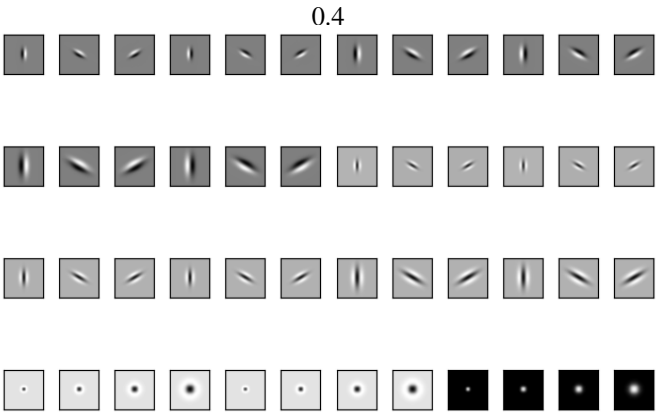

Fig. 2. Leung-Malik "Small" filter bank

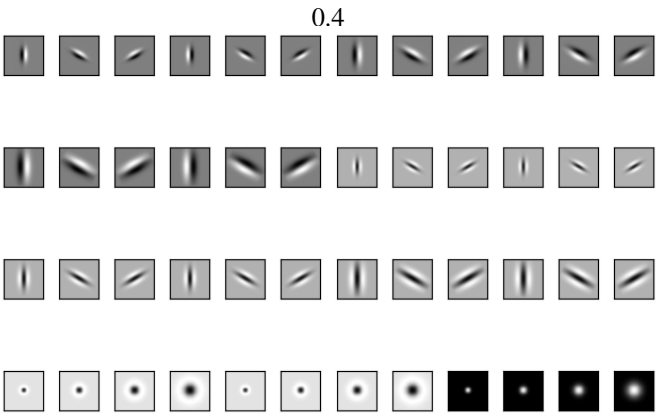

Fig. 3. Leung-Malik "Large" filter bank

Fig. 4. Leung-Malik "Small" and "Large" filter banks, occurring at the basic scales $\sigma = [1, \sqrt{2}, 2, 2\sqrt{2}]$, and $\sigma = [\sqrt{2}, 2, 2\sqrt{2}, 4]$, respectively. The first and second-order derivatives occur at the first three scales with an elongation factor of 3 ($\sigma_x = \sigma, \sigma_y = 3\sigma$)

$$g(x; y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right) \tag{2}$$

$$x' = x\cos\theta + y\sin\theta \tag{3}$$

$$y' = -x\sin\theta + y\cos\theta \tag{4}$$

The Gabor filter is described by the equation above, with wavelength λ, orientation θ, phase offset ψ, scale σ, and spatial aspect ratio (squishing in x or y) γ.

B. Texton Map

Filtering a grayscale input image, I, of dimensions mxn, with N filters results in an N-dimensional filter response of size (N, m, n). We can reduce the dimensionality of this response using K-means clustering (K=64) to quantize the N-dimensional response into 1 dimension, over the same image

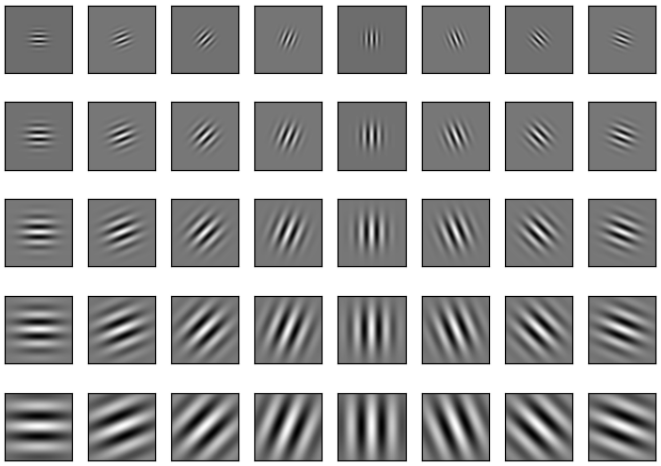

Fig. 5. Derivative-of-Gaussian filter bank over scales [1, $\sqrt{2}$] and 16 orientations

dimensions. This process generates a *Texton Map*, where the image intensity corresponds to the Texton ID (1...K).

C. Brightness Map

The brightness map is the output of applying K-means clustering (K=16) to the raw, grayscale input image. This captures changes in brightness in the image.

D. Color Map

The color map is the output of applying K-means clustering (K=16) to the color image. This captures changes in color in the image. In this implementation, the RGB space was used, but other color spaces can also be used.

E. Computing Map Gradients

The gradient of the texture, brightness and color maps provides insight into how the image changes spatially in each of these aspects, which contributes to better accuracy in detecting textures, edges and salient features in the image.

1) Half-Disk Mask Method: Oriented pairs of half-disks at different scales are used to efficiently calculate the χ^2 distances using filtering.

$$\chi^2(g, h) = \frac{1}{2} \sum_{i=1}^{K} \frac{(g_i - h_i)^2}{g_i + h_i} \tag{5}$$

The χ^2 is a measure used to compare two histograms, g and h with the same number of bins, K.

Performing N half-disk mask convolutions over the texton map, brightness map, and color map, results in the gradient for each map, denoted T_g, B_g, and C_g. In this implementation, 8 orientations at 3 scales were used for the half-disk masks as shown below.

F. Output Images for T_g, B_g, C_g

For each of the ten images, the following were the results of each map gradient.

Fig. 6. Half-disk masks over 3 scales [2.5, 3.5, 6.5] and 8 orientations

Fig. 7. T_g, B_g, C_g of Image 1

G. Sobel and Canny Baseline

The Sobel filter is a simple approximation of a discrete, first-derivative kernel. The Canny filter applies Gaussian smoothing, sobel filtering and non-max suppression (i.e., keeping the strongest gradient magnitude and orientation and setting the rest to 0).

While it is more computationally expensive than a Sobel filter, the Canny filter is less susceptible to noise due to the Gaussian smoothing. The Sobel filter on the other hand will amplify noise to some extent. The Canny and Sobel baselines are shown below for each of the ten images.

H. Pb-lite Results

Pb-lite is computed as the combination of the gradient maps and canny and sobel baselines as follows:

$$PbEdges = \frac{(T_g + B_g + C_g}{3} \odot (w_1 * cannyPb + w_2 * sobelPb) \tag{6}$$

The output of the Sobel, Canny and Pb-lite computation for each of the ten images is shown below.

I. Discussion

From the results above, it is evident that Pb-lite is better at boundary-detection than the Sobel or Canny Baselines due to additional considerations like texture and color.

The Canny baseline is an improvement over that of the Sobel because it incorporates smoothing, which reduces noise like high-frequency, non-boundaries, and non-max suppression, which focuses on the features with the highest magnitude.

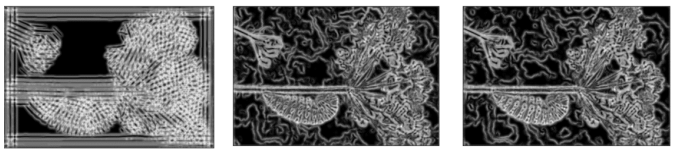

Fig. 8. T_g, B_g, C_g of Image 2

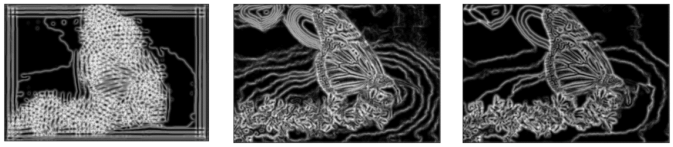

Fig. 9. T_g, B_g, C_g of Image 3

However, colors and texture are a rich source of information, especially in nature. While Canny and Sobel baselines only consider brightness to detect boundaries, Pb-lite incorporates multi-scale and multi-orientation filtering across texture, color and brightness, providing coarse and fine information about the image. Moreover, since Pb-lite uses a linear combination of the Sobel and Canny baselines, the information of the two baselines is augmented with texture, color and brightness features which improves the accuracy of the probability of boundary.

II. Phase II - Deep Dive on Deep Learning

Unlike Phase I, where the focus was on building filters to extract useful features, the focus in Phase II is to develop network architectures that are capable of learning feature-extraction from large-scale data.

A. Simple Convolutional Neural Network

Simple convolutional neural networks (CNN) consist of layers that convolve the input tensor of C_{in} channels with $Cout$ filters. By modulating the stride, dilation and padding, other functions like downsampling the image can be achieved.

My initial implementation was a simple 5-layer network. The first convolutional layer (5x5, 3-¿32) was followed by two convolutional layers (5x5, 32 -¿ 32). Each convolutional layer was followed by a rectified linear unit (ReLU) activation function. The output of the convolution layers was downsampled using maxpooling (2x2) followed by two fully connected layers (size 3200 and 512, respectively). Finally, softmax was applied to generate a probability distribution. I applied a learning rate of 0.001 and the AdamW optimizer.

However, this yielded poor results with increasing loss and poor accuracy over 50 epochs. I increased the learning rate by a factor of 10, upto a maximum rate of 0.1, but this did not help improve performance. This suggested two possible causes. First, that the architecture itself may not be deep enough. Second, that the downsampling of the already small 32x32 image may be causing the network to lose salient

Fig. 10. T_g, B_g, C_g of Image 4

Fig. 11. T_g, B_g, C_g of Image 5

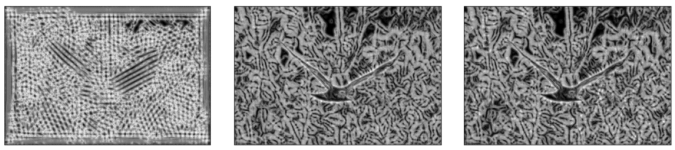

Fig. 12. T_g, B_g, C_g of Image 6

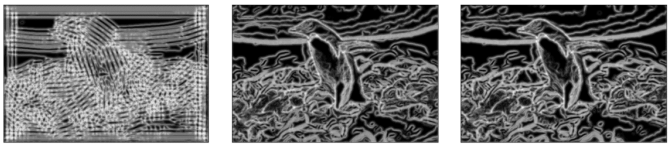

Fig. 13. T_g, B_g, C_g of Image 7

features early.

B. Improved CNN

Standardization, or normalization, can help adjust the dataset such that features contribute more evenly to the learning process. I first tried improving the network by standardizing the data between [-1 and 1], but this alone did not help reduce loss.

Ioffe and Szegedy (2015) showed that adding batch normalization, when applied to mini-batches, instead of the whole dataset, can make training more efficient. When batch normalization is applied after a layer of weights, the output is normalized about the mean of the layer's output and within 1 standard deviation of it. In my implementation, I added batch normalization after each convolution and before the activation function (5x5 conv -¿ BN -¿ ReLU) which led to a significant performance improvement.

C. ResNet

Residual networks aim to mitigate the vanishing gradients problem. As a network's depth increases, the gradient can sometimes reduce to a value close to zero during backpropagation causing learning to stagnate. A residual networks (ResNet) solves this by adding a "skip layer" function that adds the input to the output of a layer (or generally a block of layers).

I implemented a ResNet architecture with a convolutional layer (3x3, 8, S=2), maxpooling (3x3, S=2) followed by three stacks containing a pair of residual blocks, followed by average pooling and a fully connected layer. Each stack had feature-map sizes of 8, 16, and 32, respectively. Each residual block contains two convolutional layers as shown in the figure above followed by an identity skip. When the size of the layer changes, the identity is a 1x1 convolution that acts as a function that maps the input to the size of the output channels.

This yielded a validation accuracy of approximately 68%. I expected this accuracy to be higher, but this is likely because the network was not deep enough at the fully connected layer. I added another fully connected layer which improved the accuracy to 71.88%

D. ResNeXt

Aggregated Residual Networks (ResNeXt) operates similarly to ResNet but instead of direct convolutions, there is an array of parallel (group) convolutions inside each ResNeXt block. This multi-branch structure introduces the group size, also known as cardinality, as a tunable hyperparameter. The authors demonstrate that increasing cardinality is more effective than creating a deeper or wider network, which has important implications for the number of total parameters of the model. In terms of implementation, the concatenation step can be replaced by a group convolution where the number of groups is the cardinality.

ResNeXt is particularly powerful because it increases the density of the network without adding depth. This implies fewer model parameters, but more number of operations to compute the larger group convolution per block.

E. DenseNet

The philosophy behind DenseNet is to connect the output of each DenseNet block to the input of every subsequent block in a feed-forward fashion, along with the output of each block. This architecture propagates features throughout the network, leading to short connections between the input image and each layer of the network. This increases the model's density without increasing the number of parameters.

Fig. 14. T_g, B_g, C_g of Image 8

Fig. 15. T_g, B_g, C_g of Image 9

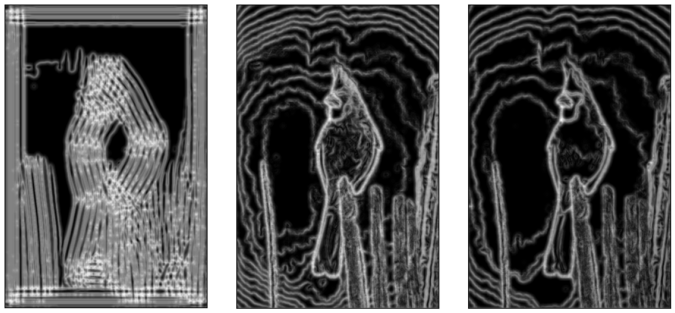

Fig. 16. T_g, B_g, C_g of Image 10

Fig. 17. Sobel, Canny, and Pb-lite output of Image 1

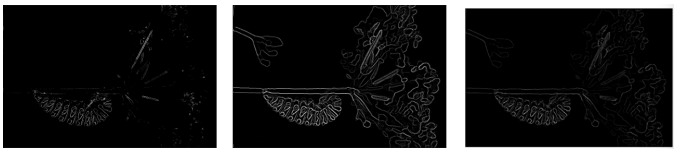

Fig. 18. Sobel, Canny, and Pb-lite output of Image 2

Fig. 19. Sobel, Canny, and Pb-lite output of Image 3

Fig. 20. Sobel, Canny, and Pb-lite output of Image 4

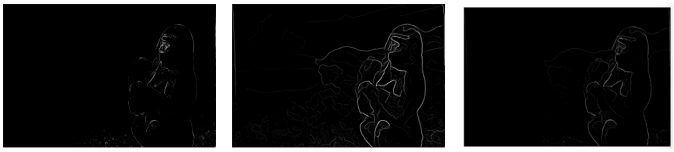

Fig. 21. Sobel, Canny, and Pb-lite output of Image 5

Fig. 22. Sobel, Canny, and Pb-lite output of Image 6

Fig. 23. Sobel, Canny, and Pb-lite output of Image 7

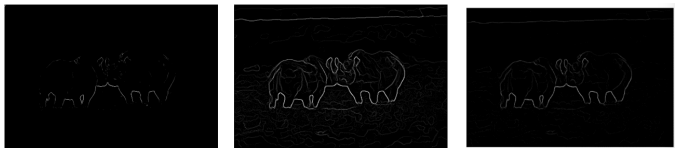

Fig. 24. Sobel, Canny, and Pb-lite output of Image 8

Fig. 25. Sobel, Canny, and Pb-lite output of Image 9

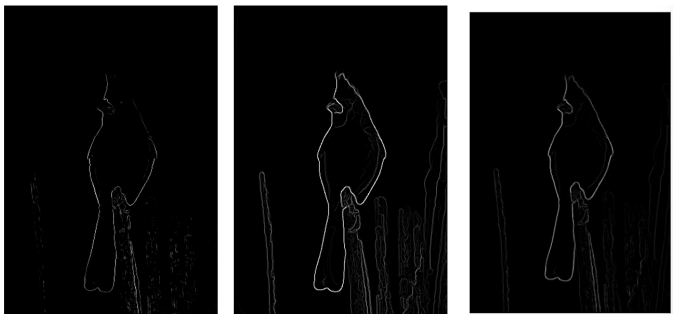

Fig. 26. Sobel, Canny, and Pb-lite output of Image 10

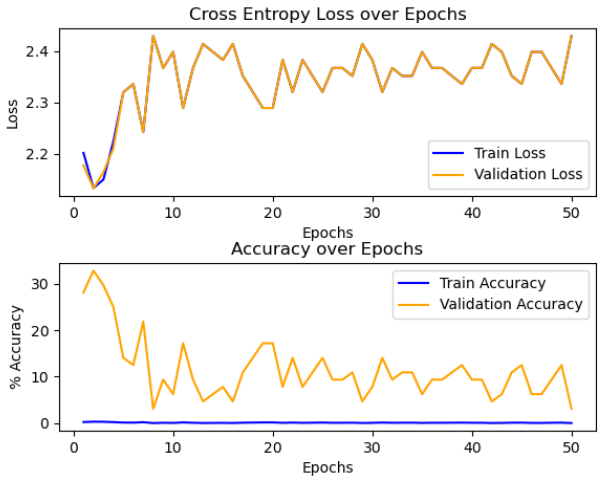

Fig. 27. Performance of the 5-layer simple CNN

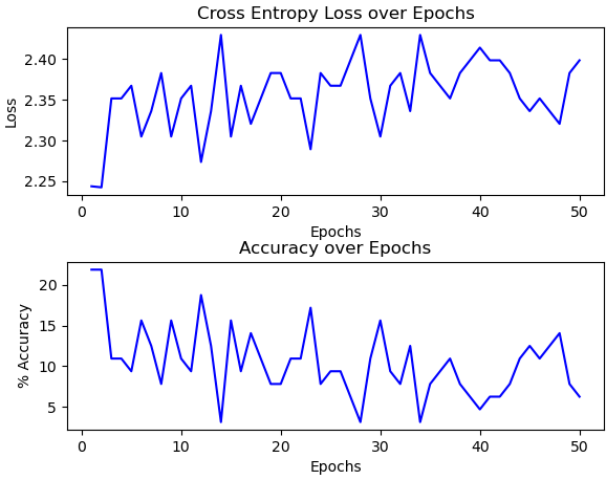

Fig. 28. 5-layer CNN with only Data Standardization

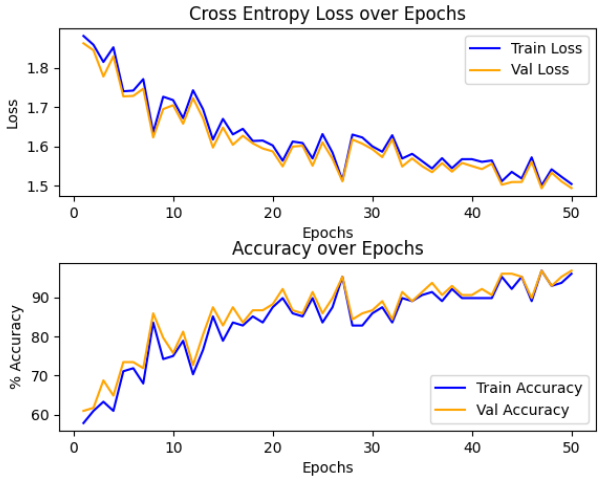

Fig. 29. 5-layer CNN with Batch Normalization and Data Standardization

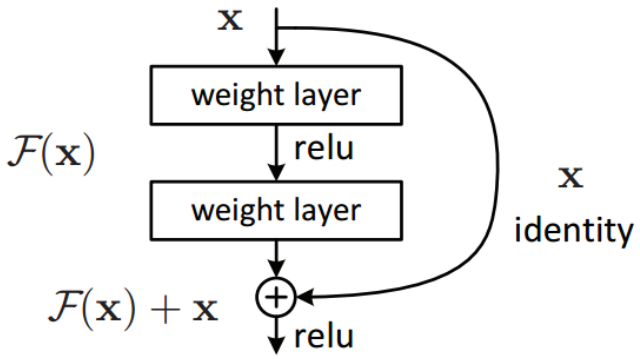

Fig. 30. Basic building block of ResNet

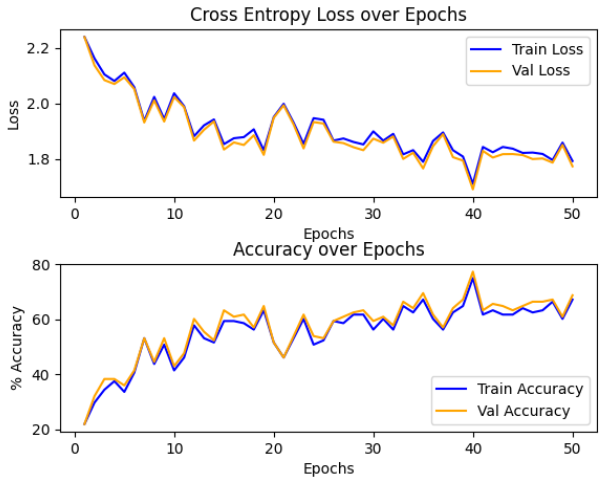

Fig. 31. Performance of ResNet with 1-fc layer

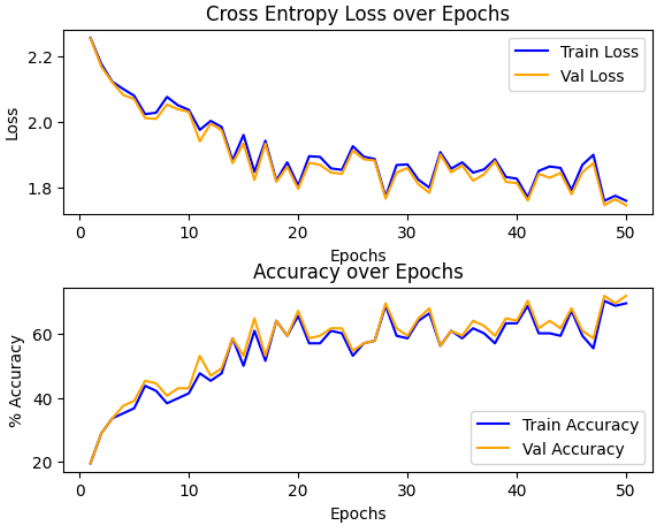

Fig. 32. Performance of ResNet with 2-fc layer

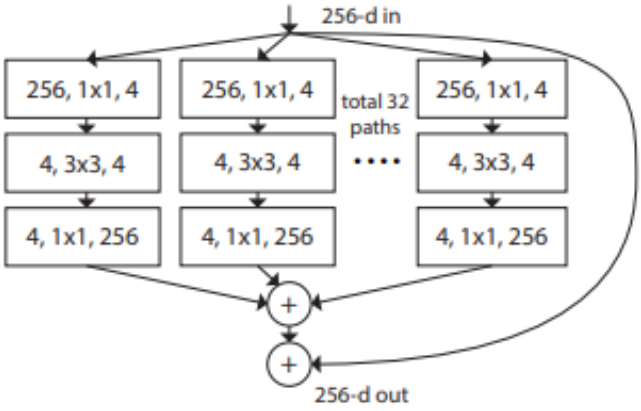

Fig. 33. ResNeXt basic building block