Computational Neuroscience

Project No. 5

Name: Mahroo Hajimehdi SID: 610399198

Introduction

The primary objective of this project is to explore and implement a convolutional spiking neural network (CSNN) to efficiently extract image features from a given dataset. Spiking neural networks (SNNs) have the potential to provide significant advantages over traditional artificial neural networks (ANNs) in terms of computational efficiency and biological plausibility.

Understanding how images are processed in the initial regions of the visual cortex is crucial for designing effective neural networks that mimic biological vision systems. The visual cortex is the part of the brain responsible for processing visual information. In this project, particular attention is given to how the initial visual cortex (specifically areas such as V1) handles and interprets visual stimuli. In the initial visual cortex, the processing involves complex operations to detect edges, orientations, and basic shapes. The purpose is to build a foundational understanding that can be translated into designing spiking neural networks capable of emulating these biological processes. By understanding how visual information is processed naturally, more efficient and accurate neural network models can be developed, leveraging the strengths of biological systems for computational purposes.

In this project, we will explore the details of CSNNs, understand their structure, function, and performance in the context of image feature extraction. The focus is on exploring various techniques for interlayer communication within a CSNN. This includes studying how spikes are transmitted across layers, how temporal dynamics and synaptic weights influence the transfer of information, and the methods to maintain the temporal fidelity of the spiking signals.

Task 1: Applying The Filters

Gabor Filter For Grayscale Images

Gabor filters are widely used in image processing and computer vision for texture analysis, edge detection, and feature extraction. They are particularly effective in simulating the response of the human visual cortex. When used in the context of Spiking Neural Networks (SNNs), Gabor filters help in preprocessing the input image to extract salient features that can be encoded as spikes.

A Gabor filter is a linear filter whose impulse response is defined by a harmonic function multiplied by a Gaussian function. The two-dimensional Gabor filter $G(x, y; \lambda, \theta, \psi, \sigma, \gamma)$ is given by:

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

where λ is wavelength of the sinusoidal factor, θ is orientation of the normal to the parallel stripes, ψ is the phase offset σ is standard deviation of the Gaussian envelope, and γ is spatial aspect ratio (ellipticity of the Gabor function)

In Spiking Neural Networks (SNNs), the preprocessed image $I_G(x, y)$ is encoded into spikes. This can be achieved using various encoding schemes, such as rate coding, temporal coding, or rank-order coding. The Gabor-filtered image serves as an intermediate representation that emphasizes edges and textures, which are crucial for the subsequent layers of the SNN to process. In the following sections, we will first analyze the effect of each parameter on the filter's representation. Then, we will proceed with Time To First Spike (TTFS) encoding and Poisson encoding to convert the convolved image into spikes, preparing it for input into a Spiking Neural Network (SNN).

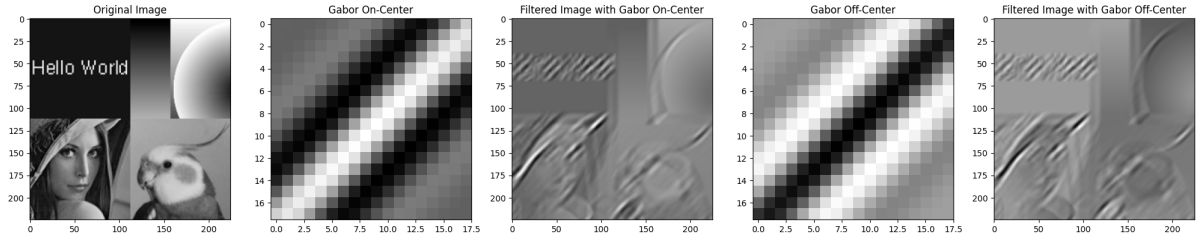

Figure 1: Image convolved with Gabor filter On and Off-Center ($Size = 17, \lambda = 8, \sigma = 4, \gamma = 0.4, \theta = \frac{\pi}{4}$)

Parameter Analysis

We selected a montage image to represent both broad and detailed features, enabling us to analyze the effects of various parameters. The image, along with an example of a Gabor filter convolved with it, is shown in Figure 1. Gabor filters are widely used in texture analysis due to their ability to provide a multi-resolution, multi-orientation analysis of the image. Selecting the right combination of $\lambda, \sigma, \gamma, \theta$, and filter size depends on the specific features we aim to capture in our image. Smaller values are generally better for fine details, while larger values are suited for broader patterns and contexts.

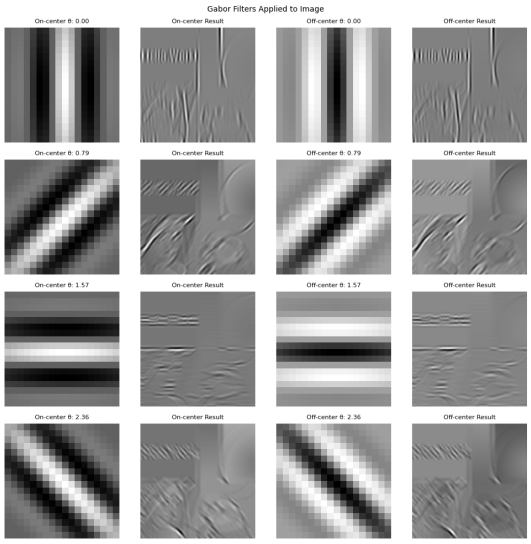

Figure 2: Analysis of Gabor filter θ for values $0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}$ ($Size = 17, \lambda = 8, \sigma = 4, \gamma = 0.4$)

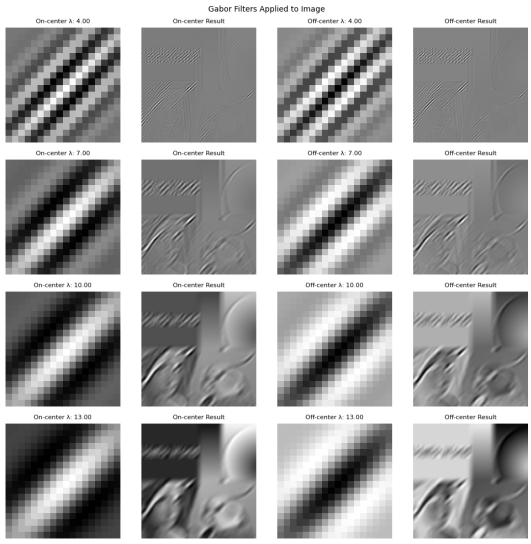

Figure 3: Analysis of Gabor filter λ for values $4, 7, 10, 13$ ($Size = 17, \sigma = 4, \gamma = 0.4, \theta = \frac{\pi}{4}$)

The parameter θ in a Gabor filter controls the orientation of the filter. This determines the direction in which the filter will be sensitive to edges and patterns. This is illustrated in Figure 2. The horizontal lines are distinctly highlighted by the third filter, while the vertical lines are prominently displayed by the first filter.

The effect of parameter λ is shown in Figure 3. In the first row ($\lambda = 4.0$), the on-center and off-center Gabor filters produce higher frequency results. With the wavelength increasing through

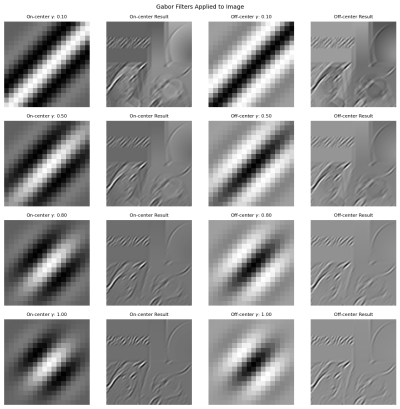

Figure 4: Analysis of Gabor filter γ for values $0.1, 0.5, 0.8, 1.0$ ($Size = 17, \lambda = 8, \sigma = 4, \theta = \frac{\pi}{4}$)

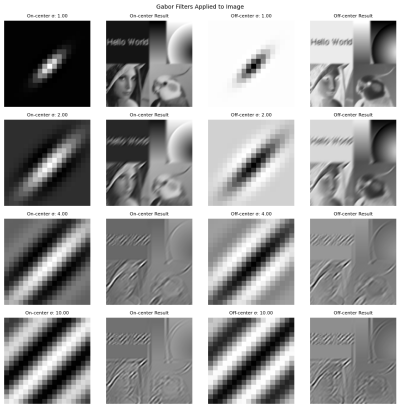

Figure 5: Analysis of Gabor filter σ for values $1, 2, 4, 10$ ($Size = 17, \lambda = 8, \gamma = 0.4, \theta = \frac{\pi}{4}$)

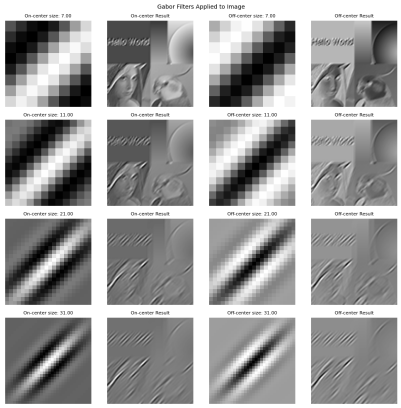

Figure 6: Analysis of Gabor filter $Size$ for values $7, 11, 21, 31$ ($\sigma = 4, \lambda = 8, \gamma = 0.4, \theta = \frac{\pi}{4}$)

rows two, three and four, the filter aims to capture feature with lower frequencies. This results in a clearer image with discernible features and smooth transitions between them.

We can conclude that large λ values, capture low-frequency information, smoothing out very fine details. Due to larger receptive field, the captured details is larger-scale patterns and contrasts. In the meantime, smaller λ responds to fine details and edges in the image and the receptive field is smaller making it focus on smaller regions.

As it is clearly seen in Figure 4, parameter γ controls the ellipticity of the Gabor filter. Smaller values produces an elongated, elliptical filter and emphasize specific orientations and directional features. Larger values for γ produce a circular filter. Therefore, it's equaly sensitivity to all orientations as the last row suggests.

The parameter σ determines the width of the Gaussian envelope and must be set carefully to ensure that the filter captures sufficient surrounding context. Smaller σ values produce a narrower Gaussian envelope, as illustrated in the first row of Figure 5 with $\sigma = 1.0$, where the filter captures very fine details of the 'Hello World' text. Such a narrow envelope focuses on a small region around the filter's center, risking loss of information if the filter size is too large. Conversely, larger σ values result in a wider Gaussian envelope, capturing broader patterns and coarser details. This provides more context but reduces sensitivity to fine details.

The effect of the filter's shape is depicted in Figure 6. When the filter size is too large relative to the image size, as seen in the last row, it captures broader, low-frequency information, which smooths out and blurs fine details while emphasizing larger patterns and features in the image. Consequently, very little context is seen in the last row due to the overly large filter size. Conversely, smaller filters, as shown in the first row, focus on very local features and are highly sensitive to details and edges.

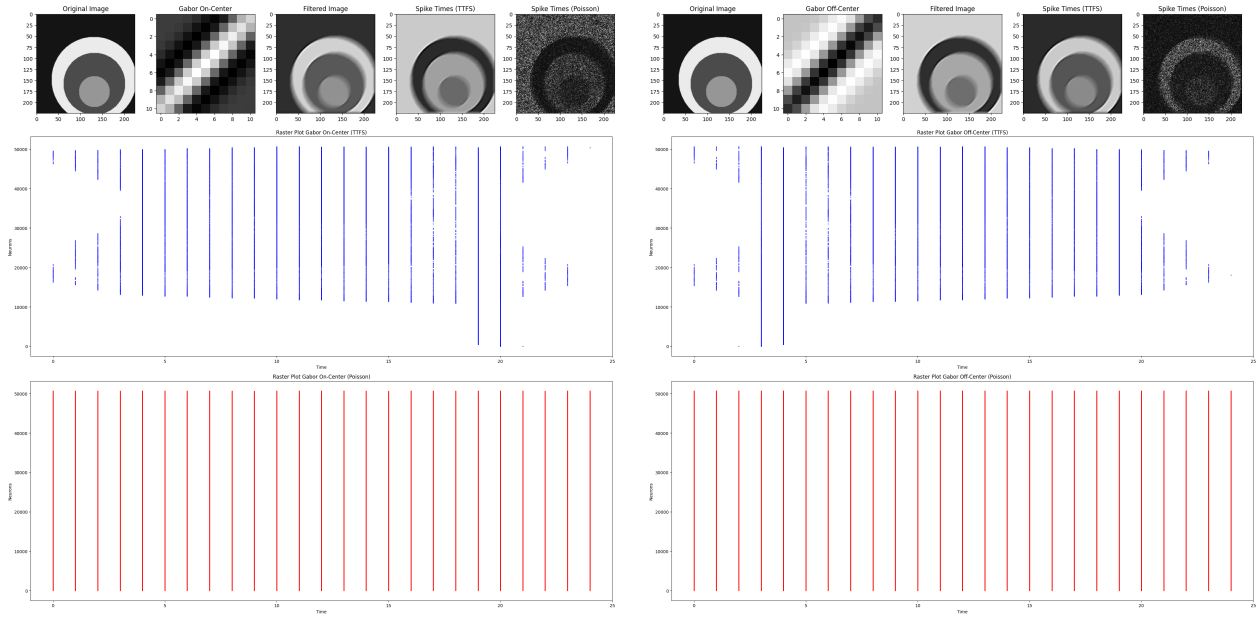

Figure 7: Encoding for an image convolved with On-Center (left column) and Off-Center (right column) Gabor filters. Blue Raster plots show TTFS encodings and red raster plots show Poisson encodings.

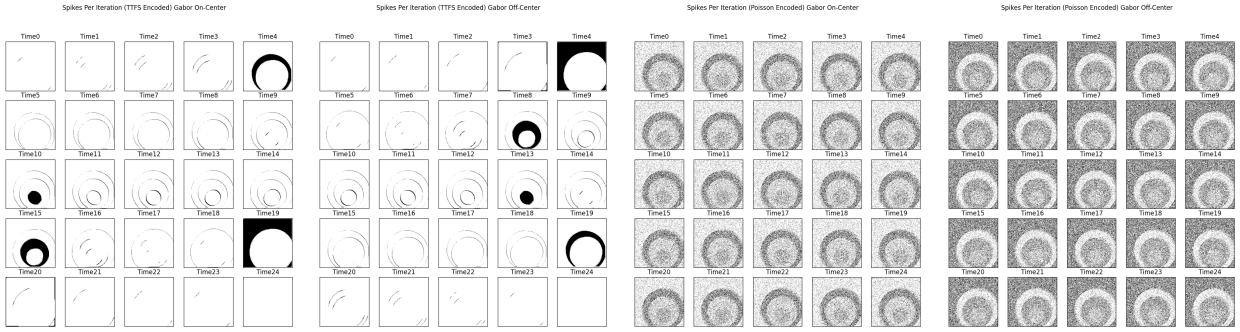

Figure 8: Spike times per TTFS (columns 1 and 2) and Poisson (column 3 and 4) encodings' time window for On-Center (columns 1 and 3) and Off-Center (columns 2 and 4) Gabor filters.

Encoding

We implemented two encoding methods, namely TTFS and Poisson encoding to convert the images to spike times. **TTFS** encoding is a method where the information is encoded based on the timing of the first spike (or firing) of a neuron in response to a stimulus. Each pixel's intensity in the Gabor-filtered image is mapped to a spike time. Higher intensities result in earlier spikes. TTFS provides precise timing information about the presence and strength of features. **Poisson** encoding represents information by generating spikes at a rate proportional to the pixel intensity. Spikes are generated stochastically according to a Poisson process. Each pixel's intensity in the Gabor-filtered image determines the firing rate of a neuron. Higher intensities lead to higher firing rates.

Figures 7 and 8 confirm the above explanations. There is a clear difference between images generated with TTFS and Poisson (7 top row). We expect TTFS to results in fewer spikes, as

each neuron fires once (Figure 7 blue raster plots) while Poisson should results in more spikes, as neurons fire repeatedly according to a Poisson process (Figure 7 red raster plots).

Also, we can clearly see that TTFS encodes information in the precise timing of the first spike while Poisson encodes information in the rate of spikes over time (8). Therefore, TTFS provides high temporal precision, as the exact timing of the first spike carries the information while, Poisson provides less temporal precision, as the information is spread over multiple spikes.

On viewing Figure 8, we can see that On-Center Gabor filters enhance features corresponding to bright regions on dark backgrounds regardless of encoding method, which is expected. On the other hand Off-Center Gabor filters enhance features corresponding to dark regions on bright backgrounds. With TTFS the spike trains are more deterministic, with spikes occurring at precise moments so we can observed high temporal precision as each spike represents the timing of a prominent feature detected by the Gabor filters. With Poisson encoding, temporal information is less precise, but the overall firing rate represents the intensity of the features detected by the Gabor filters.

DoG Filter For Grayscale Images

The Difference of Gaussian (DoG) filter is an image processing technique used primarily for edge detection. It is created by subtracting one Gaussian blurred version of an image from another, less blurred version of the same image. This subtraction emphasizes regions of the image where there are rapid intensity changes, which correspond to edges.

To implement it, we first apply two Gaussian filters with different standard deviations (σ_1 and σ_2) to the image. The Gaussian filter $G_\sigma(x, y)$ is defined as:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{1}$$

To get Difference of Gaussians (DoG) we need to subtract the result of the second Gaussian filter from the first Gaussian filter: $\mathrm{DoG}_{\sigma_1,\sigma_2}(x, y) = G_{\sigma_1}(x, y) - G_{\sigma_2}(x, y)$. To apply it to our image we need to convolve the grayscale image $I(x, y)$ with the DoG filter:

$$I_{\mathrm{DoG}}(x, y) = I(x, y) \star \mathrm{DoG}_{\sigma_1,\sigma_2}(x, y) \tag{2}$$

where $\star$ denotes the convolution operation.

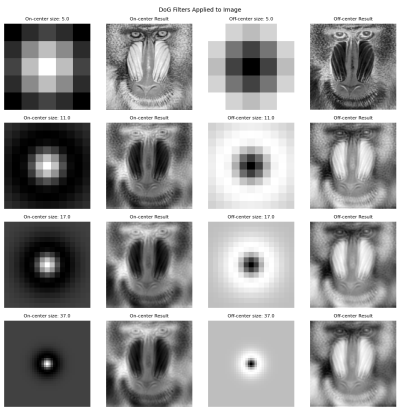

Figure 9: Analysis of DoG filter $Size$ for values 5, 11, 17, 37 ($\sigma_s = 3.5, \sigma_c = 1.5$)

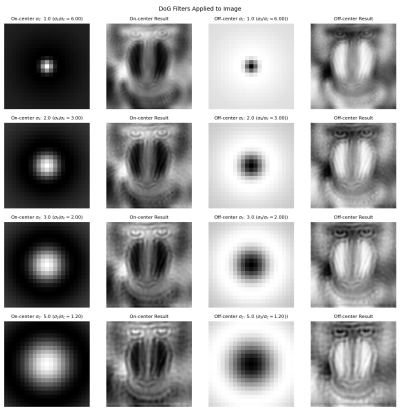

Figure 10: Analysis of DoG filter σ_c for values 1, 2, 3, 4 ($Size = 21, \sigma_s = 6$)

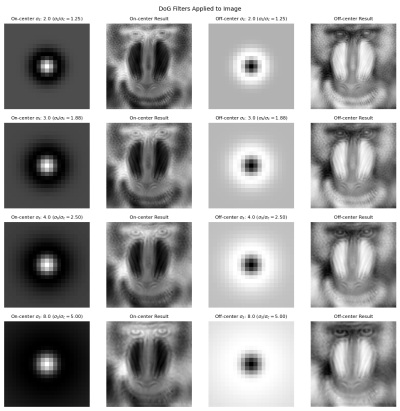

Figure 11: Analysis of DoG filter σ_s for values 2, 3, 4, 8 ($Size = 21, \sigma_c = 1.6$)

Parameter Analysis

As it can be observed in Figure 9, smaller kernel size (5) focus local, fine-grained details while larger kernel size (37) allow the filter to capture larger-scale features and broader context in the image. The problem with small kernels is that they may truncate the Gaussian distributions, potentially leading to less accurate filtering, especially for larger σ values. Overall, the kernel size should be large enough to represent the full extent of the Gaussian (determined by σ_s).

Smaller σ_c creates a narrower, more peaked center in the DoG kernel while larger σ_c produces a wider, flatter center in the kernel (10). Smaller values for this parameter lead to detection of finer details and sharper edges. We can also observe that smaller σ_s/σ_c ratio has detected finer details but has increased noise significantly.

Larger σ_s creates a wider inhibitory surround in the DoG kernel (11). The difference between σ_s and σ_c determines the width of the positive center region in the kernel. We can also see that larger σ_s/σ_c detects broader features. On viewing Figures 10 and 11, we can conclude that generally smaller σ values enhance finer details but may amplify noise while larger values may blur small details of the image too much.

Encodings

Same as the previous section, we can see a clear difference between images generated with TTFS and Poisson (12 top row). TTFS has resulted in fewer spikes (Figure 12 blue raster plots) while Poisson encoding resulted in more spikes (Figure 12 red raster plots). Again, we observe that TTFS encodes information in the precise timing of the first spike while Poisson encodes information in the rate of spikes over time (13).

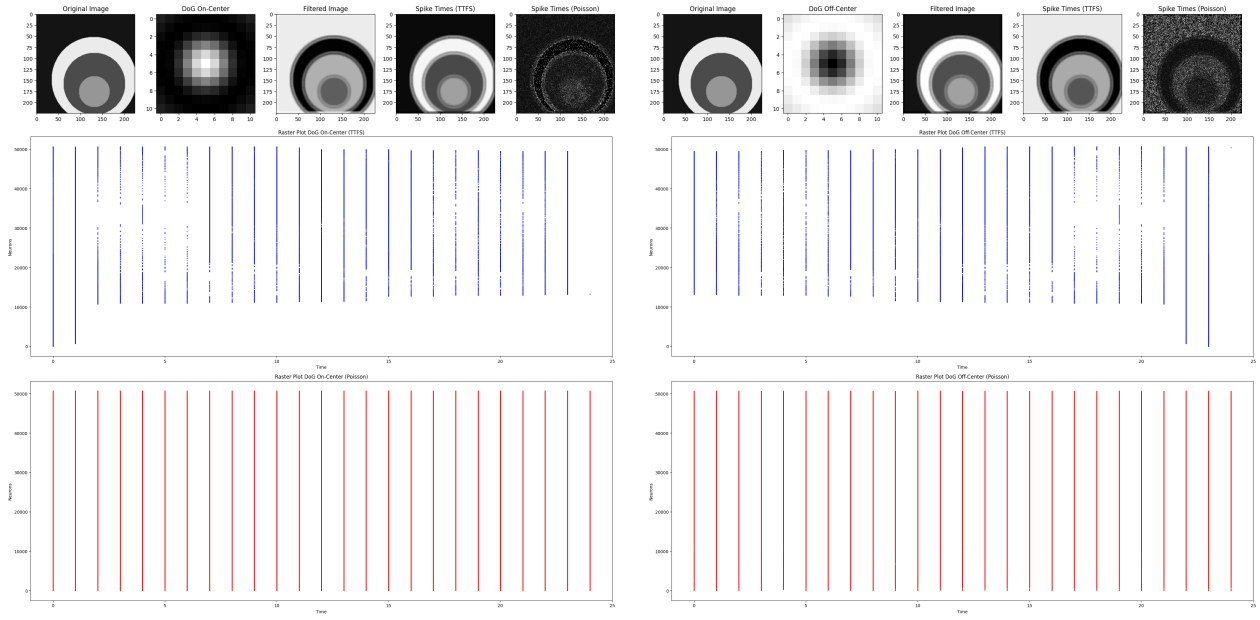

Figure 12: Encoding for an image convolved with On-Center (left column) and Off-Center (right column) DoG filters. Blue Raster plots show TTFS encodings and red raster plots show Poisson encodings.

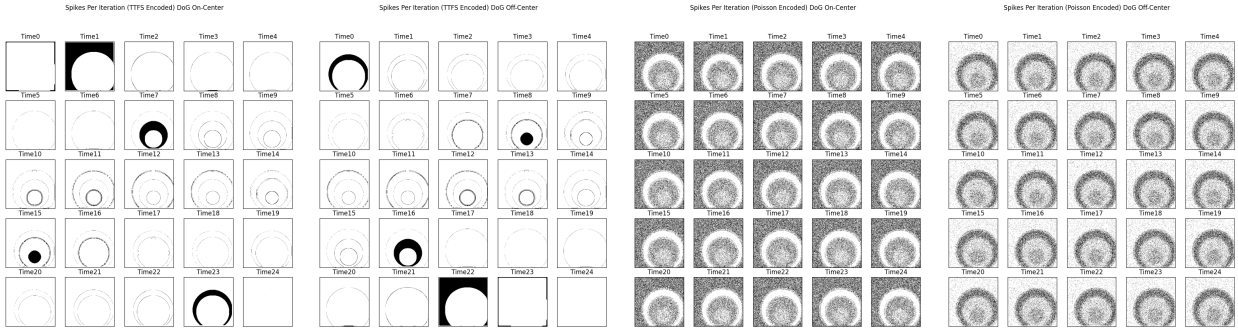

Figure 13: Spike times per TTFS (columns 1 and 2) and Poisson (column 3 and 4) encodings' time window for On-Center (columns 1 and 3) and Off-Center (columns 2 and 4) DoG filters.

On comparing Figures 8 and 13, we notice that Gabor filters seems to break the circles into oriented segments while DoG filters preserve the complete circular structure. This is also more pronounced with TTFS encodings rather than Poisson encodings. Here's a refined version of the text:

The difference in behavior is rooted in the fundamental characteristics of Gabor and DoG filters. Gabor filters are directional, responding strongly to edges or textures aligned with specific orientations. In contrast, DoG filters are rotationally symmetric, reacting to intensity changes regardless of direction. When applied to circular shapes, Gabor filters exhibit a selective response, highlighting only the portions of the circle's edge that align with the filter's orientation. This results in a fragmented representation of the circular contour.

DoG filters, on the other hand, operate by subtracting a more blurred version of the image

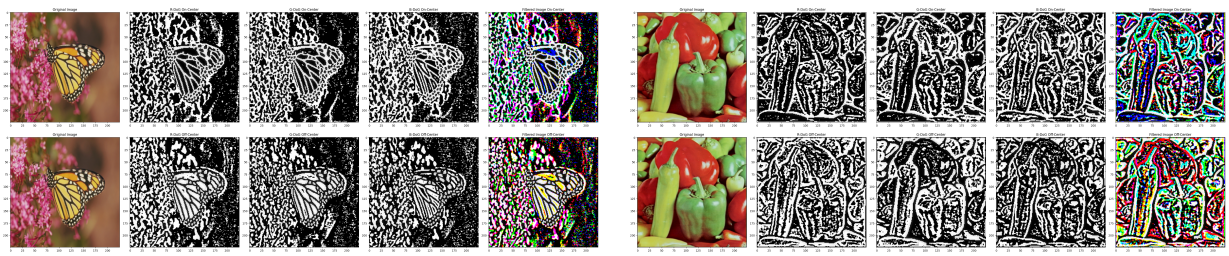

Figure 14: RGB DoG filters. Top figures shows R, G and B On-Center filters along with the original and final images. Bottom figures shows R, G and B Off-Center filters along with the original and final images.

from a less blurred one. This process enhances edges uniformly across all orientations, making DoG filters particularly effective at detecting closed contours such as circles. The result is a more complete and accurate representation of circular shapes.

This distinction underscores why DoG filters are often preferred for tasks involving the detection of closed contours or circular objects, while Gabor filters excel in applications requiring the detection of oriented features or textures.

DoG Filter for RGB Image

For an RGB image, the DoG filter is applied to each channel separately. We first need to apply two Gaussian filters with different standard deviations (σ_1 and σ_2) to each channel. The Gaussian filter $G_\sigma(x, y)$ is as equation 1. Again, we substract the result of the second Gaussian filter from the first Gaussian filter same to get $\text{DoG}_{\sigma_1,\sigma_2}(x, y)$. Finally, we have to convolve each **channel** of the RGB image $I_c(x, y)$ (where c denotes the color channel: Red, Green, or Blue) with the DoG filter:

$$I_{c,\text{DoG}}(x, y) = I_c(x, y) \star \text{DoG}_{\sigma_1,\sigma_2}(x, y) \tag{3}$$

where $\star$ denotes the convolution operation.

Task 2: Training An Convolutional SNN

In this task, we utilized the Difference of Gaussians (DoG) filters tested previously to train a spiking neural network on the MNIST dataset. The network's first layer (Input Layer) employs DoG filters to detect contrasts in the input images, encoding the strength of these contrasts into the latencies of its output spikes (i.e., higher contrast results in shorter latency). The values of $size = 5$, $sigma_c = 1$ and $sigma_s = 2$ were chosen for DoG filter.

The convolutional layers aim to detect more complex features by integrating the simpler visual features identified by the previous layer. Convolutional neurons emit spikes when they detect their preferred visual feature, which is determined by their synaptic weights. During the learning process, neurons that fire earlier undergo Spike-Timing-Dependent Plasticity (STDP) and inhibit the others through a winner-take-all mechanism. Consequently, the network tends to learn the more salient and frequent features from the input data.

Input Layer

It is important to note that we include a silent interval before processing each subsequent image to allow the spike traces and potentials of neurons to decay, ensuring they are ready for the next image. Additionally, learning occurs exclusively in the convolutional layer.

The input is preprocessed and converted into spikes beforehand. The first Neuron Group (first layer) spikes based on this preprocessed input, using the Intensity2Latency function from the CoNex library. By using the sparsity parameter instead of a fixed threshold, we can automatically adapt to different input images. Setting the sparsity to 0.2 means that only the top 20% of the brightest pixels will spike, making the input spikes sparse but more informative.

Convolutional Layer

A convolutional layer contains several neuronal maps, with each neuron being selective to a particular visual feature determined by its input synaptic weights. Neurons in a specific map detect the same visual feature but at different locations within the input. Therefore, synaptic weights of neurons within the same map are always identical, a concept known as weight sharing. Each neuron receives input spikes from neurons within a defined window in all neuronal maps of the previous (input) layer. We used 7 feature maps each of shape 5×5 in the second layer.

All neurons in the convolutional layers are non-leaky integrate-and-fire (LIF) neurons. These neurons accumulate input spikes from presynaptic neurons and emit a spike when their internal potential reaches a predetermined threshold. Each presynaptic spike increases the neuron's potential by its corresponding synaptic weight. Additionally, there is a lateral inhibition mechanism in the convolutional layer: when a neuron fires at a specific location, it inhibits other neurons around it.

It is important to note that selecting large values for the learning parameters (i.e., a^+ and a^-) will decrease the learning memory. Consequently, neurons would learn the most recently presented images while unlearning previously seen images. Conversely, choosing very small values for these parameters would slow down the learning process.

At the start of the learning phase, when synaptic weights are random, neurons are not yet selective for any specific patterns and thus respond to a wide variety of patterns. This makes the probability of synaptic depression higher than that of potentiation. Therefore, setting a^- to be greater than a^+ will gradually cause the synaptic weights to decay until neurons can no longer reach their firing threshold. To avoid this, it is advisable that a^+ be slightly greater than a^-. However, setting a^+ much higher than a^- can cause neurons to learn multiple patterns and respond to all of them. It is best to choose values for a^+ and a^- that are neither too large nor too small, with a^+ being slightly greater than a^-. We chose $a^+ = 0.2$ and $a^- = 0.15$ for our convolutional STDP.

During the learning process of a convolutional layer, neurons within the same map, which detect the same feature at different locations, integrate input spikes and compete with each other to perform Spike-Timing-Dependent Plasticity (STDP) using the k-Winner-Take-All (KWTA) mechanism. The first neuron to reach the threshold and fire is designated as the winner (global intra-map competition). The winning neuron triggers STDP and updates its synaptic weights. As previously mentioned, neurons at different locations within the same map share the same input synaptic weights (weight sharing) to ensure they are selective to the same feature. Consequently, the winning neuron prevents other neurons within its map from performing STDP and replicates its updated synaptic weights to them.

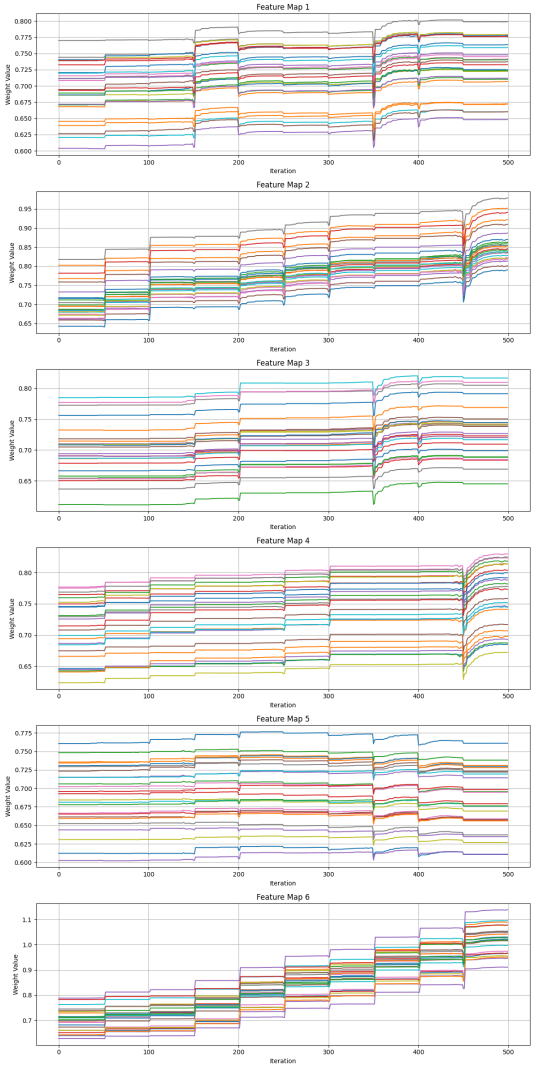

Figure 15: Weight change during iterations.

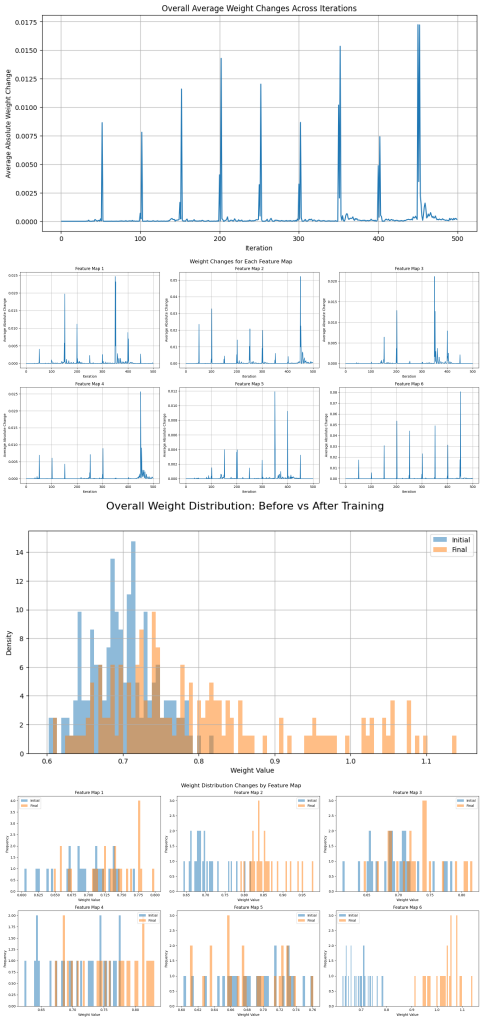

Figure 16: Average weight change across iterations.

Additionally, there is a local inter-map competition for STDP. When a neuron is permitted to perform STDP, it inhibits neurons in other maps within a small neighborhood around its location from performing STDP. This local competition is crucial as it encourages neurons from different maps to learn different features.

Synaptic weights of convolutional neurons are initialized with random values drawn from a normal distribution with a mean (μ) of 0.6 and a standard deviation (σ) of 0.05. Choosing a small μ would prevent neurons from reaching their firing threshold, thereby inhibiting learning. On the other hand, selecting a large σ would result in some initial synaptic weights being significantly smaller or larger than others, leading to unequal contributions to neuron activity. According to the STDP rule, weights with smaller initial values would have a higher tendency to converge to zero, while larger initial weights would tend to converge to one. This implies that a larger σ increases the dependency on the initial weights.

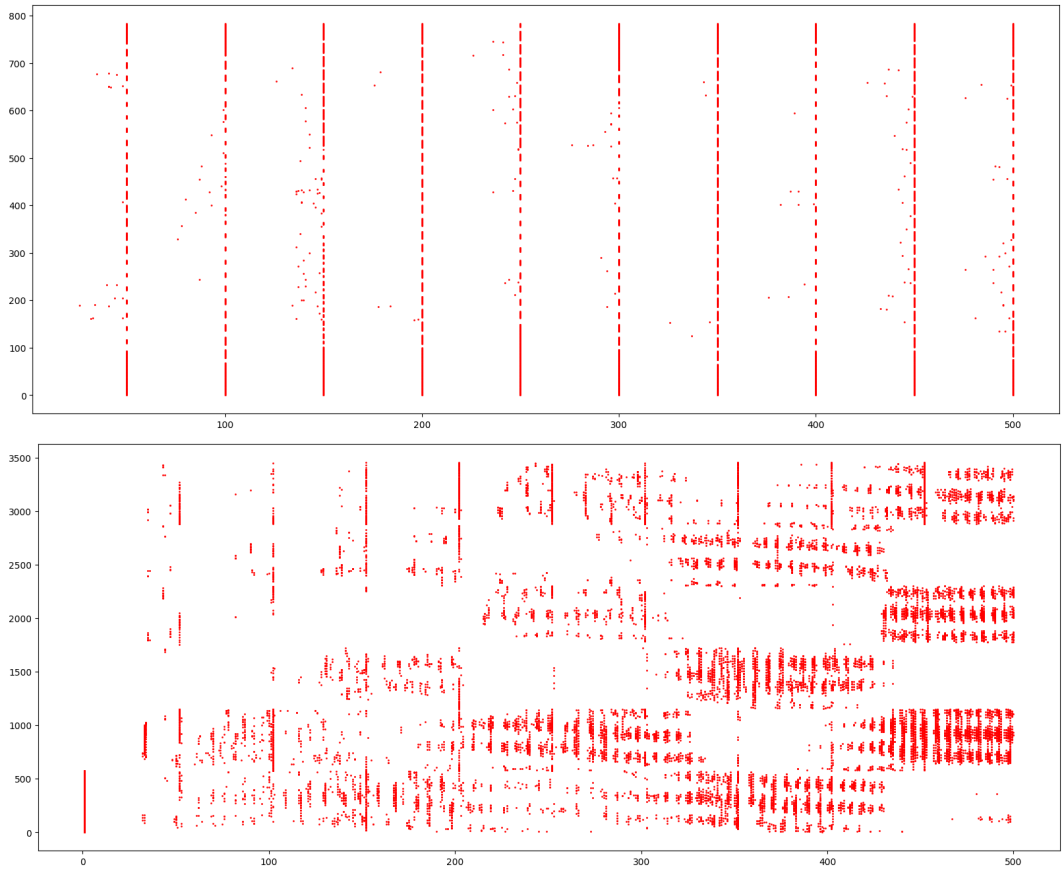

Figure 17: Raster plot of input and output layers spikes.

Raster plots for input and output spikes can be seen in Figure . Ten images were presented to the network for 20 timesteps, followed by a 20-timestep interval between each image. The effect of Voltage-Based Homeostasis is evident at the end of the iterations. Additionally, each time, one of the maps was selected by the KWTA mechanism, clearly delineating the maps in the raster plot.

The results of the learning process are presented in Figures 15 to 16. Figure 15 illustrates the concept of weight sharing and the changes in synaptic weights; when even a single neuron in the feature map spikes, all weights in the map are adjusted. Figure 16 shows the overall weight changes for each feature map.

Each feature map's weight change is displayed in the top row of Figure 18. For instance, the first feature map likely tries to identify vertical-right shapes. This means that the neurons within this map become more responsive to patterns that have strong vertical components tilted to the right.

The second and fourth maps appear to capture oriented lines. This suggests that neurons in these maps are tuning to line segments oriented at specific angles, making them sensitive to various line orientations which are crucial for recognizing edges and contours in visual data.

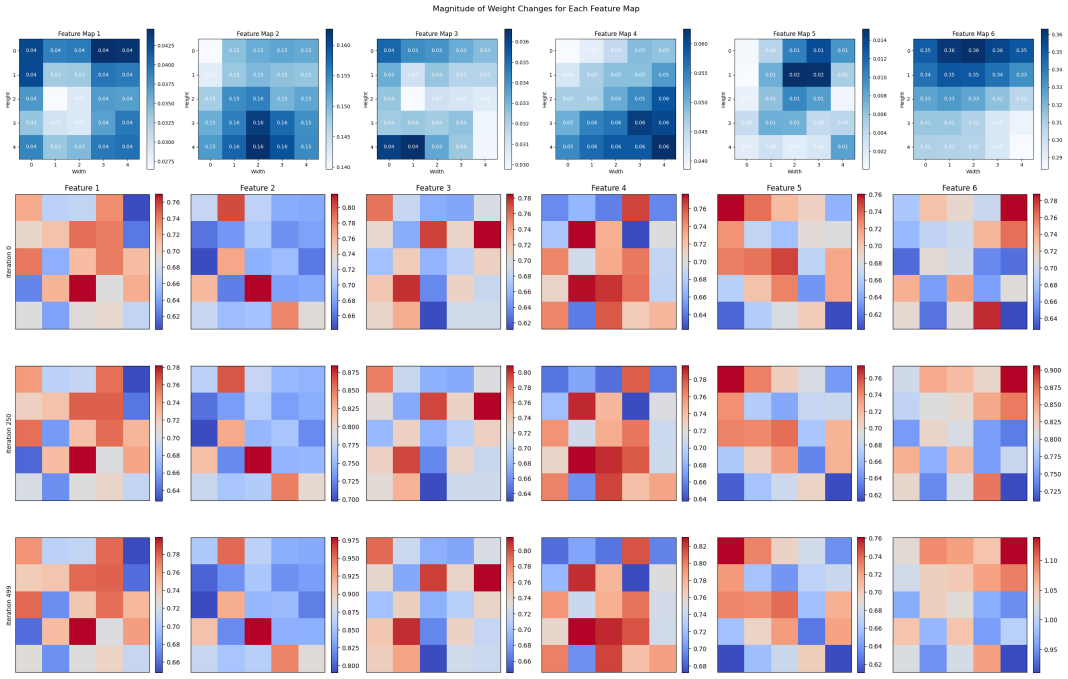

Figure 18: Features and their change over iterations.

The last map seems to focus on horizontal-top features. In this case, the neurons are fine-tuned to detect horizontal patterns located towards the top of the visual input, which could be indicative of certain structure or texture orientations that are prevalent in the training images.

Although the features represented in the bottom row of Figure 18 are not as apparent, we can still infer some characteristics based on the patterns of weight changes detailed in Figures 15 to 16, as well as the corresponding plots. For example, the weight changes indicate how the synaptic strengths are adjusted in response to the input stimuli. By examining these adjustments, we can deduce the types of features that the neurons are learning to detect. These synaptic modifications typically reflect patterns in the input data that are relevant to specific features, even if they are not immediately obvious from a visual inspection of the plots alone.

Overall, by closely analyzing these weight changes and understanding the specific map responses, we gain insight into the kinds of visual features the network is learning to recognize and how it differentiates between various patterns in the input data.