Monday, 19 August 2024

Articles Notes

CCNN

Introduction

1. Continuous Convolutional Kernels:

- Traditional CNNs use discrete convolutional kernels (e.g., 3x3, 5x5 filters) that slide over the image, capturing local spatial features.
- **Continuous convolutional kernels** imply that the kernel's function is continuous, allowing it to model dependencies in a more fluid and potentially more detailed manner. This means it can adapt more flexibly to the data and capture information across different scales and distances within the image.

2. Model Long-Range Dependencies at Every Layer:

- Traditional CNNs are limited in their ability to capture long-range dependencies (relationships between distant pixels) within the image, especially in earlier layers, because they typically rely on small kernels that focus on local features.
- With continuous convolutional kernels, the network can model these long-range dependencies directly at every layer, not just in the deeper layers where receptive fields have grown large enough due to stacking multiple layers.

3. Remove the Need for Downsampling Layers:

- Downsampling layers (like max pooling or strided convolutions) are traditionally used in CNNs to reduce the spatial resolution of the image, which helps to increase the receptive field and reduce computation.
- If every layer can model long-range dependencies effectively, there is less need to downsample the image. The network can maintain high-resolution features throughout, potentially preserving more detail and improving performance on tasks requiring fine-grained information.

4. Remove the Need for Task-Dependent Depths:

- In traditional CNNs, the depth of the network (number of layers) is often designed or adjusted based on the specific task (e.g., image classification vs. object detection). Deeper networks are used to capture more complex patterns.
- If the continuous convolutional kernels can model both local and global dependencies at every layer, then the network's depth may become less critical. A shallower network might achieve the same or better performance, reducing the need to tailor the architecture's depth to specific tasks.

5. Implications:

This approach could simplify CNN architectures by reducing the need for downsampling and potentially leading to more general-purpose, robust networks that do not require task-specific tuning of depth or structure. It might also lead to better performance on tasks involving complex spatial dependencies, as the network can maintain high-resolution data and capture long-range interactions more effectively.

How it works

1. Continuous Kernel Convolutions

Continuous kernel convolutions parameterize convolutional kernels as continuous functions by using a small neural network $G_{Kernel}: R^D \rightarrow R^{Nout \times Nin}$ as a kernel generator network. This network maps a coordinate $c_i \in R^D$ to the value of the convolutional kernel at that position: $G_{Kernel}(c_i) \in R^{Nout \times Nin}$. By passing a vector of K coordinates $[c_i]_{i \in [1,...,K]}$ through G_{Kernel}, a convolutional kernel K of equal size can be constructed, i.e., $K=[G_{Kernel}(c_i)]_{i \in [1,...,K]}$.

Subsequently, a convolution operation takes place between an input signal $x : R^D \rightarrow R^{Nin}$ and the generated convolutional kernel $K : R^D \rightarrow R^{Nout \times Nin}$ to construct an output feature representation $y : R^D \rightarrow R^{Nout}$. That is: y=ConvNd(K, x).

2. Continuous Kernel Convolutions Example

- Each coordinate c_i is fed into the GKernel, which is a small neural network.
- Suppose GKernel has been trained and, for example:
  - For $c_1 = (-1, -1)$, GKernel outputs a 3x5 matrix:

$$K(c_1) = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \\ 0.6 & 0.7 & 0.8 & 0.9 & 1.0 \\ 1.1 & 1.2 & 1.3 & 1.4 & 1.5 \end{bmatrix}$$

  - For $c_2 = (0, -1)$, GKernel outputs another 3x5 matrix:

$$K(c_2) = \begin{bmatrix} 0.2 & 0.3 & 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 & 1.0 & 1.1 \\ 1.2 & 1.3 & 1.4 & 1.5 & 1.6 \end{bmatrix}$$

3. Convolution Operation:

- To convolve the image with this kernel, each pixel neighborhood in the image is multiplied by the respective kernel matrices $K(c_1), K(c_2), \ldots, K(c_9)$, and the results are summed to produce the output.

4. Continuous Nature:

- If the coordinates c_i change slightly (e.g., shifting the kernel slightly), the GKernel will produce different matrices, adapting the convolutional kernel to the new positions. This makes the kernel continuous and adaptable.

5. Running Example

1. **Position (-1, -1)**:

$$\text{Output from } (-1,-1) = \begin{bmatrix} r_{-1,-1} & g_{-1,-1} & b_{-1,-1} \end{bmatrix} \times K(c_{-1,-1})$$

$$= \begin{bmatrix} r_{-1,-1} & g_{-1,-1} & b_{-1,-1} \end{bmatrix} \times \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \\ 0.6 & 0.7 & 0.8 & 0.9 & 1.0 \\ 1.1 & 1.2 & 1.3 & 1.4 & 1.5 \end{bmatrix}$$

$$= \begin{bmatrix} r_{-1,-1} \times 0.1 + g_{-1,-1} \times 0.6 + b_{-1,-1} \times 1.1, & \ldots (5 \text{ output values}) \end{bmatrix}$$

2. **Position (-1, 0)**:
   Similarly, for position $(-1, 0)$:

$$\text{Output from } (-1,0) = \begin{bmatrix} r_{-1,0} & g_{-1,0} & b_{-1,0} \end{bmatrix} \times K(c_{-1,0})$$

3. **Repeat for All Positions**:
   - We do this multiplication for each of the 9 positions, obtaining a vector of 5 values from each position.
4. **Summing the Results**:
   - Finally, sum the 9 vectors (each of length 5) to get the output for the pixel at position (x, y):

$$\text{Output at } (x,y) = \sum_{i,j} \text{Output from } (i,j)$$

   This results in a 5-channel output for the pixel (x, y).

- **Step 1**: Suppose you have a small image of 5×5 pixels with 3 channels.
- **Step 2**: For each pixel, the GKernel generates a 3×3 convolutional kernel for each of the 3 input channels.
- **Step 3**: Apply these kernels depthwise on the image, generating intermediate outputs for each channel.
- **Step 4**: Use a 1×1 pointwise convolution to combine the channels into the desired 6 output channels.

Implementation

Context: Kernel Generator Networks (GKernel)

In traditional convolutional neural networks (CNNs), convolutional kernels (or filters) are typically initialized with specific techniques designed to maintain a balance between input

1. Depthwise Separable Convolutions

- **Traditional Convolution**:
  - In a standard convolutional layer, a single kernel (or filter) is applied across all input channels simultaneously to produce each output channel. If you have 32 input channels and want 64 output channels with a 3×3 kernel, you'd need $3 \times 3 \times 32 \times 64$ parameters.
- **Depthwise Separable Convolution**:
  - **Depthwise Convolution**: Instead of applying a single kernel across all channels, a separate kernel is applied to each input channel individually. For 32 input channels with a 3×3 kernel, you'd only need $3 \times 3 \times 32$ parameters, one set per channel.
  - **Pointwise Convolution**: After the depthwise convolution, a 1×1 convolution (pointwise) is applied to combine the information from different channels, which maps the result from N_{in} channels to N_{out} channels. This involves 32×64 parameters.
  - **Benefit**: This separation greatly reduces the number of parameters and computations, enabling more efficient models.

and output variance. This balance is crucial to avoid problems like vanishing or exploding gradients, which can severely hinder the training process.

However, when using a neural network (GKernel) to generate these convolutional kernels dynamically based on input coordinates, there's an additional layer of complexity. The initialization of this GKernel affects the properties of the generated kernels and, consequently, the entire convolutional layer's behavior.

2. Traditional Kernel Initialization: Preserving Variance

- **Goal**: When initializing a convolutional layer, the goal is to ensure that the variance of the output of the layer matches the variance of the input (i.e.,Var(x)=Var(y)).
- **Initialization Formula**: Typically, convolutional kernels are initialized so that their variance is given by:

$$\mathrm{Var}(K) = \frac{\mathrm{gain}^2}{\text{in channels} \times \text{kernel size}}$$

  - **gain** is a factor that depends on the nonlinearity used (for example, in He initialization, the gain is rad(2) when using ReLU activations).
  - **in channels** refers to the number of input channels.
  - **kernel size** is the size of the convolutional kernel (e.g., 3×3).

- This formula helps ensure that the variance of the outputs does not blow up or diminish as data flows through layers, preserving the stability of training.

3. Problem with Standard Initialization for GKernel

When a neural network is used as a kernel generator (GKernel), it is typically initialized with the aim of preserving the unitary variance of its outputs (i.e., the output variance is 1). This is standard practice in initializing neural networks, ensuring that inputs and outputs have consistent variance, which is crucial for stable learning.

However, when this network is used to generate convolutional kernels, this initialization leads to the following issue:

- **Unitary Variance**: If the GKernel is initialized to have unitary variance (Var(K)=1), and then used to generate convolutional kernels, the variance of these kernels does not automatically adhere to the desired formula (i.e., it doesn't factor in the number of input channels and kernel size).
- **Layer-Wise Growth**: Because of this, the variance of the feature representations in the network grows layer by layer. Specifically, the variance grows proportional to

in channels × kernel size in channels×kernel size at each layer, which can result in the outputs (like logits) of the network reaching extremely large values in the order of 10^19

4. Consequences

This layer-wise growth in variance is problematic for several reasons:

- **Unstable Training**: The extremely high variance can lead to numerical instability during training. The network's outputs can become excessively large, which makes the gradients during back propagation also very large, potentially leading to exploding gradients.
- **Low Learning Rates**: To counteract this instability, very low learning rates might be required, which can slow down training significantly and make convergence difficult.

5. Proper Initialization for GKernel

To address this problem, the initialization of the GKernel should be adapted so that the convolutional kernels it generates have the correct variance from the start. Specifically:

- **Adjusted Initialization**: The GKernel should be initialized in such a way that the variance of the kernels it produces adheres to the formula:

$$\text{Var}(K) = \frac{\text{gain}^2}{\text{in channels} \times \text{kernel size}}$$

  Var(K)=in channels×kernel sizegain2 instead of having a unitary variance. This requires modifying the standard initialization methods used in neural networks, taking into account the role of GKernel as a generator of convolutional filters.

- **Implementation**: This could involve scaling the weights of the GKernel network or applying custom initialization schemes that ensure the generated kernels meet the desired variance criteria from the outset.

we re-weight the last layer of the kernel generator network by gain/ in channels · kernel size.

The kernel generator network G_{Kernel}. Our kernel generator network is parameterized as a 3-layer MAGNet (Romero et al., 2022a) with 32 hidden units for the $\text{CCNN}_{4,140}$ models, and 64 hidden units for the larger $\text{CCNN}_{6,380}$ models. The output size of the kernel generator network corresponds to the input channels of each layer in the network

FlexNet

Solving the Aliasing problem

However, deploying these models at higher resolutions can lead to a problem called **aliasing**, which the authors propose to solve using a novel method called **Multiplicative Anisotropic Gabor Networks (MAGNets)**.

1. FlexConvs and High-Resolution Deployment

- **FlexConvs**: These are a type of convolutional operation that can be used on images or other grid-based data. A unique feature of FlexConvs is their ability to operate at higher resolutions than those used during training.
- **High-Resolution Deployment**: When deploying FlexConvs at higher resolutions, you essentially use a finer grid of kernel indices. For example, if the model was trained on 64×64 images, you might want to use it on 128×128 images by sampling the kernels more densely.

2. The Problem of Aliasing

- **Alias**: In signal processing, aliasing occurs when a signal is sampled at a rate that is insufficient to capture its frequency content. This leads to distortion or artifacts in the signal. The same problem can occur in convolutional kernels: if the kernel has high-frequency components that exceed the Nyquist frequency (the highest frequency that can be accurately sampled), those components will be misrepresented, leading to aliasing.
- **Nyquist Frequency**: This is the maximum frequency that can be accurately represented given a certain sampling rate. If the bandwidth of the kernel exceeds this frequency, aliasing artifacts will appear when the model is used at higher resolutions.

3. Solution: Multiplicative Anisotropic Gabor Networks (MAGNets)

- **MAGNets**: These are a type of **Multiplicative Filter Network** designed to parameterize convolutional kernels. The key feature of MAGNets is that they allow for precise control over the frequency spectrum of the generated kernels. This means that you can regulate which frequencies the kernels will capture, avoiding those that would cause aliasing at higher resolutions.
- **Gabor Filters**: Gabor filters are known for their ability to capture specific frequency and orientation characteristics of a signal. By using an anisotropic version of these filters, MAGNets can adapt to different frequency requirements depending on the direction and scale, making them more flexible and powerful.

- **Multiplicative**: The term "multiplicative" in this context means that the network generates kernels by multiplying simpler filters together, allowing for a more complex and controlled frequency response.

4. Benefits of Using MAGNets

- **Regularization Against Aliasing**: By analyzing and controlling the frequency spectrum of the kernels, MAGNets can ensure that the kernels do not capture frequencies that would cause aliasing when deployed at higher resolutions. This regularization helps in maintaining accuracy even when the model is used on data with different resolutions.

- **Higher Descriptive Power**: MAGNets can describe more complex patterns and features in the data than previous continuous kernel parameterizations. This makes them more effective in capturing the intricacies of high-resolution data.

- **Faster Convergence**: Due to their structure and ability to control the frequency spectrum, MAGNets allow the network to converge (or learn) faster during training, which can save time and computational resources.

5. Impact on Classification Accuracy

The use of MAGNets in FlexConvs leads to better performance in classification tasks, especially when the model is deployed at resolutions different from those it was trained on. This is because the regularization against aliasing and the increased descriptive power allow the model to generalize better across different data scales.

1. Introduction

Gabor filters are a powerful tool in image processing and computer vision, known for their ability to analyze spatial frequencies and orientations in an image. Let's break down what Gabor filters are, how they work, and what it means to use an **anisotropic version** of these filters.

2. Function of Gabor Filters

- **Frequency Analysis**: The sinusoidal component of the Gabor filter allows it to capture information about the frequency content of the image. Different frequencies correspond to different levels of detail or texture in the image.
- **Orientation Sensitivity**: The orientation parameter θ allows the filter to detect edges or patterns at specific angles. For instance, a Gabor filter oriented at 45° will respond strongly to diagonal features in the image.
- **Localized Filtering**: The Gaussian envelope localizes the filter in the spatial domain, meaning the filter focuses on a specific region of the image. This is useful for detecting features in localized areas without being influenced by the entire image.

3. Anisotropic Gabor Filters

Anisotropy refers to the directional dependence of a property. In the context of Gabor filters, an anisotropic filter has different properties (such as frequency response or spread) in different directions.

Key Characteristics of Anisotropic Gabor Filters:

- **Elliptical Gaussian Envelope**: In anisotropic Gabor filters, the Gaussian envelope is typically elliptical rather than circular. This means that the filter has different spreads along different axes, controlled by the aspect ratio γ
    - When γ=1, the Gaussian envelope is circular, and the filter is isotropic (the same in all directions).
    - When γ<1, the envelope is elongated along one direction, making the filter more sensitive to features in that direction. This anisotropy allows the filter to capture textures and edges that are more pronounced in one direction than in others.

- **Directional Selectivity**: By adjusting the orientation θ and aspect ratio γ, anisotropic Gabor filters can be tuned to be more sensitive to specific directions, making them powerful for detecting oriented patterns, like edges or textures that vary in a preferred direction.
- **Frequency Control**: The wavelength λ and the standard deviation σ control the frequency content and the spatial extent of the filter, respectively. In anisotropic Gabor filters, these can be adjusted differently along different directions, allowing the filter to focus on specific frequencies depending on the direction.

4. Why Use Anisotropic Gabor Filters?

Anisotropic Gabor filters are more flexible than isotropic filters because they can be tailored to detect patterns with specific directional characteristics. This is particularly useful in applications like:

- **Texture Analysis**: Different textures in an image might have different dominant directions. Anisotropic Gabor filters can be used to detect these textures more effectively by aligning the filter's orientation with the texture's direction.
- **Edge Detection**: In images with edges or contours that are more prominent in one direction, anisotropic filters can be tuned to detect these edges with greater accuracy.
- **Feature Extraction**: For tasks like image classification or object recognition, anisotropic Gabor filters can extract features that are more descriptive of the underlying image structures, improving the model's ability to distinguish between different objects or textures.

5. Application in MAGNets

In the context of MAGNets (Multiplicative Anisotropic Gabor Networks):

- **Frequency Control**: MAGNets use anisotropic Gabor filters to carefully control the frequency spectrum of the convolutional kernels they generate. This control helps prevent aliasing when the model is deployed at higher resolutions.
- **Descriptive Power**: By using anisotropic Gabor filters, MAGNets can create more descriptive and flexible kernels, allowing the model to capture more complex patterns in the data.

- **Regularization**: The anisotropic properties of these filters allow for better regularization, meaning that the model can maintain high accuracy even when applied to data at different scales or resolutions.

To learn the kernel size during training, FlexConvs define their convolutional kernels ψ as the product of the output of a neural network **MLP**$^{\psi}$ with a Gaussian mask of local support. The neural network **MLP**$^{\psi}$ parameterizes the kernel, and the Gaussian mask parameterizes its size

Multiplicative Anisotropic Gabor Networks (MAGNets). Our MAGNet formulation is based on the observation that isotropic Gabor functions, i.e., with equal γ for the horizontal and vertical directions, are undesirable as basis for the construction of MFNs. Whenever a frequency is required along a certain direction, an isotropic Gabor function automatically introduces that frequency in both directions. As a result, other bases must counteract this frequency in the direction where the frequency is not required, and thus the capacity of the MFN is not used optimally (Daugman, 1988).

Following the original formulation of the 2D Gabor functions (Daugman, 1988), we alleviate this limitation by using anisotropic Gabor functions instead:

$$\mathbf{g}\big([x,y];\boldsymbol{\theta}^{(l)}\big) = \exp\left(-\frac{1}{2}\Big[\big(\boldsymbol{\gamma}_{\mathrm{X}}^{(l)}\big(x-\boldsymbol{\mu}_{\mathrm{X}}^{(l)}\big)\big)^2+\big(\boldsymbol{\gamma}_{\mathrm{Y}}^{(l)}\big(y-\boldsymbol{\mu}_{\mathrm{Y}}^{(l)}\big)\big)^2\Big]\right)\mathrm{Sin}\big(\mathbf{W}_{\mathrm{g}}^{(l)}[x,y]+\mathbf{b}_{\mathrm{g}}^{(l)}\big) \quad (7)$$

$$\boldsymbol{\theta}^{(l)}=\Big\{\boldsymbol{\gamma}_{\mathrm{X}}^{(l)}\in\mathbb{R}^{\mathrm{N_{hid}}},\boldsymbol{\gamma}_{\mathrm{Y}}^{(l)}\in\mathbb{R}^{\mathrm{N_{hid}}},\boldsymbol{\mu}_{\mathrm{X}}^{(l)}\in\mathbb{R}^{\mathrm{N_{hid}}},\boldsymbol{\mu}_{\mathrm{Y}}^{(l)}\in\mathbb{R}^{\mathrm{N_{hid}}},\mathbf{W}_{\mathrm{g}}^{(l)}\in\mathbb{R}^{\mathrm{N_{hid}}\times 2},\mathbf{b}_{\mathrm{g}}^{(l)}\in\mathbb{R}^{\mathrm{N_{hid}}}\Big\}. \quad (8)$$

The resulting *Multiplicative Anisotropic Gabor Network* (MAGNet) obtains better control upon frequency components introduced to the approximation, and demonstrates important improvements in terms of descriptive power and convergence speed (Sec. 4).

MAGNet initialization. Fathony et al. (2021) proposes to initialize MGNs by drawing the size of the Gaussian envelopes, i.e., the $\boldsymbol{\gamma}^{(l)}$ term, from a $\mathrm{Gamma}(\alpha\cdot \mathrm{L}^{-1},\beta)$ distribution at every layer $l\in[1,..,\mathrm{L}-1]$. We observe however that this initialization does not provide much variability on the initial extension of the Gaussian envelopes and in fact, most of them cover a large portion of the space at initialization. To stimulate diversity, we initialize the $\{\boldsymbol{\gamma}_{\mathrm{X}}^{(l)},\boldsymbol{\gamma}_{\mathrm{Y}}^{(l)}\}$ terms by a $\mathrm{Gamma}(\alpha l^{-1},\beta)$ distribution at the l-th layer. We observe that our proposed initialization consistently leads to better accuracy than the initialization of Fathony et al. (2021) across all tasks considered. (Sec. 4).

Appendix Frequencies and Images

To understand how different frequencies can be present in different directions in an image, let's break down the concept with a simple example and explanation.

1.Understanding Frequencies in Images

Frequencies in the context of images refer to how quickly pixel values change. High-frequency components represent rapid changes in pixel values (e.g., edges), while low-frequency components represent gradual changes (e.g., smooth regions).

Directional Frequencies mean that these frequency changes can be different along various directions (horizontal, vertical, diagonal, etc.).

2. Anisotropic vs. Isotropic Gabor Functions

1. **Isotropic Gabor Function**:
    - **What it does**: An isotropic Gabor function is like a "blurry" filter that applies the same frequency characteristics in all directions. If it detects a pattern or texture, it does so uniformly in every direction.
    - **Example**: Imagine you use an isotropic Gabor function to detect a pattern in an image with stripes. If the stripes are vertical, the function will also detect the same frequency in the horizontal direction, even though it's not needed.
2. **Anisotropic Gabor Function**:
    - **What it does**: An anisotropic Gabor function allows you to control the frequency differently in different directions. This means you can fine-tune the function to detect specific patterns along particular directions only.
    - **Example**: Suppose you have an image with vertical stripes. An anisotropic Gabor function can be designed to detect the frequency of these stripes specifically in the vertical direction, while ignoring or reducing sensitivity to horizontal frequencies. This is more efficient because it focuses only on the relevant direction.

3.Visualizing the Concept

Isotropic Gabor Function:

- Picture a filter that looks like a wavy pattern (like a sinusoidal wave). When applied to an image, it's like using a sieve with equal-sized holes in every direction. It detects patterns uniformly regardless of their orientation.

Anisotropic Gabor Function:

- Imagine a filter with elongated, directional waves. If you apply it to an image with vertical stripes, it's like using a sieve with elongated holes that align with the stripes. This filter will focus on the stripes' orientation and ignore other directions.

Illustration with a Simple Example

Image:

- Imagine an image with a clear vertical stripe pattern.

Using Isotropic Gabor Function:

- The filter will try to detect patterns both horizontally and vertically, which can be inefficient because the horizontal detection is unnecessary.

Using Anisotropic Gabor Function:

- The filter will be tailored to detect patterns specifically in the vertical direction where the stripes are present. This improves efficiency and accuracy by focusing on the relevant direction.

4.Summary

- **Isotropic Gabor Functions**: Treat all directions equally, which can lead to inefficiency if patterns are only present in certain directions.
- **Anisotropic Gabor Functions**: Allow you to customize frequency detection for specific directions, enhancing performance by focusing on relevant directions and ignoring others.

4.Understanding Aliasing

Aliasing in the context of images and signal processing refers to the distortion or artifacts that occur when a signal is sampled at a rate that is not high enough to accurately represent the original signal. This concept is crucial when working with Gabor functions or any other frequency-based analysis techniques.

Aliasing happens when high-frequency components of a signal (or image) are incorrectly represented as lower frequencies because the sampling rate is too low. This can cause misleading or incorrect interpretations of the original data.

5.How Aliasing Occurs

When an image or signal is sampled, it is effectively converted from a continuous domain to a discrete one. If the sampling rate is insufficient to capture all the variations in the signal,

higher frequencies may be misinterpreted as lower frequencies, resulting in artifacts or distortions.

6.Example of Aliasing

1. **In Signal Processing**:
    - **Original Signal**: Imagine a signal with a high frequency, like a rapidly oscillating sine wave.
    - **Sampling Rate**: If you sample this signal at too low a rate (fewer points per oscillation), the sampled data might not capture the rapid changes accurately.
    - **Aliased Signal**: The result might look like a completely different, slower oscillation or even appear as a constant value, distorting the true nature of the original signal.
2. **In Image Processing**:
    - **Original Image**: Consider an image with high-frequency patterns, such as fine lines or intricate textures.
    - **Sampling Rate**: If the resolution of the image is too low, these high-frequency details may not be represented accurately.
    - **Aliased Image**: The high-frequency patterns might appear as jagged lines, moiré patterns, or other artifacts that misrepresent the original image details.

7.Aliasing in Gabor Functions

In the context of Gabor functions and frequency analysis:

- **Aliasing** can occur if the sampling rate of the image is too low relative to the frequencies present in the Gabor functions. This can lead to the introduction of artifacts that do not accurately reflect the true frequency content of the image.
- **Gabor Functions**: When using Gabor functions to detect patterns, if the image is not sampled fine enough, high-frequency details can be misrepresented as lower frequencies, leading to less accurate feature detection.

8.Preventing Aliasing

To prevent aliasing:

1. **Increase Sampling Rate**: Ensure the sampling rate is high enough to capture the highest frequency components in the image or signal.
2. **Anti-Aliasing Filters**: Apply filters that reduce high-frequency components before sampling to avoid aliasing. In image processing, this can be done using techniques like blurring or smoothing to reduce high-frequency details before downsampling.

9.Summary

Aliasing is the distortion that occurs when high-frequency details are misrepresented due to insufficient sampling rates. In image processing and frequency analysis, like with Gabor

functions, aliasing can lead to inaccurate representations of image features, so proper sampling techniques and filtering are essential to avoid these artifacts.

Maximum frequency of MAGNets. The maximum frequency component of a MAGNet is given by:

$$f^{+}_{\text{MAGNet}} = \sum_{l=1}^{\text{L}} \max_{i_l} \left(\left(\max_{j} \frac{\mathbf{W}^{(l)}_{\text{g},i_l,j}}{2\pi} \right) + \frac{\sigma_{\text{cut}} \min\{\boldsymbol{\gamma}^{(l)}_{\text{X},i_l}, \boldsymbol{\gamma}^{(l)}_{\text{Y},i_l}\}}{2\pi} \right), \tag{21}$$

where L corresponds to the number of layers, $\mathbf{W}^{(l)}_{\text{g}}, \boldsymbol{\gamma}^{(l)}_{\text{X}}, \boldsymbol{\gamma}^{(l)}_{\text{Y}}$ to the MAGNet parameters as defined in Eq. 8, and σ_{cut}=2 $\cdot$ `stdev` to the cut-off frequency of the Gaussian envelopes in the Gabor filters. A formal treatment as well as the derivations can be found in Appx. A.1.

Effect of the FlexConv mask. The Gaussian mask used to localize the response of the MAGNet also has an effect on the frequency spectrum. Hence, the maximum frequency of a FlexConv kernel is:

$$f^{+}_{\text{FlexConv}} = f^{+}_{\text{MAGNet}} + f^{+}_{w_{\text{gauss}}}, \quad \text{with} \quad f^{+}_{w_{\text{gauss}}} = \frac{\sigma_{\text{cut}}}{\max\{\sigma_{\text{X}}, \sigma_{\text{Y}}\} 2\pi}. \tag{22}$$

Here, $\sigma_{\text{X}}, \sigma_{\text{Y}}$ correspond to the mask parameters (Eq. 1). Intuitively, multiplication with the mask blurs in the frequency domain, as it is equivalent to convolution with the Fourier transform of the mask.

Aliasing regularization of FlexConv kernels. With the analytic derivation of f^{+}_{FlexConv} we penalize the generated kernels to have frequencies smaller or equal to their Nyquist frequency $f_{\text{Nyq}}(k)$ via:

$$\mathcal{L}_{\text{HF}} = \| \max\{f^{+}_{\text{FlexConv}}, f_{\text{Nyq}}(k)\} - f_{\text{Nyq}}(k) \|^2, \quad \text{with} \quad f_{\text{Nyq}}(k) = \tfrac{k-1}{4}. \tag{25}$$

Here, k depicts the size of the FlexConv kernel before applying the Gaussian mask, and is equal to the size of the input signal. In practice, we implement Eq. 25 by regularizing the individual MAGNet layers, as is detailed in Appx. A.2. To verify our method, Fig. 8 (Appx. A.1) shows that the frequency components of FlexNet kernels are properly regularized for aliasing.

We analytically derive the maximum frequency of a MAGNet, and penalize it whenever it exceeds the Nyquist frequency of the training resolution.

10. Limitations

Dynamic kernel sizes: computation and memory cost of convolutions with large kernels. Per- forming convolutions with large convolutional kernels is a compute-intensive operation. FlexConvs are initialized with small kernel sizes and their inference cost is relatively small at the start of training. However, despite the cropping operations used to improve computational efficiency (Figs. 1, 3, Tab. 3), the inference time may increase to up to double as the learned masks increase in size.

Remaining accuracy drop in alias-free FlexNets. Some drop in accuracy is still observed when using alias-free FlexNets at a higher test resolutions (Tab. 4). Although more evidence is needed, this may be caused by aliasing effects introduced by ReLU (Vasconcelos et al., 2021), or changes in the activation statistics of the feature maps passed to global average pooling