SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Mathematics in Data Science

Transformations between fully connected and convolutional neural networks

Hanqi Huo

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Mathematics in Data Science

Transformations between fully connected and convolutional neural networks

Author:	Hanqi Huo
Supervisor:	Prof. Dr. Felix Dietrich
Advisor:	Prof. Dr. Felix Dietrich
Submission Date:	15.10.2024

I confirm that this master's thesis in mathematics in data science is my own work and I have documented all sources and material used.

Munich, 15.10.2024 Hanqi Huo

Abstract

This thesis explores the mathematical equivalence between convolutional neural networks (CNNs) and fully connected neural networks (FCNNs), focusing on transforming convolutional layers into fully connected layers. By expressing convolution operations as specific forms of matrix multiplications, I establish a formal equivalence that enables the conversion of a CNN into an FCNN with a specially structured weight matrix. The constructed matrix Q captures the sparsity and weight-sharing mechanisms inherent in CNNs within a fully connected framework. Additionally, I investigate the behavior of wide neural networks under prior assumptions, demonstrating that their output distributions converge to a Gaussian process as the network width approaches infinity. Through theoretical analysis and numerical experiments, I validate the univariate and multivariate normality of the outputs of wide CNNs and FCNNs. These findings provide valuable insights into neural network architectures, highlighting fundamental connections between different layer types and contributing to the understanding of neural networks' behavior in the infinite-width limit.

Contents

1 Introduction

In recent years, the incredible advancements in Convolutional Neural Networks (CNNs) (LeCun et al. 2015) have showcased their remarkable pattern-recognition abilities. These advancements have significantly enhanced critical tasks like image classification (Krizhevsky et al. 2012), speech processing (Hinton et al. 2012), and natural language processing (Kim 2014). CNNs have proven their superiority over traditional Fully Connected Neural Networks (FCNNs) in many domains, primarily due to their ability to exploit spatial and local dependencies in data. The efficiency of CNNs is largely attributed to their architectural design, which enables parameter sharing and local connectivity, making them both computationally efficient and effective at capturing the underlying structure of various types of input data, such as spatial patterns in images or temporal patterns in audio.

The fundamental operation within CNNs, known as the convolution operation, allows the network to extract features from local regions of the input, which are then aggregated through layers of convolutions and activations. This enables CNNs to capture low-level features, such as edges and textures in earlier layers, progressing to more abstract representations, such as objects and shapes in deeper layers. By utilizing convolutional layers and pooling layers, CNNs have consistently outperformed FCNNs in tasks requiring processing high-dimensional data like images and audio signals.

Despite the widespread success of CNNs, there is still much to explore regarding the theoretical underpinnings and relationships between different types of neural network architectures. One intriguing aspect is the mathematical relationship between CNNs and FCNNs. Specifically, the convolution operation in CNNs can be expressed as a matrix multiplication, making it mathematically equivalent to the dot product operation in FCNNs.

1.1 Research Objectives

The primary objective of this thesis is to analyze and demonstrate the transformation between a convolutional layer and a fully connected layer. The transformation is approached by converting the convolution operation into an equivalent matrix multiplication, allowing for a direct comparison between these two layers. This comparison is expected to show how weight sharing and local connectivity in CNNs contribute to their superior performance over FCNNs.

A second objective is to explore the Gaussian Process (GP) interpretation of infinitely wide neural networks. It has been established that neural networks with infinitely wide layers can be approximated under certain conditions by a GP, a powerful stochastic model for regression and classification tasks. This connection between neural networks and GPs offers a new perspective for understanding the behavior of neural networks, particularly in terms of output distributions and their convergence to normal distributions.

To achieve these objectives, this thesis is structured around two key parts:

1. **Transformation Between Convolutional and Fully Connected Layer:** Section 3.1 outlines the mathematical equivalence between convolutional operations in CNNs and matrix multiplications in FCNNs, focusing on how convolutional layers can be restructured into fully connected layers. A numerical comparison between these two architectures is also provided, demonstrating their equivalence in terms of output.
2. **Gaussian Process Interpretation of infinitely Wide Neural Networks:** Section 3.2 explores the theoretical framework of Gaussian Processes (GPs) in the context of neural networks, with a focus on infinitely wide Convolutional Neural Networks (CNNs). It examines how the output distributions behave as the network width increases under appropriate weight priors and applies GP interpretations to classification tasks. It also numerically tests the output distribution's probability and convergence properties.

1.2 Significance of the Study

Understanding the relationship between CNNs and FCNNs enhances the theoretical knowledge of deep learning architectures and improves the understanding of the efficiency of CNNs on images over FCNNs. Moreover, exploring GPs in infinitely wide neural networks opens up new avenues for applying statistical methods to deep learning.

1.3 Structure of the Thesis

The thesis is organized into four parts. To refine the text, I employ ChatGPT and Grammarly.

Intro 1: provides an overview of the study's motivation, research objectives, and significance.

State of the art 2: presents a comprehensive review of CNNs, FCNNs, the convolution operation, weight matrix decomposition of CNNs, and the role of Gaussian Processes in infinitely wide neural networks.

Main part 3: discusses the mathematical equivalence between convolutional layers and fully connected layers, presenting both theoretical analysis and numerical experiments;

investigates the behavior of infinitely wide neural networks under appropriate priors, supported by numerical experiments demonstrating the theoretical properties of output distributions.

Conclusion 4: summarizes the key findings of the thesis, discusses their implications, and suggests future research directions.

2 State of the art

2.1 Convolutional Neural Network

The content of this section is largely based on the comprehensive discussion provided in the book by Bengio et al. (2017). Convolutional networks (also known as convolutional neural networks or CNNs) are a specialized kind of neural network for processing data that has a known, grid-like topology. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

2.1.1 Key Advantages of CNNs over FCNNs

As discussed extensively in (Bengio et al. 2017), CNNs offer several advantages over FCNNs due to their ability to capture spatial hierarchies in data. Convolution leverages three key principles that significantly enhance the performance of machine learning systems: sparse interactions, parameter sharing, and equivariant representations. These principles allow for the encoding of specific properties of input images directly into the network architecture. Consequently, this results in a more efficient implementation of the forward function and a substantial reduction in the number of parameters compared to fully-connected neural networks.

Traditional neural network layers use matrix multiplication to describe the interactions between each input unit and each output unit, resulting in every output unit interacting with every input unit. In contrast, convolutional networks typically feature sparse interactions (also referred to as sparse connectivity or sparse weights). This is achieved by using kernels smaller than the input. For instance, when processing an image with thousands or millions of pixels, small, meaningful features, such as edges, can be detected using kernels that occupy only tens or hundreds of pixels. This reduces the number of parameters that need to be stored, decreasing the model's memory requirements and enhancing its statistical efficiency. Additionally, computing the output requires fewer operations. These efficiency improvements are substantial. With m inputs and n outputs, matrix multiplication requires $m \times n$ parameters and has a runtime of $O(m \times n)$ per example. By limiting the number of connections each output may have to k, the sparsely connected approach only requires $k \times n$ parameters and $O(k \times n)$ runtime. For many practical applications, it is possible to achieve good performance on the machine learning task while keeping k several orders of magnitude smaller than *m*. Figure 2.1 is a graphical demonstration of sparse connectivity. In a deep convolutional network, units in the deeper layers may indirectly interact with a larger portion of the input, as shown in Figure 2.2. This allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions.

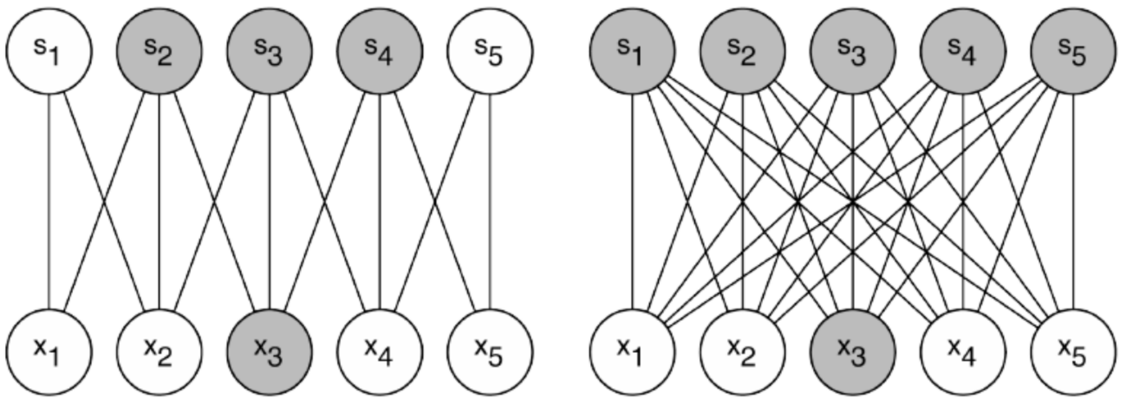


Figure 2.1: *Sparse connectivity, viewed from below:* One input unit, X_3, is highlighted along with the corresponding output units in **S** that are influenced by X_3. *(Left)* When **S** formed by convolution with a kernel of width 3, only three outputs are impacted by X_3. *(Right)* In contrast, when **S** is formed by matrix multiplication, connectivity loses sparsity, causing all outputs to be affected by X_3. This figure is taken from Bengio et al. (2017).

Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional neural network, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, it can be said that a network has tied weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a convolutional neural network, each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary). The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, only one set is learned. This does not affect the runtime of forward propagation—it is still $O(k \times n)$—but it does further reduce the storage requirements of the model to k parameters. Recall that k is usually several orders of magnitude less than m. Since m and n are usually roughly the same size, k is practically insignificant compared to $m \times n$. Convolution is thus dramatically more efficient than dense matrix multiplication in terms of memory requirements and statistical efficiency. For a graphical depiction of how parameter sharing works, see Figure 2.3

In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$

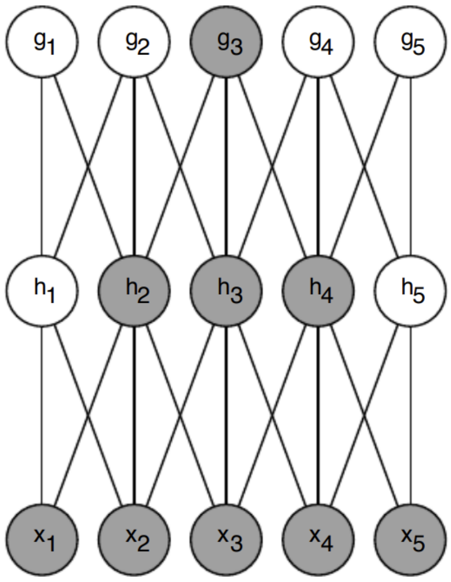


Figure 2.2: The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution or pooling. This means that even though *direct* connections in a convolutional network are very sparse, units in the deeper layers can be *indirectly* connected to all or most of the input image.). This figure is taken from Bengio et al. (2017).

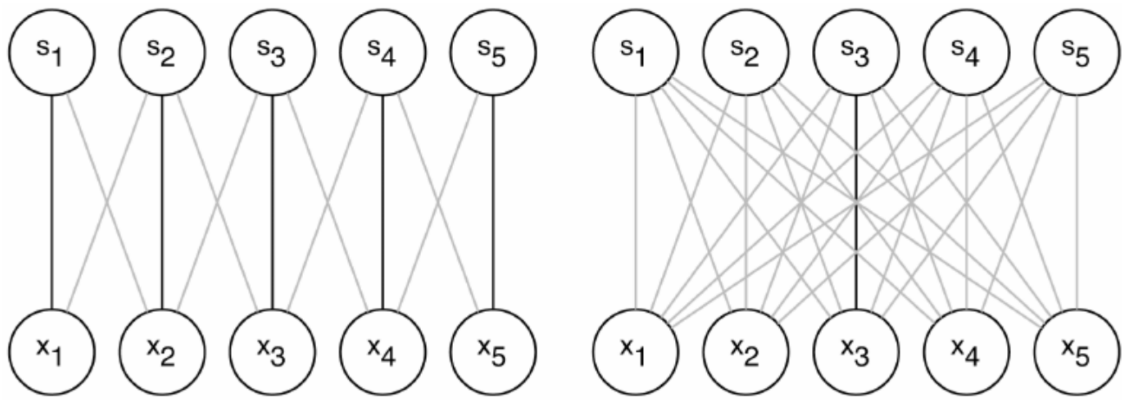


Figure 2.3: *Parameter sharing:* The connections that use a particular parameter in two different models are highlighted. *(Left)* The uses of the central element of a 3-element kernel in a convolutional model are highlighted. Due to parameter sharing, this single parameter is used at all input locations. *(Right)* The use of the central element of the weight matrix in a fully connected model is highlighted. This model has no parameter sharing, so the parameter is used only once. This figure is taken from Bengio et al. (2017).

is equivariant to a function g if $f(g(x)) = g(f(x))$. In the case of convolution, if g is any function that translates the input, i.e., shifts it, then the convolution function is equivariant to g. This characteristic is especially significant in image processing where convolution constructs a two-dimensional map indicating the locations of specific features within the input. Consequently, moving an object within the input results in an identical shift in its output representation. This is useful when the same local function is beneficial across the entire input. For example, when processing images, detecting edges in the first layer of a convolutional network is useful because the same edges appear throughout the image, making it practical to share parameters across the entire image. In some cases, sharing parameters across the entire image may not be desirable. For instance, when processing images cropped to be centered on an individual's face, different features need to be extracted at different locations—the part of the network processing the top of the face needs to look for eyebrows, while the part processing the bottom needs to look for a chin. It is important to note that convolution does not exhibit equivariance to other transformations, such as changes in scale or rotation, necessitating alternative mechanisms to manage such transformations.

2.1.2 The Convolution Operation

The convolution operation is a fundamental building block of CNNs, allowing them to detect local features in input data (Bengio et al. 2017).

Mathematically, the convolution operation between two functions, f and g, denoted as $f * g$, is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau.$$

In discrete terms, for functions defined at discrete points, this becomes:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m],$$

where n and m are indices in the data arrays of f and g.

In convolutional network terminology, the first argument (in this definition, the function f) to the convolution is often referred to as the input and the second argument (in this definition, the function g) as the kernel. The output is sometimes referred to as the feature map.

In machine learning applications, the input is usually a multidimensional array of data, and the kernel is a multidimensional array of learnable parameters. These multidimensional arrays are referred to as tensors. Since each element of the input and kernel must be explicitly stored separately, it is typically assumed that these functions are zero everywhere except at the finite set of points where the values are stored. This allows the infinite summation to be implemented as a summation over a finite number of array elements. Convolution operations are often used over more than one axis simultaneously. For instance, when using a two-dimensional image I as the input, a two-dimensional kernel K is likely also employed:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m, j-n].$$

It is noteworthy that convolution is commutative, allowing for the expression to be equivalently written as:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n].$$

The latter formulation is often more straightforward to implement in a machine learning library due to less variation in the range of valid values for m and n. While the commutative property is useful for writing proofs, it typically does not play a significant role in neural network implementations. Instead, many neural network libraries implement a related function called cross-correlation, which resembles convolution but does not involve flipping the kernel:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n]K[m,n].$$

Many machine learning libraries implement cross-correlation but label it as convolution. In this text, the convention of calling both operations convolution is followed, and clarification is provided regarding whether the kernel is flipped in contexts where kernel flipping is relevant.

See Figure 2.4 for an example of convolution (without kernel flipping) applied to a 2-D tensor.

Discrete convolution can be viewed as multiplication by a matrix. However, the matrix has several entries constrained to be equal to other entries. For example, for univariate discrete convolution, each row of the matrix is constrained to be equal to the row above shifted by one element. This is known as a Toeplitz matrix. In two dimensions, a *doubly block circulant matrix* corresponds to convolution. In addition to these constraints that several elements be equal to each other, convolution usually corresponds to a very sparse matrix (a matrix whose entries are mostly equal to zero).

While the conceptualization of convolution as matrix multiplication may not directly aid in its implementation, it offers valuable insights for understanding and designing neural networks. Neural network algorithms designed to operate via matrix multiplication, independent of specific matrix structural properties, can seamlessly incorporate convolution without necessitating adjustments to the neural network architecture. Although typical convolutional neural networks employ further optimizations for efficiently handling large inputs, these adaptations are not essential from a theoretical perspective.

2.1.3 Convolutional Layer

The convolutional layer applies a set of filters to the input, creating feature maps that represent various aspects of the input data (Bengio et al. 2017).

When discussing convolution in the context of neural networks, the functions used in practice differ slightly from the standard discrete convolution operation as it is usually understood in the mathematical literature. Convolution in neural networks usually refers to an operation that consists of many applications of convolution in parallel, allowing for the extraction of various features at multiple spatial locations. The input is typically a grid of

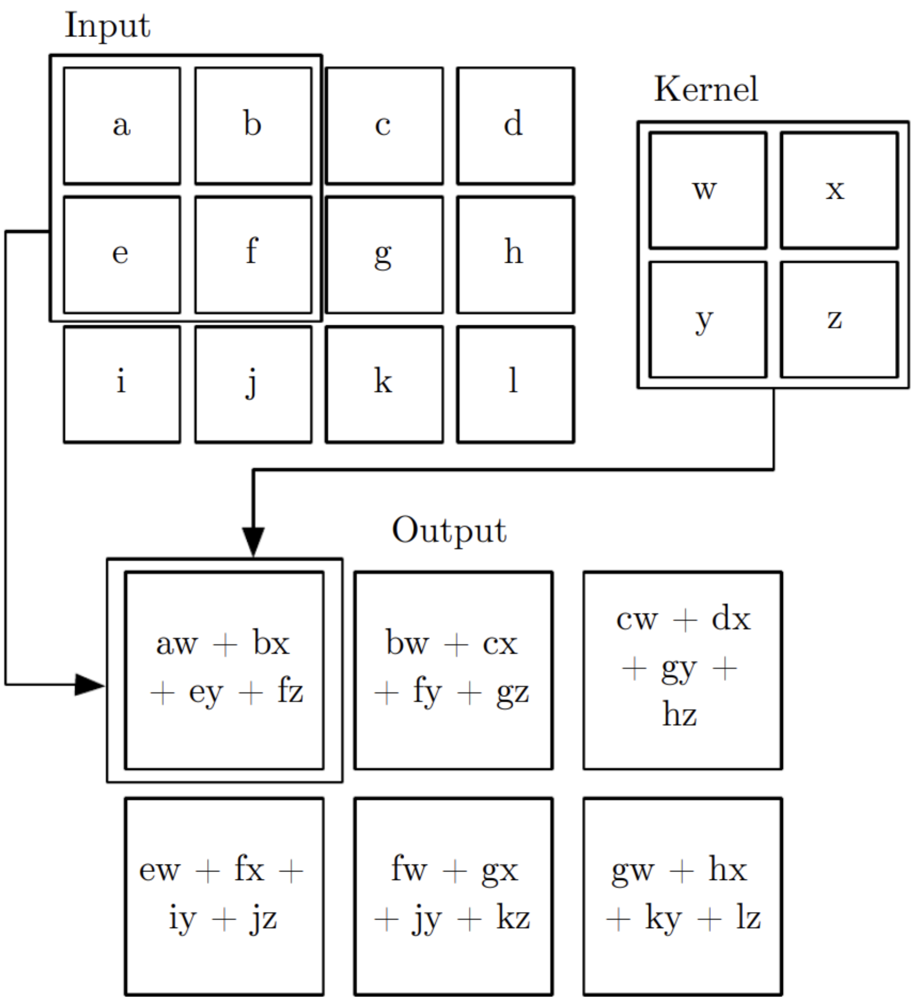


Figure 2.4: An example of 2-D convolution without kernel-flipping involves restricting the output to positions where the kernel lies entirely within the image, often termed "valid" convolution in some contexts. Boxes with arrows are used to illustrate how the upper-left element of the output tensor is generated by applying the kernel to the corresponding upper-left region of the input tensor. This figure is taken from Bengio et al. (2017).

vector-valued observations, such as a color image with red, green, and blue intensity at each pixel. In a multilayer convolutional network, the input to the second layer is the output of the first layer, which usually has the output of many different convolutions at each position.

Because convolutional networks often use multi-channel convolution, the linear operations they are based on are not guaranteed to be commutative, even if kernel-flipping is used. These multi-channel operations are only commutative if each operation has the same number of output channels as input channels. Assume a 4-D kernel tensor K with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel i of the output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit. Assume the input consists of observed data V with element $V_{i,j,k}$ giving the value of the input unit within channel i at row j and column k. If the output consists of Z with the same format as V, produced by convolving K across V without flipping K, then:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n}.$$

To reduce computational cost, downsampling can be employed, defined as:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j\cdot s+m,k\cdot s+n} K_{i,l,m,n},$$

where s is the stride of the downsampled convolution. Zero-padding the input is essential to prevent the output from shrinking with each convolutional layer, allowing control over the kernel width and output size independently. This ensures the network maintains its expressive power without rapidly shrinking the spatial extent of the representation. See Figure 2.5 for an example.

In convolutional layers, stride and padding are critical parameters that significantly influence the output dimensions and the computational efficiency of the network.

Stride refers to the number of pixels by which the filter (or kernel) moves across the input image or feature map. In a standard convolution operation, the stride is typically set to 1, meaning the filter moves one pixel at a time. However, using a stride greater than 1 results in a downsampled output, effectively reducing the spatial dimensions of the output feature map. This process is known as strided convolution.

Using a larger stride value reduces the overlap between receptive fields, which can speed up the computation and decrease the size of the output feature maps, making the model more computationally efficient. However, this also means that some spatial information is lost due to the reduced overlap.

Padding involves adding extra pixels around the border of the input image or feature map. This is typically done using zero-padding, where the added pixels are set to zero. Padding allows the size of the output feature map to be controlled. Without padding, the output size decreases with each convolutional layer, which can rapidly reduce the spatial dimensions of the data. By adding appropriate padding, it is possible to maintain the same spatial dimensions (width and height) of the output feature map as the input. This is known as "same" padding. Padding also ensures that the filter is applied to all pixels in the input image, including those at the edges, which helps in preserving spatial features near the borders.

In practice, padding can be categorized into valid padding and same padding. Valid padding (no padding) means no extra pixels are added, and the filter is only applied to valid positions within the input, resulting in an output feature map that is smaller than the input. Same padding involves adding pixels such that the output feature map has the same spatial dimensions as the input, with the amount of padding applied calculated based on the filter size and stride.

Combining stride and padding allows for flexible control over the spatial dimensions and computational efficiency of convolutional neural networks, enabling the design of deep architectures that can learn hierarchical features effectively from the input data.

The concept of the receptive field is also crucial in understanding convolutional neural networks. The receptive field of a particular layer in a CNN refers to the region of the input space that affects the values of that layer's output. In other words, it is the spatial extent of the input that contributes to the activation of a neuron in the feature map. As layers are added to the network, the receptive field typically increases, allowing the network to capture more complex and larger-scale features from the input data. Stride and padding influence the size and coverage of the receptive field. For example, larger strides reduce the overlap between receptive fields, thus increasing the distance between receptive fields of adjacent neurons in the feature map, while padding can help maintain the receptive field's coverage even at the edges of the input.

2.1.4 Pooling Layer

Pooling layers reduce the spatial dimensions of the feature maps, which helps in reducing the computational complexity and preventing overfitting (Bengio et al. 2017).

A typical layer in a convolutional network consists of three main stages. First, multiple convolutions are performed simultaneously on the input data to generate a set of initial feature maps. In the second stage, each feature map undergoes a nonlinear activation function, such as the rectified linear unit (ReLU), which introduces nonlinearity to the network and enables it to learn complex patterns. This stage is often referred to as the "detector" stage. The third stage involves applying a pooling function to the feature maps, which reduces their size by retaining only the most significant features.

A pooling function modifies the output of the network at a specific location by summarizing the values from a nearby region. For instance, the max pooling operation selects the maximum value from within a rectangular neighborhood. Other commonly used pooling functions include averaging the values in a rectangular neighborhood, calculating the L2 norm of the neighborhood, or applying a weighted average based on the distance from the central pixel.

Pooling operations enhance the invariance of representations to small translations of the input. This means that translating the input by a small amount typically does not alter the values of most of the pooled outputs. This property of local translation invariance is advantageous when the presence of a feature is more critical than its precise location. For instance, when determining if an image contains a face, it is sufficient to know that there are eyes on both sides of the face, without requiring pixel-perfect accuracy in their positions. Conversely, in cases where the precise location of a feature is crucial, such as identifying a

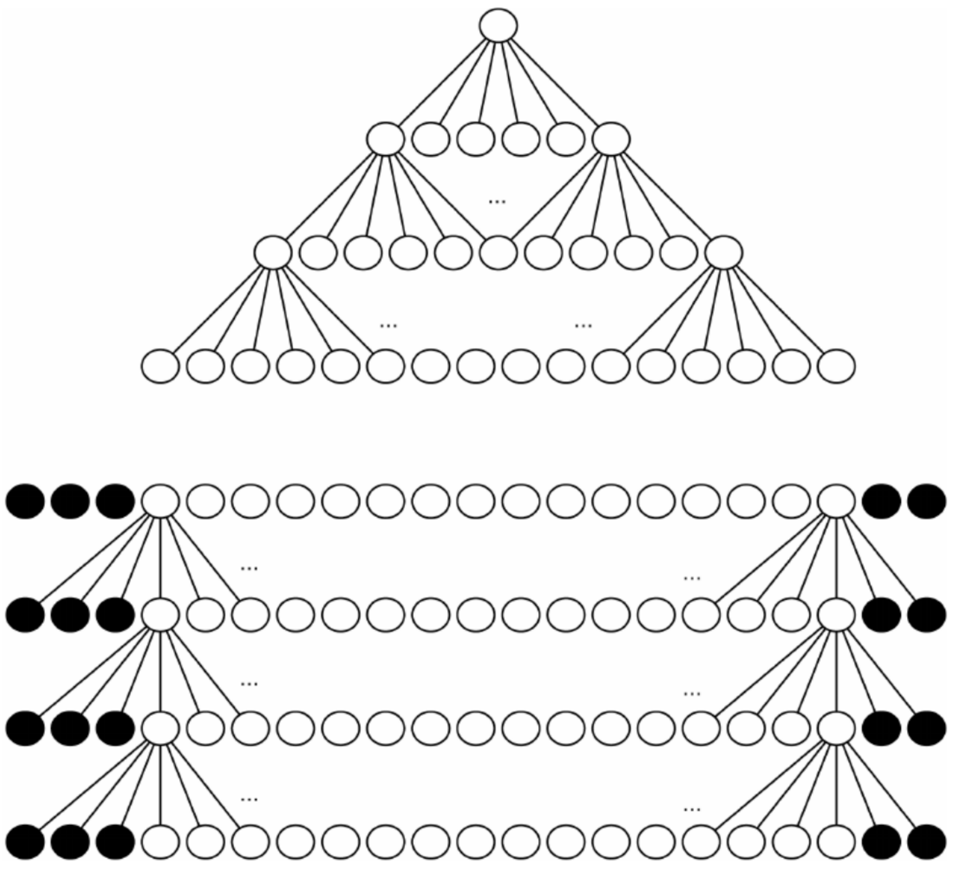


Figure 2.5: *The effect of zero padding on network size*: Consider a convolutional network with a kernel of width six at every layer. In this example, no pooling is used, so only the convolution operation itself affects the network size. *Top:* Without using any implicit zero padding, the representation shrinks by five pixels at each layer. Starting with an input of sixteen pixels, this setup allows for only three convolutional layers, with the last layer not moving the kernel, effectively making only two layers truly convolutional. The shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive, and some degree of shrinking is inevitable in this kind of architecture. *Bottom:* By adding five implicit zeros to each layer, the representation does not shrink with depth. This allows the construction of an arbitrarily deep convolutional network without losing spatial dimensions, maintaining the expressive power of the network. This figure is taken from Bengio et al. (2017).

corner formed by two edges meeting at a specific orientation, preserving the exact position of the edges is essential to accurately detect the corner.

The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network.

Pooling over spatial regions produces invariance to translation. However, by pooling over the outputs of separately parametrized convolutions, the features can learn to become invariant to specific transformations.

Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced k pixels apart rather than 1 pixel apart. This improves the computational efficiency of the network because the next layer has roughly k times fewer inputs to process. When the number of parameters in the next layer is a function of its input size (such as when the next layer is fully connected and based on matrix multiplication) this reduction in the input size can also result in improved statistical efficiency and reduced memory requirements for storing the parameters.

Pooling is essential for handling inputs of varying sizes in many tasks. For instance, in the classification of images with variable sizes, the input to the classification layer must be of fixed size. This is typically achieved by adjusting the size and offset between pooling regions to ensure the classification layer consistently receives the same number of summary statistics, regardless of input size. For example, the final pooling layer of a network may be designed to output four sets of summary statistics, one for each quadrant of an image, irrespective of the image size.

2.2 Convolutional Rainbow Network

A rainbow network is introduced as a probabilistic model of trained deep neural networks (Guth et al. 2023). It's constructed by cascading random feature maps, with the weight distributions of these maps being learned during training. A key aspect of rainbow networks is the reduction of dependencies between weights at different layers to rotations, which align input activations. This alignment results in neuron weights within a layer becoming independent.

In rainbow networks, the activations at each layer define kernels that converge to deterministic values as the network's width approaches infinity. Here, activations refer to the output of each layer in the neural network after applying the non-linear activation function to the linear transformations (weighted sums of inputs). In rainbow networks, the activations at each layer define kernels that converge to deterministic values as the network's width approaches infinity. Activations refer to the output of each layer in the neural network after applying the non-linear activation function to the linear transformations (weighted sums of inputs). These kernels are mathematical functions that measure the similarity between different input data points based on their activations. Specifically, for a one-hidden-layer random feature network, the kernel is defined as:

$$k(x,x') = \mathbb{E}_{w\sim\pi}\left[\rho(\langle x,w\rangle)\rho(\langle x',w\rangle)\right],$$

where ρ is a non-linear activation function, π is the probability distribution of the weights w, and $\langle x,w\rangle$ represents the dot product between the input x and the weight vector w. In the infinite-width limit, these kernels become deterministic, capturing stable and predictable patterns in the feature representations.

This deterministic behavior is verified numerically for networks such as ResNets trained on the ImageNet dataset. Additionally, the weight distributions in trained networks typically have low-rank covariances, leading to linear dimension reductions alternating with nonlinear high-dimensional embeddings using white random features. Gaussian rainbow networks, a specific type of rainbow network, use Gaussian weight distributions. These networks have been numerically validated on tasks like image classification with the CIFAR-10 dataset, demonstrating that stochastic gradient descent (SGD) primarily updates the weight covariances while preserving the initial Gaussian distribution of the weights.

The convolutional rainbow network extends the rainbow network model to convolutional architectures. This extension incorporates translation-equivariance and locality, fundamental properties in convolutional networks. The weight matrix in each convolutional layer is factorized into two components: a predefined operator P_j that acts along the spatial dimension and a learned pointwise operator L_j that acts along the channel dimension. This factorization enables the construction of convolutional layers that perform linear dimensionality reductions followed by high-dimensional nonlinear embeddings.

Convolutional rainbow networks maintain the theoretical properties of fully connected rainbow networks. These properties include the convergence of activations to deterministic kernel feature vectors in the infinite-width limit. As in fully connected rainbow networks, convolutional rainbow networks define kernels to measure the similarity between different input data points based on their activations. Specifically, the deterministic kernel $k_j(x,x')$ at layer j is given by:

$$k_j(x,x') = \sum_u \mathbb{E}_{w\sim\pi_j}\left[\rho\left(\langle P_j\phi_{j-1}(x)[u],w\rangle\right)\rho\left(\langle P_j\phi_{j-1}(x')[u],w\rangle\right)\right],$$

where $P_j\phi_{j-1}(x)[u]$ and $P_j\phi_{j-1}(x')[u]$ are patches of the activations $\phi_{j-1}(x)$ and $\phi_{j-1}(x')$ at layer $j-1$, centered at location u. The expectation $\mathbb{E}_{w\sim\pi_j}$ is taken over the weight distribution π_j, and ρ is the non-linear activation function. Additionally, the alignment properties of the original rainbow networks are preserved in the convolutional setting. Numerical results validate these theoretical properties, demonstrating that the network activations in trained convolutional rainbow networks converge to fixed deterministic limits after alignment as the width increases. This convergence ensures that convolutional rainbow networks can implement functions that are invariant to transformations, such as translations, and maintain high performance on tasks like image classification.

2.3 Weight Matrix Decomposition

Decomposing the weight matrix of Convolutional Neural Networks (CNNs) is an important technique aimed at improving computational efficiency and performance. This method divides the weight matrix into specific components, each optimized for different data processing tasks, thereby making CNNs more effective. A good example of this method in action is seen in Convolutional Rainbow Networks (Guth et al. 2023). Here, the weight matrix W_j at each layer is decomposed into two components: a prior convolutional operator P_j and a learned pointwise convolution L_j.

P_j is a fixed prior convolutional operator that acts along spatial dimensions and is replicated across channels, also known as depthwise convolution. It extracts patches of size $s_j \times s_j$ at each spatial location and reshapes them into vectors, creating a channel vector of size $d'_{j-1} = d_{j-1}s_j^2$, where d_{j-1} is the number of input channels. This operator is fixed during training and imposes the architectural constraints inherent to convolutional layers.

The learned operator L_j is a 1 × 1 convolutional operator that acts along channels. It processes the reshaped patches from P_j by performing 1 × 1 convolutions across these input channels, effectively learning the relationships between channels at each spatial location. This operator transforms the convolution kernel of W_j from a size of $d_j \times d_{j-1} \times s_j \times s_j$ into a 1 × 1 convolution L_j with a kernel size of $d_j \times d'_{j-1} \times 1 \times 1$, where d_j is the number of output channels.

By factorizing the weight matrix in this manner, the network can enforce translation equivariance and locality while allowing the weights to be learned in a constrained and structured manner. It ensures that identical random feature embeddings are computed for each input patch(by the fixed spatial operator P_j), thus maintaining consistency in feature representation across different spatial locations. The prior operator P_j can also incorporate predefined filters, such as wavelets, to further influence the learned operator L_j and the resulting weight distributions π_j. The adaptable channel-wise operator L_j then tailors the network's response to specific features, optimizing the extraction and processing of information across different channels.

Decomposing a large convolutional filter into a fixed spatial filter and a learned pointwise filter can reduce the number of parameters, leading to more efficient training and inference.

2.4 Gaussian Process Interpretations in Wide Neural Networks for Classification

In probability theory and statistics, a Gaussian process is a type of stochastic process, defined as a collection of random variables indexed by time or space, where every finite subset of these variables follows a multivariate normal distribution. The distribution of a Gaussian process represents the joint distribution of an infinite number of such random variables, making it a distribution over functions with continuous domains, such as time or space.

The term "Gaussian process" is derived from Carl Friedrich Gauss, as it is fundamentally based on the Gaussian (or normal) distribution. A Gaussian process can be viewed as an infinite-dimensional extension of the multivariate normal distribution.

Gaussian processes are highly useful in statistical modeling due to their connection with the normal distribution, which allows for explicit derivations of distributions for various quantities. For example, if a random process is modeled as a Gaussian process, quantities such as the average value of the process over a given interval and the associated estimation error from a small sample set can be computed exactly. Although exact models tend to scale poorly with increasing data size, various approximation methods have been developed to maintain high accuracy while significantly reducing computational complexity.

Recent advances in neural network theory have provided a strong connection between neural networks and Gaussian Processes (GPs), especially in the context of infinitely wide networks. This relationship allows for the reframing of neural network classification problems as regression tasks by leveraging GP theory. Two important studies that formalize this approach are the works by Garriga-Alonso et al. (2019) and Lee et al. (2018), which are highly relevant for understanding the behavior of the output distribution of neural networks and their convergence to GPs under certain conditions.

Lee et al. (2018) extend the GP framework to deep fully-connected neural networks (DNNs), showing that an infinitely wide deep neural network with i.i.d. priors over its parameters is equivalent to a GP. This equivalence facilitates Bayesian inference in neural networks, enabling them to perform well on standard tasks such as MNIST and CIFAR-10. The study introduces Neural Network Gaussian Processes (NNGPs), which provide a GP interpretation of DNNs. The key insight is that priors over the weights and biases, as the width of each layer increases, the network's output distribution converges to a GP. The GP framework offers an efficient way to compute the covariance functions of these deep networks, which can then be used for Bayesian inference. The authors show that as the network width increases, its performance approaches that of the corresponding GP, making GPs a powerful tool for understanding the distribution of outputs in neural networks.

Garriga-Alonso et al. (2019) build on this concept by demonstrating that convolutional neural networks (CNNs) with certain priors over the weights and biases behave like GPs in the limit of infinitely many convolutional filters. In this case, the CNN can be treated as a shallow GP, where the equivalent kernel can be computed exactly with a few parameters, making it efficient to compute. The study highlights that for classification tasks, such as MNIST, the GP approach based on CNNs achieves remarkable performance, outperforming some traditional GP methods. The equivalence between CNNs and GPs is established through the Central Limit Theorem (CLT), where, as the number of filters increases, the output of the CNN converges to a GP. The authors emphasize the practical advantage of this framework: it enables exact Bayesian inference for CNNs by reframing classification as a regression task under the GP paradigm.

In both DNN and CNN cases, the output distribution of the network converges to a Gaussian distribution as the network width increases, consistent with the Central Limit Theorem. This convergence facilitates efficient Bayesian training and makes GPs a natural framework for analyzing the output behavior of neural networks. Furthermore, reframing classification as a regression problem under GPs allows for exact Bayesian predictions, often outperforming traditional finite-width networks, especially as the network width increases.

These works provide a theoretical foundation for understanding the output distribution of neural networks, particularly in wide networks, and their behavior under GP priors.

3 Transformation between CNNs and FCNNs

3.1 One-Layer Neural Network Transformation

The convolution operation in Convolutional Neural Networks (CNNs) is mathematically equivalent to the matrix dot product performed in fully connected (dense) layers. Understanding this equivalence involves expressing the convolution operation as a matrix multiplication, revealing the fundamental similarities between these two types of neural network layers.

The weights of a fully connected layer can be interpreted as a convolutional operation with kernels of size 1×1 that span the entire depth of the input. In a fully connected layer:

$$Y = W \cdot X_{\text{flat}} + b,$$

where W is the weight matrix of size $M \times N$, with M being the number of output neurons and N the number of input features, X_{flat} is the flattened input vector of size $N \times 1$ and b is the bias vector of size $M \times 1$.

This operation can be seen as applying M different 1×1 convolutional kernels over the input feature map, where each kernel corresponds to a row in W.

Conversely, in a convolutional layer, the convolution operation involves sliding a kernel (filter) over the input feature map and computing the dot product at each spatial location. For simplicity, consider a single-channel input image X of size $H \times W$ and a single convolutional kernel K of size $k \times k$. The convolution operation produces an output feature map Y, where each element is computed as:

$$Y(i, j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} K(m, n) \cdot X(i + m, j + n).$$

To express this operation as a matrix multiplication, the following steps are performed:

1. Unfold the Input Image (Im2col): Transform the input image X into a matrix X_{unfolded} where each column corresponds to a vectorized receptive field (local region) over which the kernel is applied. For an input image of dimensions $H \times W$ and a kernel of size $k \times k$, X_{unfolded} will have k^2 rows and $(H - k + 1)(W - k + 1)$ columns.

2. Flatten the Convolutional Kernel: Flatten the convolutional kernel K into a vector k_{flat} of size $k^2 \times 1$.

3. Compute the Output Feature Map via Matrix Multiplication: Compute the output feature map by multiplying the transposed kernel vector k_{flat}^t (of size $1 \times k^2$) by the

unfolded input matrix X_{unfolded}:

$$Y_{\text{flat}} = k_{\text{flat}}^{t} \cdot X_{\text{unfolded}}.$$

This results in a row vector Y_{flat} of size $1 \times (H-k+1)(W-k+1)$.

4. Reshape the Output Vector: Reshape Y_{flat} back into the spatial dimensions of the output feature map Y of size $(H-k+1) \times (W-k+1)$.

The above process effectively transforms the convolution operation into a standard matrix multiplication commonly implemented in programming languages.

When replicating the dot product operation in the fully connected layer, I transform the convolutional kernels into a matrix Q, whose construction process is illustrated in Section 3.1.1, and directly flatten the three-dimensional input image into a vector.

The structure of the matrix Q, sparse and structured, demonstrates that convolutional layers naturally distribute weights across various spatial locations, reflecting the local connectivity of convolutional layers. In contrast, fully connected layers do not share weights. The matrix in fully connected layers is typically dense, linking every input to every output neuron.

By representing convolutional operations as matrix multiplications and interpreting fully connected layer weights as convolutional kernels, I establish a mathematical equivalence between these two types of neural network layers. This revelation emphasizes that both layers conduct linear transformations of the input data, primarily differing in weight sharing, connectivity patterns, and computational efficiency.

3.1.1 Construction of the Matrix Q

Let the input image have dimensions (W, H, C), where W and H are the width and height of the image, and C is the number of input channels. The convolutional kernel K has dimensions (M, C, k, k), where M is the number of kernels, and $k \times k$ is the kernel size.

The output dimensions are determined by the stride S and padding P, given by:

$$W_{\text{out}} = \left\lfloor \frac{W-k+2P}{S} \right\rfloor + 1,$$

$$H_{\text{out}} = \left\lfloor \frac{H-k+2P}{S} \right\rfloor + 1,$$

where W_{out} and H_{out} are the output width and height, and $\lfloor \cdot \rfloor$ denotes the floor function. Padding P is the number of pixels added around the borders of the input, and stride S defines the step size by which the filter moves across the input.

The matrix Q is designed to simulate the convolution operation within a fully connected neural network structure. It transforms the flattened input image into a form that can be multiplied by the kernel weights. The dimensions of Q are $(M \times W_{\text{out}} \times H_{\text{out}}, W \times H \times C)$, where $M \times W_{\text{out}} \times H_{\text{out}}$ represents the total number of output pixels across all kernels, and $W \times H \times C$ represents the flattened input image dimensions.

The matrix Q is constructed by vertically stacking blocks B_m, each corresponding to the m-th kernel:

$$Q = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_M \end{bmatrix}.$$

Each block B_m has dimensions $(W_{\text{out}} \times H_{\text{out}}, W \times H \times C)$. The rows of B_m represent the application of the m-th kernel at specific output pixel locations, and the columns correspond to the pixels in the flattened input image.

For each kernel m and each output pixel location (i, j) in the output feature map, the corresponding kernel weights are located in row q_i of matrix Q.

For each input channel c and each kernel element at position (d_i, d_j), where d_i, d_j range from 0 to $k - 1$, the kernel element $K[m, c, d_i, d_j]$ is placed at position $Q[q_i, q_j]$ in matrix Q. The row and column indices q_i and q_j are calculated as:

$$\begin{aligned} q_i &= m \times W_{\text{out}} \times H_{\text{out}} + i \times W_{\text{out}} + j, \\ q_j &= c \times W \times H + (i \times S + d_i - P) \times W + (j \times S + d_j - P). \end{aligned}$$

Since the input image has multiple channels, each of size $W \times H$, the offset for the c-th channel in the flattened input image is calculated as $c \times W \times H$.

It is crucial to check the validity of the column index to ensure that the convolution interacts only with valid pixels in the input image. The indices $(i \times S + d_i - P)$ and $(j \times S + d_j - P)$ must satisfy:

$$0 \le (i \times S + d_i - P) < H \quad \text{and} \quad 0 \le (j \times S + d_j - P) < W.$$

If the input indices are valid, the weight of the kernel element $K[m, c, d_i, d_j]$ is assigned to the corresponding position in matrix Q:

$$Q[q_i, q_j] = K[m, c, d_i, d_j].$$

By systematically repeating the above calculations for each kernel m, each output pixel location (i, j), each input channel c, and each kernel element (d_i, d_j), the matrix Q is fully populated. Each block B_m within Q applies a specific kernel across the entire input image. The resulting matrix Q simulates the convolution operation when multiplied by the flattened input image.

Visual Representation of B_m

Consider the m-th kernel applied to the first input channel (i.e., $c = 0$) with a kernel size of $k = 3$. The weights of this kernel are represented as:

$$K[m][0] = \begin{bmatrix} k_{00} & k_{01} & k_{02} \\ k_{10} & k_{11} & k_{12} \\ k_{20} & k_{21} & k_{22} \end{bmatrix}.$$

Assuming that the convolution operation uses 'same' padding and that the top-left pixel (position $(0,0)$) in the output feature map corresponds to the top-left corner of the input image, the corresponding row vector in B_m is:

$$[k_{00}, k_{01}, k_{02}, 0, 0, \dots, 0, k_{10}, k_{11}, k_{12}, 0, 0, \dots, 0, k_{20}, k_{21}, k_{22}, 0, \dots, 0].$$

This arrangement aligns the kernel weights with corresponding pixels in the flattened input vector, padded with zeros elsewhere.

As subsequent rows in B_m are processed, this kernel pattern shifts to the right, aligning with the next receptive field in the input. Additionally, it moves down by W_{out} columns to align with the next row in the output feature map.

The structure of the matrix Q reveals key characteristics of the convolutional layer, such as weight sharing and locality. The matrix Q contains many zeros and is composed of repeated blocks of weights, which highlights the localized nature of convolution operations. This block structure contrasts with the dense, fully connected layers, where each element in the weight matrix is non-zero. These differences illustrate the computational efficiency of the convolutional layer compared to the fully connected layer, as fewer parameters are required while still preserving important spatial in the input data.

3.1.2 Numerical Comparison

In this section, I explore the relationship between the weights of a single-layer Convolutional Neural Network (CNN) and a single-layer Fully Connected Neural Network (FCNN) by setting the weight matrix of the FCNN as Q, constructed according to the steps outlined in Section 3.1.1. The experiment is divided into three parts:

First, a single-layer CNN, consisting of one convolutional layer and one fully connected layer acting as a classifier, was trained on the MNIST and CIFAR-10 datasets. I set the kernel size of the convolutional layer as $(100, C, 3, 3)$, where $C = 1$ for the MNIST dataset and $C = 3$ for CIFAR-10 datasets. In this setup, the kernel processes input images with either one channel (grayscale) or three channels (RGB), depending on the dataset used. The training process ran for 50 epochs with an early stopping patience of 5, resulting in a final training duration of approximately 30–40 epochs. The `CrossEntropyLoss` function was used as the loss function, and Stochastic Gradient Descent (SGD) with a learning rate of 0.001 and momentum of 0.9 was employed as the optimizer. Data preprocessing included normalization, with pixel values scaled to the range $[0, 1]$.

Second, the weight matrix of the FCNN, comprising one fully connected layer and one classification layer, was initialized to correspond with the weights of the CNN. Specifically, the weights of the first fully connected layer were derived using the reconstructed matrix Q (as described in Section 3.1.1), and the biases were obtained by applying repeat interleave operations on the biases of the convolutional layer. In particular, the PyTorch function `repeat_interleave()` was utilized:

$$\texttt{fnn\_biases} = \texttt{fc1\_bias.repeat\_interleave}(H_{\text{out}} \times W_{\text{out}}),$$

where (W_{out}, H_{out}) represent the dimensions of the output feature maps. I tested the classification accuracy of both neural networks on the same datasets, and the results showed identical accuracies, demonstrating that the two neural networks are numerically equivalent. This part of the experiment validated the theoretical structural consistency between the two neural networks through numerical results.

3.2 Gaussian Process Behaviour in infinitely wide Neural Networks

It has long been known that a single-layer fully-connected neural network with an i.i.d. prior over its parameters is equivalent to a Gaussian process (GP) (Neal 1994), in the limit of infinite network width.

By mathematically deriving the output distribution of convolutional neural networks (CNNs) in Section 3.2.1 under appropriate prior assumptions, a similar correspondence to that of fully connected networks can be demonstrated for single-layer CNNs. Specifically, when the weights and biases of both the convolutional layer and the final fully connected layer are independently and identically distributed (i.i.d.) according to a normal distribution, and the variance of the fully connected layer is scaled proportionally to the inverse of M, where M is the number of convolutional kernels, the output distribution of the CNN converges to a normal distribution as M approaches infinity.

The broader requirements of weights and biases priors for the same probability properties of output distribution are discussed Section 3.2.2.

Studying this correspondence is crucial for understanding the probabilistic behavior of CNNs in the infinite-width regime. By leveraging the GP correspondence, we can model the output of both infinitely wide CNNs and infinitely wide FNNs as a distribution over functions, enabling the application of Bayesian methods to tasks such as regression and classification with exact posterior inference.

3.2.1 Output Distributions of infinitely wide NNs Under Prior Assumptions

Lemma 1. *Let a convolutional neural network (CNN) consist of a single convolutional layer followed by a ReLU activation and a fully connected (FC) layer without an activation function. Assume the weights and biases of the convolutional layer are independent and identically distributed, each following a normal distribution with zero mean and variances σ_F^2 and σ_b^2. The weights in the FC layer are also i.i.d. with zero mean and variance $1/M$, where M is the number of convolutional kernels, and the biases are i.i.d. normally distributed with zero mean and finite variance σ_d^2 . Then, as M approaches infinity, the output distribution of the CNN converges to a normal distribution.*

Proof. Consider the input to the convolutional layer $\mathbf{X}$ with dimensions $W \times H \times C$, where W, H, and C represent the width, height, and number of channels, respectively. Let $\mathbf{K}_f$ be the

f-th convolutional kernel with dimensions $k \times k \times 1$, and b_f the corresponding bias. The total number of convolutional kernels is M. Assume that the kernel weights are independently and identically distributed (i.i.d.) according to the normal distribution $N(0, \sigma_F^2)$ and that the biases b_f are also i.i.d. following the normal distribution $N(0, \sigma_b^2)$.

The feature map produced by the f-th filter and before applying the ReLU activation function is given by:

$$\begin{aligned}\mathbf{Z}_f(\mathbf{X}) &= b_f\mathbf{1} + \mathbf{K}_f * \mathbf{X} \\ &:= b_f\mathbf{1} + \sum_{c=1}^{C} \mathbf{W}_{f,c} \cdot \mathbf{x}_c,\end{aligned}$$

where $*$ denotes the convolutional operation and $\cdot$ denotes the dot product between matrices and vectors. Here, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_C$ represent the channels of the input image $\mathbf{X}$, each with a flattened size of $W \times H$. The matrix $\mathbf{W}_{f,c}$ denotes the convolutional weight matrix for the f-th filter and the c-th channel, appropriately restructured for the convolution operation as in Section 3.1.1.

Each row of $\mathbf{W}_{f,c}$ corresponds to a specific position of the convolutional filter as it slides over the input image. Specifically, the l-th row of $\mathbf{W}_{f,c}$ represents the filter applied to the l-th patch of the input image, which consists of a sliding window of pixel values covered by the filter. These patches are flattened into vectors of size $k \times k$. For each position (i, j) where the filter is applied, the weights $\mathbf{K}_f[m, n]$ from the filter are placed into the appropriate positions of $\mathbf{W}_{f,c}[l]$, covering the patch that spans rows i to $i + k - 1$ and columns j to $j + k - 1$. The dot product of each row of $\mathbf{W}_{f,c}$ with the corresponding vector from the image channel $\mathbf{x}_c$ represents the application of the filter $\mathbf{K}_f$ to the c-th channel of the input image.

Thus, the elements of each row of $\mathbf{W}_{f,c}$ are 0 where the filter does not apply, and the corresponding element of $\mathbf{K}_f$ where it does, as illustrated below:

Although the filter $\mathbf{K}_f$ has dimensions of $k \times k \times 1$, meaning it is a three-dimensional filter, it applies the same weights across each input channel. To simplify, $\mathbf{K}_f$ is represented as a two-dimensional matrix, focusing only on its spatial dimensions $k \times k$. The l-th row refers to the entire filter block applied to the l-th input patch.

The flattened pixels of the c-th channel of the input image (with 'same' padding) are:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \xrightarrow{\text{Flatten}} \mathbf{x}_c = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix}.$$

The matrix $\mathbf{W}_{f,c}$ is reconstructed from the filter $\mathbf{K}_f$ as follows:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \xrightarrow{\text{Transform}} \mathbf{W}_{f,c} = \begin{bmatrix} A & B & 0 & C & D & 0 & 0 & 0 & 0 \\ 0 & A & B & 0 & C & D & 0 & 0 & 0 \\ 0 & 0 & 0 & A & B & 0 & C & D & 0 \\ 0 & 0 & 0 & 0 & A & B & 0 & C & D \end{bmatrix}.$$

The resulting convolution is represented as:

$$\begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \circ \mathbf{K}_f \\ \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix} \circ \mathbf{K}_f \\ \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \circ \mathbf{K}_f \\ \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix} \circ \mathbf{K}_f \end{bmatrix} \rightarrow \quad \mathbf{W}_{f,c} \cdot \mathbf{x}_c,$$

where $\circ$ denotes the element-wise multiplication (Hadamard product) between the input patch and the filter $\mathbf{K}_f$.

From the matrix multiplication representation, each element in $\mathbf{Z}_f(\mathbf{X})$ can be expressed as:

$$\begin{aligned} \mathbf{Z}_f(\mathbf{X})[j] &= b_f + \sum_{c=1}^{C} \mathbf{W}_{f,c}[j] \cdot \mathbf{x}_c \qquad (j\text{-th row of the weight matrix } \mathbf{W}_{f,c}) \\ &= b_f + \sum_{c=1}^{C} \sum_{i=1}^{HW} \mathbf{W}_{f,c}[j,i] \cdot \mathbf{x}_c[i] \\ &= b_f + \sum_{c=1}^{C} \sum_{l \in \mathcal{P}_j} \mathbf{K}_f[l]\mathbf{x}_c[l], \end{aligned} \tag{3.1}$$

where l is an element of the index set $\mathcal{P}_j$, representing the indices of the j-th patch, which contains k^2 elements corresponding to the size of the convolutional kernel. According to this formulation, since $\mathcal{P}_j$ refers to the same pixels of the input image across different f, and given that the kernel weights and biases are independently and identically distributed (i.i.d.) by definition, the elements $\mathbf{Z}_f[j]$ and $\mathbf{Z}_{f'}[j]$ are also i.i.d. for $f \neq f'$, indicating that the pixels at the same position are independently and identically distributed for different feature maps. Furthermore, while the elements within a single feature map exhibit strong correlations due to shared receptive fields, the feature maps themselves are independent and identically distributed (i.i.d.) conditioned on the data $\mathbf{X}$.

Additionally, $\mathbf{Z}_f[j]$ follows a normal distribution with zero mean and a variance of σ_j^2. The variance σ_j^2 is given by:

$$\begin{aligned}\sigma_j^2 &= \mathrm{Var}(b_f + \sum_{c=1}^{C}\sum_{l\in\mathcal{P}_j}\mathbf{K}_f[l]\mathbf{x}_c[l]) \\ &= \sigma_b^2 + \sum_{c=1}^{C}\sum_{l\in\mathcal{P}_j}\mathbf{x}_c[l]^2\sigma_F^2. \end{aligned} \tag{3.2}$$

As a result, $\mathbf{Z}_f$ is a multivariate Gaussian random variable because it is composed of linear combinations of independent normal variables.

This work employs a stride of 1 and zero padding for simplicity. The resulting feature map size after the convolution operation is $(W-k+1)\times(H-k+1)\times 1$. Typically, the input image is assumed to be square, i.e., $W = H$, which simplifies the feature map size to $(W-k+1)\times(W-k+1)\times 1$. It is important to note that the stride and padding parameters do not affect the outcome of the proof. These parameters are introduced merely to complete the definition of the convolutional layer within the proof without influencing the underlying mathematical results.

After the convolutional layer, the next step is to apply the ReLU activation function, which introduces non-linearity into the network. The ReLU activation is applied element-wise to the feature map $\mathbf{Z}_f$, producing the output:

$$\begin{aligned}\mathbf{Y}_f(\mathbf{X}) &= \phi(\mathbf{Z}_f(\mathbf{X})) \\ &= \max(0, b_f\mathbf{1} + \sum_{c=1}^{C}\mathbf{W}_{f,c}\cdot\mathbf{x}_c).\end{aligned}$$

This activation function sets all negative values in $\mathbf{Z}_f$ to zero while keeping positive values unchanged.

In this work, the analysis focuses on the distribution of a single element O_k of the output vector from the fully connected (FC) layer. The weight matrix $\mathbf{V}$ in the FC layer, which is assumed to consist of independent random variables with a mean of 0 and variance $\frac{1}{M}$ for this analysis, can be partitioned into M parts:

$$\mathbf{V} = [\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_M],$$

where $\mathbf{V}_f$ represents the f-th part (in columns) of the matrix. Each part $\mathbf{V}_f$ corresponds to the response $\mathbf{Y}_f$ from the f-th filter of the previous convolutional layer. Let $\mathbf{v}_{f,k}$ denote the k-th row of the f-th part of the weight matrix $\mathbf{V}_f$.

The single output element $O_k(\mathbf{X})$ can then be expressed as:

$$O_k(\mathbf{X}) = d_k + \sum_{f=1}^{M}(\mathbf{v}_{f,k}\cdot\mathbf{Y}_f(\mathbf{X})),$$

where $\cdot$ denotes the dot product between vectors. Here, $\mathbf{Y}_f(\mathbf{X})$ denotes the flattened response of the ReLU activation function for the f-th filter applied to the input $\mathbf{X}$, and $\mathbf{v}_{f,k}$ and d_k represent the corresponding weight vector and bias term for the k-th output element in the fully connected layer, respectively. Here, d_k is a zero-mean Gaussian random variable with standard deviation σ_d, i.e., $d_k \sim \mathcal{N}(0, \sigma_d^2)$.

$O_k(\mathbf{X})$ is a scalar, representing one specific element of the output vector. The variables within each $\mathbf{Y}_f(\mathbf{X})$ are correlated due to the overlapping receptive fields that share common input regions. Although the ReLU activation function is applied element-wise and introduces non-linearity, it does not eliminate all correlations. The outputs across different filters are independent, given that the convolutional kernels are independently and identically distributed (i.i.d.).

To simplify the expression, define $O_{f,k}(\mathbf{X})$ as the dot product of the weight vector $\mathbf{v}_{f,k}$ and the flattened response $\mathbf{Y}_f(\mathbf{X})$:

$$O_{f,k}(\mathbf{X}) = \mathbf{v}_{f,k} \cdot \mathbf{Y}_f(\mathbf{X}). \tag{3.3}$$

Given that $\mathbf{v}_{f,k}$ and $\mathbf{Y}_f(\mathbf{X})$ are vectors, their dot product can be expanded as:

$$\mathbf{v}_{f,k} \cdot \mathbf{Y}_f(\mathbf{X}) = \sum_{j=1}^{N} \mathbf{v}_{f,k}[j]\mathbf{Y}_f(\mathbf{X})[j], \tag{3.4}$$

where $\mathbf{v}_{f,k}[j]$ is the j-th element of the vector $\mathbf{v}_{f,k}$, and $\mathbf{Y}_f(\mathbf{X})[j]$ is the j-th element of the vector $\mathbf{Y}_f(\mathbf{X})$, with $N = (W - k + 1)^2$ (where N is the size of the feature map, assuming zero padding and a stride of 1).

Equation 3.3 can then be expressed as:

$$O_{f,k}(\mathbf{X}) = \sum_{j=1}^{N} \mathbf{v}_{f,k}[j]\mathbf{Y}_f(\mathbf{X})[j]. \tag{3.5}$$

Here, $O_{f,k}$ represents the combined response of the f-th filter contributing to the k-th output element. Given that $\mathbf{v}_{f,k}[j]$ are i.i.d. variables and $\mathbf{Y}_f(\mathbf{X})[j]$ are also i.i.d. across different f (since $\mathbf{Y}_f(\mathbf{X})[j]$ and $\mathbf{Y}_{f'}(\mathbf{X})[j]$ are i.i.d. for $f \neq f'$), the variables $O_{f,k}$ are i.i.d. across different f. This independence arises because the products $\mathbf{v}_{f,k}[j]\mathbf{Y}_f(\mathbf{X})[j]$ are independent for different f. Furthermore, since both $\mathbf{v}_{f,k}[j]$ and $\mathbf{Y}_f(\mathbf{X})[j]$ share identical distributions across f, each $O_{f,k}$ follows the same distribution.

Therefore, the specific output element $O_k(\mathbf{X})$ can be viewed as the sum of these independently and identically distributed variables, plus the bias term:

$$\begin{aligned} O_k(\mathbf{X}) &= d_k + \sum_{f=1}^{M} \sum_{j=1}^{N} \mathbf{v}_{f,k}[j]\mathbf{Y}_f(\mathbf{X})[j] \\ &= d_k + \sum_{f=1}^{M} O_{f,k}(\mathbf{X}). \end{aligned} \tag{3.6}$$

For simplicity, in the following, I denote $\mathbf{Y}_f(\mathbf{X})[j]$ as $\mathbf{Y}_f[j]$, $\mu_{f,k}(\mathbf{X})$ as $\mu_{f,k}$ and $\sigma^2_{f,k}(\mathbf{X})$ as $\sigma^2_{f,k}$.

The expected value of $O_{f,k}$ can be expressed as:

$$\begin{aligned}\mu_{f,k} &= \mathbb{E}_O[O_{f,k}] \\ &= \mathbb{E}_{\mathbf{v},\mathbf{Y}_f}\left[\sum_{j=1}^{N} \mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]\right].\end{aligned}$$

Since the expectation operator is linear, this can be rewritten as:

$$\mu_{f,k} = \sum_{j=1}^{N} \mathbb{E}_{\mathbf{v},\mathbf{Y}_f}[\mathbf{v}_{f,i}[j]\mathbf{Y}_f[j]]. \tag{3.7}$$

Assuming that $\mathbf{v}_{f,i}[j]$ and $\mathbf{Y}_f[j]$ are independent, then the expectation of their product is the product of their expectations:

$$\mu_{f,k} = \sum_{j=1}^{N} \mathbb{E}_{\mathbf{v}}[\mathbf{v}_{f,k}[j]]\mathbb{E}_{\mathbf{Y}_f}[\mathbf{Y}_f[j]] = 0. \tag{3.8}$$

$\mu_{f,k}$ is 0 because $\mathbb{E}[\mathbf{v}_{f,k}[j]] = 0$.

Now, consider the variance of $O_{f,k}$, which is given by:

$$\begin{aligned}\sigma^2_{f,k} &= \mathrm{Var}(O_{f,k}) \\ &= \mathrm{Var}\left(\sum_{j=1}^{N} \mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]\right) \\ &= \sum_{j=1}^{N} \mathrm{Var}(\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]) + 2\sum_{i\le j}\mathrm{Cov}(\mathbf{v}_{f,k}[i]\mathbf{Y}_f[i], \mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]).\end{aligned} \tag{3.9}$$

Assuming that $\mathbf{v}_{f,k}[j]$ and $\mathbf{Y}_f[j]$ are independent, then the variance of their product can be expressed as:

$$\begin{aligned}\mathrm{Var}(\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]) &= \mathrm{Var}(\mathbf{v}_{f,k}[j])\mathrm{Var}(\mathbf{Y}_f[j]) + \mathrm{Var}(\mathbf{v}_{f,k}[j])\mathbb{E}^2[\mathbf{Y}_f[j]] \\ &\quad + \mathbb{E}^2[\mathbf{v}_{f,k}[j]]\mathrm{Var}(\mathbf{Y}_f[j]) \\ &= \frac{1}{M}\mathrm{Var}(\mathbf{Y}_f[j]) + \frac{1}{M}\mathbb{E}^2[\mathbf{Y}_f[j]] + 0 \times \mathrm{Var}(\mathbf{Y}_f[j]) \\ &= \frac{1}{M}\left(\mathrm{Var}(\mathbf{Y}_f[j]) + \mathbb{E}^2[\mathbf{Y}_f[j]]\right).\end{aligned} \tag{3.10}$$

The Covariance term in the Eq. 3.9 is:

$$\begin{aligned}
&\sum_{i\leq j}\mathrm{Cov}(\mathbf{v}_{f,k}[i]\mathbf{Y}_f[i],\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j])\\
&\quad=\sum_{i\leq j}\mathbb{E}[(\mathbf{v}_{f,k}[i]\mathbf{Y}_f[i]-\mathbb{E}[\mathbf{v}_{f,k}[i]\mathbf{Y}_f[i]])(\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]-\mathbb{E}[\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]])]\\
&\quad=\sum_{i\leq j}\left(\mathbb{E}[(\mathbf{v}_{f,k}[i]\mathbf{Y}_f[i])(\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j])]-\mathbb{E}[\mathbf{v}_{f,k}[i]\mathbf{Y}_f[i]]\mathbb{E}[\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]]\right)\\
&\quad=\sum_{i\leq j}\mathbb{E}_{\mathbf{v},\mathbf{Y}_f}[(\mathbf{v}_{f,k}[i]\mathbf{Y}_f[i])(\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j])]\\
&\quad=\sum_{i\leq j}\mathbb{E}_{\mathbf{v}}[\mathbf{v}_{f,k}[i]\mathbf{v}_{f,k}[j]]\mathbb{E}_{\mathbf{Y}_f}[\mathbf{Y}_f[i]\mathbf{Y}_f[j]] \qquad (\mathbf{v}_{f,k}\text{ and }\mathbf{Y}_f\text{ are independent})\\
&\quad=\sum_{i\leq j}\mathbb{E}_{\mathbf{v}}[\mathbf{v}_{f,k}[i]]\mathbb{E}_{\mathbf{v}}[\mathbf{v}_{f,k}[j]]\mathbb{E}_{\mathbf{Y}_f}[\mathbf{Y}_f[i]\mathbf{Y}_f[j]] \qquad (\mathbf{v}_{f,k}\text{ are independent variables})\\
&\quad=0. \qquad (\text{the mean of }\mathbf{v}_{f,k}\text{ is 0}) \qquad (3.11)
\end{aligned}$$

By substituting Eq. 3.10 and Eq. 3.11 into Eq. 3.9, the variance $\sigma^2_{f,k}$ can be expressed as:

$$\begin{aligned}
\sigma^2_{f,k}&=\frac{1}{M}\sum_{j=1}^{N}\left(\mathrm{Var}(\mathbf{Y}_f[j])+\mathbb{E}^2[\mathbf{Y}_f[j]]\right)\\
&=\frac{1}{M}\sum_{j=1}^{N}\mathbb{E}[\mathbf{Y}_f^2[j]] \qquad (3.12)\\
&=\frac{1}{M}\sum_{j=1}^{N}\frac{1}{2}\mathrm{Var}(\mathbf{Z}_f[j]) \qquad (3.13)\\
&=\frac{1}{2M}\sum_{j=1}^{N}(\sigma_b^2+\sum_{c=1}^{C}\sum_{l\in\mathcal{P}_j}\mathbf{x}_c[l]^2\sigma_F^2) \qquad (\text{substituting Eq. 3.2})\\
&=\frac{1}{2M}(N\sigma_b^2+\sum_{j=1}^{N}\sum_{c=1}^{C}\sum_{l\in\mathcal{P}_j}\mathbf{x}_c[l]^2\sigma_F^2)\\
&\leq\frac{1}{2M}(W-k+1)^2(\sigma_b^2+Ck^2x_{max}^2\sigma_F^2). \qquad (3.14)
\end{aligned}$$

Here, W and k represent the width of the network's input image and the size of the convolutional kernel, respectively. $|\mathcal{P}_j|=k^2$ denotes the size of the input patch. Let $x_{\max}$ be the maximum value of the input image pixels across all channels. Therefore, according to Eq. 3.14, $\sigma^2_{f,k}$ is finite.

As $\mathbf{Z}_f[j]$ is symmetric around zero, the following holds:

$$\begin{aligned}\mathrm{Var}[\mathbf{Z}_f[j]] &= \mathbb{E}[\mathbf{Z}_f[j]^2] - \mathbb{E}[\mathbf{Z}_f[j]]^2 \\ &= \mathbb{E}[\mathbf{Z}_f[j]^2] \\ &= \mathbb{E}[I(\mathbf{Z}_f[j] < 0)\mathbf{Z}_f[j]^2] + \mathbb{E}[I(\mathbf{Z}_f[j] > 0)\mathbf{Z}_f[j]^2] \\ &= 2\mathbb{E}[I(\mathbf{Z}_f[j] > 0)\mathbf{Z}_f[j]^2].\end{aligned}$$

Thus,

$$\begin{aligned}\mathbb{E}[\mathbf{Y}_f[j]^2] &= \mathbb{E}[\max(0, \mathbf{Z}_f[j])^2] \\ &= \mathbb{E}[I(\mathbf{Z}_f[j] > 0)\mathbf{Z}_f[j]^2] \\ &= \frac{1}{2}\mathbb{E}[\mathbf{Z}_f[j]^2] \\ &= \frac{1}{2}\mathrm{Var}[\mathbf{Z}_f[j]]. \end{aligned} \tag{3.15}$$

Eq. 3.15 explains why Eq. 3.13 is satisfied, as the variance of $\mathbf{Z}_f[j]$ directly contributes to the expected value of $\mathbf{Y}_f[j]^2$, with the ReLU activation preserving half of the variance.

Consequently, $O_{f,k}(\mathbf{X})$, for $f = 1, ..., M$ (as defined in Eq. 3.5), is a sequence of independently and identically distributed random variables, each with finite (0 in this case) expected value $\mathbb{E}(O_{f,k}(\mathbf{X})) = \mu_{f,k}$ (Eq. 3.8) and finite variance $\mathrm{Var}(O_{f,k}(\mathbf{X})) = \sigma^2_{f,k}$ (Eq. 3.14). By Lindeberg–Lévy central limit theorem (Billingsley 1995, p. 357), the random variables $\sqrt{M}(\frac{\sum_{f=1}^{M} O_{f,k}(\mathbf{X})}{M} - \mathbb{E}(O_{f,k}(\mathbf{X})))$ converge in distribution to a normal random variable $\mathcal{N}(0, \sigma_k^2)$ as M approaches infinity:

$$\sqrt{M}(\frac{\sum_{f=1}^{M} O_{f,k}(\mathbf{X})}{M} - \mathbb{E}(O_{f,k}(\mathbf{X}))) \xrightarrow{d} \mathcal{N}(0, \sigma_k^2).$$

Since the output $O_k(\mathbf{X}) = d_k + \sum_{f=1}^{M} O_{f,k}(\mathbf{X})$ (Eq. 3.6), and $d_k \sim \mathcal{N}(0, \sigma_d^2)$, the distribution of $O_k(\mathbf{X})$ will be:

$$O_k(\mathbf{X}) \sim \mathcal{N}\left(0, M\sigma_k^2 + \sigma_d^2\right) = \mathcal{N}\left(0, \sigma_d^2 + \frac{1}{2}(N\sigma_b^2 + \sigma_F^2 \sum_{j=1}^{N} \sum_{c=1}^{C} \sum_{l \in \mathcal{P}_j} \mathbf{x}_c[l]^2)\right). \tag{3.16}$$

Eq 3.16 shows that as the number of kernels M tends to infinity, the output of the CNN will converge to a normal distribution. In this context, N represents the dimension of the feature map, given by $(W - k + 1)^2$, where W is the width of the input, and k is the size of the convolutional kernel. The parameter C denotes the number of input channels, and $\mathcal{P}_j$ represents the indices of the j-th patch, which contains k^2 elements corresponding to the size of the convolutional kernel. This formulation illustrates that a single pixel j in the feature map is influenced by the input pixels within the patch $\mathcal{P}_j$. Additionally, σ_b^2, σ_F^2, and σ_d^2 represent the variances of the biases in the convolutional layer, the kernel weights in the convolutional layer, and the biases in the fully-connected layer, respectively. □

Note that any two output elements $O_{k_1}(\mathbf{X})$ and $O_{k_2}(\mathbf{X})$ for $k_1 \neq k_2$ are jointly Gaussian and have zero covariance. This is because the weights $\mathbf{v}_{f,k}$ are independent and have zero mean, eliminating any potential correlations that could arise through the shared random vector $\mathbf{Y}_f$. The bias terms are also independent, ensuring that the covariances between biases are zero for different output elements. Consequently, the output elements remain independent despite both being derived from the same features produced by the hidden layer (specifically, the feature map $\mathbf{Y}_f(\mathbf{X})$).

Furthermore, these output elements are identically distributed because the weights $\mathbf{v}_{f,k}$ and biases d_k are independently and identically distributed (i.i.d.). From Eq. 3.16, it is evident that for different output elements k and a given input, the elements of the CNN's output, as defined in the lemma, converge to the same normal distribution, as the only variable parameter, $\mathbf{X}$, remains fixed.

Output Distribution of infinitely wide FNN

As detailed in Section 3.1.1, the CNN can be reinterpreted in the form of an FNN as follows:

$$\mathbf{Y}_j(\mathbf{X}) = \phi\left(b_j + \sum_{i=1}^{d_{\text{in}}} \mathbf{Q}_{ji}\mathbf{X}_i^{\text{flat}}\right), \tag{3.17}$$

$$O_k(\mathbf{X}) = d_k + \sum_{j=1}^{N} \mathbf{V}_{kj}\mathbf{Y}_j(\mathbf{X}), \tag{3.18}$$

where $N = MW_{\text{out}}H_{\text{out}}$ and $d_{\text{in}} = CWH$.

Building upon the analysis in Lemma 1, by setting the variance of the weight matrix $\mathbf{V}$ to σ_v^2/N, similarly to how the weights in the fully connected (FC) layer are handled in 1, this constraint is necessary to maintain a constant variance in the final output. As the number of hidden units N approaches infinity, the output element of an FNN, which mirrors the structure of a single hidden layer with a ReLU activation function followed by a fully connected layer, converges to a normal distribution as follows:

$$O_k(\mathbf{X}) \sim \mathcal{N}\left(0, \sigma_d^2 + \frac{\sigma_v^2}{2}(\sigma_b^2 + \sigma_Q^2 \sum_{i=1}^{d_{\text{in}}} (\mathbf{X}_i^{\text{flat}})^2)\right). \tag{3.19}$$

For the FNN, σ_Q^2 denotes the variance of elements in the matrix Q, which is fully populated. This contrasts with the CNN scenario, where the elements manifest a block-structured pattern.

Moreover, when weights and biases are considered independent and identically distributed variables, any two output elements for the same input image are jointly Gaussian and are independently and identically normally distributed.

3.2.2 The Generality of the Lemma

This subsection discusses the broader applicability of the Lemma 1, focusing on the conditions under which the output distribution of the one-layer CNN converges to a normal distribution. The lemma is general enough to apply to networks with identical and non-identical convolutional kernels, provided certain conditions are met. These conditions include the independence and identical distribution (i.i.d.) of weights and biases for each kernel, as well as the boundedness of the activation function, which ensures finite variances and expected values. In cases where kernels are non-identically distributed, the Lyapunov Central Limit Theorem (LCLT) (Billingsley 1995, p. 362) can be invoked, ensuring convergence under appropriate moment conditions.

Identical kernels

To establish the output distribution property of the CNN as defined in the lemma 1, it is essential to verify that the conditions of Lindeberg–Lévy central limit theorem (Billingsley 1995, p. 357) are satisfied: the variables (defined in Eq. 3.3) must be independent and identically distributed and have finite expected values (Eq. 3.8) and finite variances (Eq. 3.14).

It can be observed from the proof process that the lemma holds more generally for any bounded activation function and any prior distribution on convolutional weights and biases, provided that the weights and biases for each kernel are independent and identically distributed (i.i.d.). The i.i.d. property of the weights and biases for each convolutional filter ensures that the feature maps of the convolutional layer are independent from one another. Similarly, the weights of the fully connected layer are also i.i.d. variables, which implies that the variables considered in the proof, $O_{f,k}(\mathbf{X}) = v_{f,k} \cdot Y_f(\mathbf{X})$ (Eq. 3.3), are also independent and identically distributed.

The expectation of the variables $O_{f,k}(\mathbf{X})$ is unaffected by the feature map variables, as the mean of the fully connected layer's weights is zero. Regarding the variance of $O_{f,k}(\mathbf{X})$, as demonstrated in the deduction from Eq. 3.9 to Eq. 3.12, the bounded activation function and the independence of the convolutional layer weights ensure that the variance is finite. Eq. 3.12 is derived from the independence of the weights and the fact that the mean of the fully connected layer's weights is zero. Since the activation function is bounded, its second moment $\mathbb{E}[X^2]$ is also bounded. Consequently, given that the summation involves N terms (where N represents the number of input image pixels), the variance of $O_{f,k}(\mathbf{X})$ remains finite, as indicated by Eq. 3.12.

The result extends to cases where the prior distribution for the fully connected layer's weights has a zero mean and finite variance that decreases with M, specifically, being proportional to the inverse square root of M. This constraint is necessary to maintain the final output variance constant. Therefore, if the variance of the weights in the convolutional layer decreases with the inverse of M, that is, $\sigma^2 = \sigma_F^2/M$, then the result can also be applied to the prior distribution for the fully connected layer's weights with zero mean and finite variance.

Non-identical kernels

Furthermore, if the kernel weights within each filter are independent and identically distributed but vary across different filters, the variables $O_{f,k}$ are independent across filters f but not identically distributed, setting that $\mathbf{K}_f \sim \mathcal{N}(0, \sigma_f^2)$. In this scenario, the Lyapunov Central Limit Theorem is applicable. Therefore, in addition to ensuring the finiteness of the mean and variance of $O_{f,k}$ (as derived from the expected value and variance, with the independence properties fulfilling these conditions), Lyapunov's condition must also be satisfied. Additionally, it is necessary for the $\delta + 1$ moment of the weights in the fully connected layer, when multiplied by M, to decrease as M approaches infinity. In the subsequent proof, δ is set to 1.

The Lyapunov's condition (Billingsley 1995, p. 362) is proven as follows:

For some $\delta > 0$:

$$\lim_{M\to\infty} R(M,\delta) := \lim_{M\to\infty} \frac{1}{s_M^{2+\delta}} \sum_{f=1}^{M} \mathbb{E}\left[|O_{f,k} - \mu_{f,k}|^{2+\delta}\right] = 0,$$

where

$$s_M^2 = \sum_{f=1}^{M} \sigma_{f,k}^2.$$

From Eq. 3.2 and Eq. 3.13, it follows:

$$\begin{aligned}\sigma_{f,k}^2 &= \frac{1}{M}\sum_{j=1}^{N}\frac{1}{2}\mathrm{Var}(\mathbf{Z}_f[j]) \\ &= \frac{1}{2M}\sum_{j=1}^{N}(\sigma_b^2 + \sum_{c=1}^{C}\sum_{l\in\mathcal{P}_j}\mathbf{x}_c[l]^2\sigma_f^2). \qquad (3.20)\end{aligned}$$

For $\delta = 1$, by substituting Eq. 3.3, Eq. 3.4 and Eq. 3.8 into the expectation term of $R(M,1)$, the following is obtained:

$$\begin{aligned}\sum_{f=1}^{M}\mathbb{E}\left[|O_{f,k} - \mu_{f,k}|^3\right] &= \sum_{f=1}^{M}\mathbb{E}\left[|O_{f,k}|^3\right] \\ &= \sum_{f=1}^{M}\mathbb{E}\left[|\mathbf{v}_{f,k}\cdot\mathbf{Y}_f|^3\right] \\ &= \sum_{f=1}^{M}\mathbb{E}\left[\left|\sum_{j=1}^{N}\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]\right|^3\right] \\ &\leq \sum_{f=1}^{M}\mathbb{E}\left[\sum_{j=1}^{N}|\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]|^3\right]. \qquad (3.21)\end{aligned}$$

By linearity of expectation and the assumption of independence:

$$\begin{aligned}(3.21) &= \sum_{f=1}^{M}\sum_{j=1}^{N}\mathbb{E}\left[\left|\mathbf{v}_{f,k}[j]\mathbf{Y}_f[j]\right|^3\right] \\ &= \sum_{f=1}^{M}\sum_{j=1}^{N}\mathbb{E}\left[\left|\mathbf{v}_{f,k}[j]\right|^3\right]\mathbb{E}\left[\left|\mathbf{Y}_f[j]\right|^3\right].\end{aligned}$$

Thus, $\sum_{f=1}^{M}\mathbb{E}\left[|O_{f,k}-\mu_{f,k}|^3\right]$ is bounded by:

$$\sum_{f=1}^{M}\mathbb{E}\left[|O_{f,k}-\mu_{f,k}|^3\right] \le \sum_{f=1}^{M}\sum_{j=1}^{N}\mathbb{E}\left[\left|\mathbf{v}_{f,k}[j]\right|^3\right]\mathbb{E}\left[\left|\mathbf{Y}_f[j]\right|^3\right]. \tag{3.22}$$

In this case, the variance of variables $\mathbf{Z}_f[j]$ is σ_f^2. If let σ_F^2 be the minimum variance across all filters and $x_{\min}$ be the minimum value of the input image pixels across all channels. According to Eq. 3.20:

$$\sigma_{f,k}^2 \ge \frac{1}{2M}(W-k+1)^2(\sigma_b^2 + Ck^2x_{\min}^2\sigma_f^2).$$

Thus,

$$\begin{aligned}s_M^3 &= (\sum_{f=1}^{M}\sigma_{f,k}^2)^{\frac{3}{2}} \\ &\ge (\sum_{f=1}^{M}\frac{1}{2M}(W-k+1)^2(\sigma_b^2 + Ck^2x_{\min}^2\sigma_f^2))^{\frac{3}{2}} \\ &\ge \frac{1}{2\sqrt{2}}(W-k+1)^3(\sigma_b^2 + Ck^2x_{\min}^2\sigma_F^2)^{\frac{3}{2}}.\end{aligned} \tag{3.23}$$

According to Eq. 3.22 and Eq. 3.23, $R(M,1)$ is bounded by:

$$R(M,1) \le \frac{\sum_{f=1}^{M}\sum_{j=1}^{N}\mathbb{E}\left[\left|\mathbf{v}_{f,k}[j]\right|^3\right]\mathbb{E}\left[\left|\mathbf{Y}_f[j]\right|^3\right]}{\frac{1}{2\sqrt{2}}(W-k+1)^3(\sigma_b^2 + Ck^2x_{\min}^2\sigma_F^2)^{\frac{3}{2}}}. \tag{3.24}$$

The probability density function (PDF) of $\mathbf{Y}_f[j] = \max(0, \mathbf{Z}_f[j])$, where $\mathbf{Z}_f[j]$ follows a normal distribution, is a rectified Gaussian distribution. Given that $\mathbf{Z}_f[j] \sim N(0, \sigma_{Z_f}^2)$, with

$$\sigma_{Z_f}^2 = \sigma_b^2 + \sum_{c=1}^{C}\sum_{l\in P_j}\mathbf{x}_c[l]^2\sigma_f^2.$$

The PDF of a rectified Gaussian distribution is:

$$f(x;0,\sigma^2) = 0.5\delta(x) + \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} U(x),$$

where $\Phi(0) = 0.5$ is the cumulative distribution function (CDF) of the normal distribution evaluated at zero, representing the probability that $\mathbf{Z}_f[j] \leq 0$, $\delta(x)$ is the Dirac delta function, representing the probability mass at $x = 0$, and $U(x)$ is the unit step function ensuring that $f(x;0,\sigma^2)$ is defined for $x \geq 0$.

To compute $\mathbb{E}[|\mathbf{Y}_f[j]|^3]$, the process involves integrating the third power of $\mathbf{Y}_f[j]$ over its PDF. Since $|\mathbf{Y}_f[j]|^3 = \mathbf{Y}_f[j]^3$ for $\mathbf{Y}_f[j] \geq 0$, the expectation can be written as:

$$\mathbb{E}[\mathbf{Y}_f[j]^3] = \int_0^\infty x^3 \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \right) dx.$$

This integral is a standard calculation for the third moment of a rectified Gaussian (or half-normal) distribution. To simplify the integral, use the change of variable $u = \frac{x}{\sigma}$, which gives $x = u\sigma$ and $dx = \sigma du$. Substituting this into the integral:

$$\mathbb{E}[\mathbf{Y}_f[j]^3] = \int_0^\infty (u\sigma)^3 \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(u\sigma)^2}{2\sigma^2}} \right) \sigma du,$$

which simplifies to:

$$\mathbb{E}[\mathbf{Y}_f[j]^3] = \frac{\sigma^4}{\sqrt{2\pi\sigma^2}} \int_0^\infty u^3 e^{-\frac{u^2}{2}} \, du.$$

The remaining integral, $\int_0^\infty u^3 e^{-\frac{u^2}{2}} \, du$, is a known result and evaluates to $\sqrt{\frac{\pi}{2}}$. Therefore, the third moment of $\mathbf{Y}_f[j]$ becomes:

$$\mathbb{E}[\mathbf{Y}_f[j]^3] = \frac{\sigma^4}{\sqrt{2\pi\sigma^2}} \cdot \sqrt{\frac{\pi}{2}} = \frac{4\sigma^3}{\sqrt{2\pi}}.$$

Substituting the variance $\sigma^2 := \sigma^2_{Z_f} = \sigma_b^2 + \sum_{c=1}^{C} \sum_{l \in P_j} \mathbf{x}_c[l]^2 \sigma_f^2$, the final expression for $\mathbb{E}[|\mathbf{Y}_f[j]|^3]$ is:

$$\mathbb{E}[|\mathbf{Y}_f[j]|^3] = \frac{4}{\sqrt{2\pi}} \left(\sigma_b^2 + \sum_{c=1}^{C} \sum_{l \in P_j} \mathbf{x}_c[l]^2 \sigma_f^2 \right)^{\frac{3}{2}}. \tag{3.25}$$

Substituting Eq. 3.25 into Eq. 3.24:

$$R(M,1) \leq \frac{\sum_{f=1}^{M} \sum_{j=1}^{N} \mathbb{E}\left[\left| \mathbf{v}_{f,k}[j] \right|^3 \right] \cdot \frac{4}{\sqrt{2\pi}} \left(\sigma_b^2 + \sum_{c=1}^{C} \sum_{l \in P_j} \mathbf{x}_c[l]^2 \sigma_f^2 \right)^{\frac{3}{2}}}{\frac{1}{2\sqrt{2}} (W-k+1)^3 (\sigma_b^2 + Ck^2 x_{\min}^2 \sigma_F^2)^{\frac{3}{2}}}.$$

Let $\sigma^2_{F_0}$ be the maximum variance across all filters, $x_{\max}$ be the maximum value of the input image pixels across all channels, and the expectation of the third moment of the absolute value of the weights in the FC layer be given by $\frac{1}{M^2}$:

$$\begin{aligned} R(M,1) &\leq \frac{MN \cdot \frac{1}{M^2} \cdot \frac{4}{\sqrt{2\pi}} \left(\sigma_b^2 + Ck^2 x_{\max}^2 \sigma_{F_0}^2\right)^{\frac{3}{2}}}{\frac{1}{2\sqrt{2}}(W-k+1)^3 (\sigma_b^2 + Ck^2 x_{\min}^2 \sigma_F^2)^{\frac{3}{2}}} \\ &= \frac{1}{M} \cdot \frac{8\left(\sigma_b^2 + Ck^2 x_{\max}^2 \sigma_{F_0}^2\right)^{\frac{3}{2}}}{\sqrt{\pi}(W-k+1)(\sigma_b^2 + Ck^2 x_{\min}^2 \sigma_F^2)^{\frac{3}{2}}} \\ &:= \frac{A}{M}. \qquad \text{(A is a constant)} \end{aligned}$$

As $M \to \infty$, the term $\frac{1}{M}$ tends to zero. Therefore, it follows that:

$$\lim_{M\to\infty} R(M,1) \leq \lim_{M\to\infty} \frac{A}{M} = 0,$$

then the sum of $\frac{O_{f,k}-\mu_{f,k}}{s_M}$ converges in distribution to a standard normal random variable, as M goes to infinity:

$$\frac{1}{s_M} \sum_{f=1}^{M} (O_{f,k} - \mu_{f,k}) \xrightarrow{d} \mathcal{N}(0,1).$$

Since the output $O_k = \sum_{f=1}^{M} O_{f,k} + d_k$, and $d_k \sim \mathcal{N}(0, \sigma_d^2)$, the distribution of O_k will be:

$$O_k \sim \mathcal{N}\left(0, s_M^2 + \sigma_d^2\right) = \mathcal{N}\left(0, \sigma_d^2 + \frac{1}{2}N\sigma_b^2 + \frac{1}{2M} \sum_{f=1}^{M} \sum_{j=1}^{N} \sum_{c=1}^{C} \sum_{l \in \mathcal{P}_j} \mathbf{x}_c[l]^2 \sigma_f^2 \right). \tag{3.26}$$

Eq 3.26 shows that as the number of kernels M tends to infinity, the output of the CNN will converge to a normal distribution.

From the proof process above, it is evident that the result extends to cases where the prior distribution of convolutional weights and biases is independent across filters, and the activation function is bounded. This prior constraint ensures that $\mathbb{E}\left[\left|\mathbf{Y}_f[j]\right|^3\right]$ in Eq. 3.24 is finite. Therefore, the computation of $\mathbb{E}\left[\left|\mathbf{Y}_f[j]\right|^3\right]$ is no longer necessary in this prior case.

3.2.3 Infinitely wide NNs as Gaussian Process

The previous section demonstrated that as the width of the convolutional layer in a CNN or the fully connected layer in an FNN approaches infinity, each output element for a given input converges to an identical normal distribution.

In this section, I explore the distribution of a single output element when evaluated on different input images, such as $\mathbf{X}$ and $\mathbf{X}'$, under the same conditions as in Lemma 1. I show that the corresponding output elements follow a multivariate Gaussian joint distribution, which can be expressed as:

$$O_k(\mathbf{X}, \mathbf{X}') = \begin{pmatrix} O_k(\mathbf{X}) \\ O_k(\mathbf{X}') \end{pmatrix}.$$

This formulation and subsequent reasoning can be extended to the case of infinitely many input points.

Define $\mathbf{Z}_f(\mathbf{X}, \mathbf{X}')$ as:

$$\begin{aligned} \mathbf{Z}_f(\mathbf{X}, \mathbf{X}') &= \begin{pmatrix} \mathbf{Z}_f(\mathbf{X}) \\ \mathbf{Z}_f(\mathbf{X}') \end{pmatrix} \\ &= b_f \mathbf{1} + \sum_{c=1}^{C} \begin{pmatrix} \mathbf{W}_{f,c} & 0 \\ 0 & \mathbf{W}_{f,c} \end{pmatrix} \begin{pmatrix} \mathbf{x}_c \\ \mathbf{x}'_c \end{pmatrix}, \end{aligned}$$

where $\mathbf{1}$ is a vector of ones, and $\mathbf{x}_c$ and $\mathbf{x}'_c$ for $c = 1, \ldots, C$ are the flattened row vectors corresponding to the c-th channel of the input images $\mathbf{X}$ and $\mathbf{X}'$, respectively, each of size WH. As in Lemma 1, the weights $\mathbf{W}_{f,c}$ and biases b_f are independent and identically normally distributed.

For any pair of data points $\mathbf{X}$ and $\mathbf{X}'$, the feature maps $\mathbf{Z}_f(\mathbf{X}, \mathbf{X}')$ have a multivariate Gaussian joint distribution. This is because each element of $\mathbf{Z}_f(\mathbf{X}, \mathbf{X}')$ is a linear combination of independent Gaussian random variables: the biases b_f and the filters $\mathbf{W}_{f,c}$.

While the elements within a feature map may display strong correlations, different feature maps are independent and identically distributed (i.i.d.) conditioned on the data (i.e., $\mathbf{Z}_f(\mathbf{X}, \mathbf{X}')$ and $\mathbf{Z}_{f'}(\mathbf{X}, \mathbf{X}')$ are i.i.d. for $f \neq f'$), because the parameters for different feature maps (i.e., the biases b_f and the filters $\mathbf{W}_{f,c}$) are themselves i.i.d.

Consider the following expression for $O_k(\mathbf{X}, \mathbf{X}')$:

$$\begin{aligned} O_k(\mathbf{X}, \mathbf{X}') &= \begin{pmatrix} O_k(\mathbf{X}) \\ O_k(\mathbf{X}') \end{pmatrix} \\ &= \begin{pmatrix} d_k + \sum_{f=1}^{M} O_{f,k}(\mathbf{X}) \\ d_k + \sum_{f=1}^{M} O_{f,k}(\mathbf{X}') \end{pmatrix} \\ &= d_k \mathbf{1} + \sum_{f=1}^{M} \begin{pmatrix} O_{f,k}(\mathbf{X}) \\ O_{f,k}(\mathbf{X}') \end{pmatrix} \\ &= d_k \mathbf{1} + \sum_{f=1}^{M} \begin{pmatrix} v_{f,k} & 0 \\ 0 & v_{f,k} \end{pmatrix} \begin{pmatrix} \mathbf{Y}_f(\mathbf{X}) \\ \mathbf{Y}_f(\mathbf{X}') \end{pmatrix} \\ &= d_k \mathbf{1} + \sum_{f=1}^{M} \begin{pmatrix} v_{f,k} & 0 \\ 0 & v_{f,k} \end{pmatrix} \phi(\mathbf{Z}_f(\mathbf{X}, \mathbf{X}')), \end{aligned} \tag{3.27}$$

where ϕ is the ReLU activation function, d_k is the bias term for the k-th output node, $v_{f,k}$ are the weights connecting the f-th feature map to the k-th output, and $Y_f(\mathbf{X}) = \phi(\mathbf{Z}_f(\mathbf{X}))$.

The first bias term $d_k\mathbf{1}$ is itself i.i.d. Gaussian by assumption. As the number of filters M goes to infinity, the second term in Eq. 3.27 becomes a sum of infinitely many i.i.d. terms. Specifically, the $\mathbf{Z}_f(\mathbf{X}, \mathbf{X}')$ are i.i.d. by assumption, thus $\phi(\mathbf{Z}_f(\mathbf{X}, \mathbf{X}'))$ are i.i.d., and the weights $v_{f,k}$ are i.i.d. by definition. By applying the multivariate Central Limit Theorem, this sum converges to a multivariate normal distribution. Therefore, the random variable $O_k(\mathbf{X}, \mathbf{X}')$ converges to a multivariate Gaussian random variable in the limit as $M \to \infty$.

Furthermore, the random vectors $O_k(\mathbf{X}, \mathbf{X}')$ are i.i.d.; that is, $O_k(\mathbf{X}, \mathbf{X}')$ and $O_{k'}(\mathbf{X}, \mathbf{X}')$ are i.i.d. for $k \neq k'$. They are identically distributed because d_k and $v_{f,k}$ are i.i.d., and $\phi(\mathbf{Z}_f(\mathbf{X}, \mathbf{X}'))$ is shared across different k. To show that they are independent, note that $O_k(\mathbf{X}, \mathbf{X}')$ and $O_{k'}(\mathbf{X}, \mathbf{X}')$ are jointly Gaussian, so it suffices to show that they are uncorrelated. Since the weights $v_{f,k}$ have zero means and are independent, any potential correlations arising through the shared random vector $\phi(\mathbf{Z}_f(\mathbf{X}, \mathbf{X}'))$ are eliminated. Therefore, the outputs are independent.

Gaussian Process

As previously discussed, any finite set of random variables $\{O_k(\mathbf{X}_1), \ldots, O_k(\mathbf{X}_n)\}$ will follow a joint multivariate Gaussian distribution, consistent with the definition of a Gaussian process. Therefore, we conclude that $O_k \sim \mathcal{GP}(\mu, K)$, a GP with mean μ and covariance K, which are themselves independent of k. Given that the parameters have a zero mean, we have $\mu(\mathbf{X}) = \mathbb{E}[O_k(\mathbf{X})] = 0$.

To compute the mean and covariance, it is easiest to express the convolutional layer in the network entirely in index notation:

$$\begin{aligned} A_{f,j}(\mathbf{X}) &= \mathbf{Z}_f(\mathbf{X})[j] \\ &= b_f + \sum_{c=1}^{C} \sum_{i=1}^{HW} W_{f,c,j,i} \mathbf{x}_{c,i}, \qquad \text{(Substituting Eq. 3.1)} \end{aligned}$$

where $f \in \{1, \ldots, M\}$ represents the convolution kernels, $c \in \{1, \ldots, C\}$ denotes the input channels, and $j \in \{1, \ldots, (H-k+1)(W-k+1)\}$ indicates the location within the feature maps, while $i \in \{1, \ldots, HW\}$ represents the location within the input channel.

The mean function is then straightforward to compute:

$$\mathbb{E}\left[A_{f,j}(\mathbf{X})\right] = \mathbb{E}\left[b_f\right] + \sum_{c=1}^{C} \sum_{i=1}^{HW} \mathbb{E}\left[W_{f,c,j,i} \mathbf{x}_{c,i}\right] = 0,$$

since b_f and $W_{f,c,j,i}$ are zero-mean random variables, and $W_{f,c,j,i}$ is independent of the input pixels $\mathbf{X}$.

Formally, as the convolutional layer activations are composed of a sum of terms, their covariance is the sum of the covariances of all those underlying terms,

$$\mathbb{C}[A_{f,j}(\mathbf{X}), A_{f,j}(\mathbf{X}')] = \mathbb{V}[b_f] + \sum_{c=1}^{C} \sum_{c'=1}^{C} \sum_{i=1}^{HW} \sum_{i'=1}^{HW} \mathbb{C}[W_{f,c,j,i} \mathbf{x}_{c,i}, W_{f,c',j,i'} \mathbf{x}'_{c',i'}].$$

As the terms in the covariance have zero mean, and as the weights and inputs are independent,

$$\mathbb{C}[A_{f,j}(\mathbf{X}), A_{f,j}(\mathbf{X}')] = \sigma_b^2 + \sum_{c=1}^{C}\sum_{c'=1}^{C}\sum_{i=1}^{HW}\sum_{i'=1}^{HW} \mathbb{E}[W_{f,c,j,i}W_{f,c',j,i'}]\mathbb{E}[\mathbf{x}_{c,i}\mathbf{x}'_{c',i'}].$$

The weights are independent for different channels: $W_{f,c}$ and $W_{f,c'}$ are i.i.d. for $c \neq c'$, therefore, $\mathbb{E}\left[W_{f,c,j,i}W_{f,c',j,i'}\right] = 0$ for $c \neq c'$. Further, each row j of the weight matrices $W_{f,c,j,i}$ only contains independent variables or zeros (see $\mathbf{W}_{f,c}$ in Section 3.2.1), so $\mathbb{E}[W_{f,c,j,i}W_{f,c',j,i'}] = 0$ for $i \neq i'$. Thus, we can eliminate the sums over c' and i':

$$\mathbb{C}[A_{f,j}(\mathbf{X}), A_{f,j}(\mathbf{X}')] = \sigma_b^2 + \sum_{c=1}^{C}\sum_{i=1}^{HW} \mathbb{E}[W_{f,c,j,i}W_{f,c,j,i}]\mathbb{E}[\mathbf{x}_{c,i}\mathbf{x}'_{c,i}].$$

The j-th row of $W_{f,c,j,i}$ is zero for indices i that do not belong to its convolutional patch so that we can restrict the sum over i to that region. The covariance of the convolutional layer is:

$$K_j^0(\mathbf{X}, \mathbf{X}') = \mathbb{C}\left[A_{f,j}(\mathbf{X}), A_{f,j}(\mathbf{X}')\right] = \sigma_b^2 + \sigma_F^2 \sum_{c=1}^{C} \sum_{i \in j\text{th patch}} \mathbf{X}_{c,i}\mathbf{X}'_{c,i}.$$

The covariance function $K(\mathbf{X}, \mathbf{X}')$ can then be expressed as:

$$\begin{aligned} K(\mathbf{X}, \mathbf{X}') &= \mathbb{C}[O_k(\mathbf{X})O_k(\mathbf{X}')] \\ &= \mathbb{V}[d_k] + \sum_{f=1}^{M} \sum_{j=1}^{(H-k+1)^2} \mathbb{C}[\mathbf{V}_{k,f}\phi(A_{f,j}(\mathbf{X})), \mathbf{V}_{k,f}\phi(A_{f,j}(\mathbf{X}'))]. \end{aligned}$$

Since expectations of the terms in the covariance are zero and the weights and inputs are independent, assuming $H = M$ (which holds in most cases) and denoting the variance of the weights $\mathbf{V}$ as σ_v^2, the covariance becomes:

$$\begin{aligned} K(\mathbf{X}, \mathbf{X}') &= \sigma_d^2 + \sum_{f=1}^{M}\sum_{f'=1}^{M}\sum_{j=1}^{(H-k+1)^2}\sum_{j'=1}^{(H-k+1)^2} \mathbb{E}[\mathbf{V}_{k,f'}\mathbf{V}_{k,f'}]\mathbb{E}[\phi(A_{f,j}(\mathbf{X}))\phi(A_{f',j'}(\mathbf{X}'))] \\ &= \sigma_d^2 + \sigma_v^2 \sum_{f=1}^{M}\sum_{j=1}^{(H-k+1)^2} \mathbb{E}[\phi(A_{f,j}(\mathbf{X}))\phi(A_{f,j}(\mathbf{X}'))]. \end{aligned}$$

For the ReLU nonlinearity ($\phi(x) = \max(0, x)$), from the paper (Garriga-Alonso et al. 2019), we have:

$$\mathbb{E}[\phi(A_{f,j}(\mathbf{X}))\phi(A_{f,j}(\mathbf{X}'))] = \frac{\sqrt{K_j^0(\mathbf{X}, \mathbf{X})K_j^0(\mathbf{X}', \mathbf{X}')}}{\pi}\left(\sin\theta_j + (\pi - \theta_j)\cos\theta_j\right),$$

where $\theta_j = \cos^{-1}\left(K_j^0(\mathbf{X},\mathbf{X}')/\sqrt{K_j^0(\mathbf{X},\mathbf{X})K_j(\mathbf{X}',\mathbf{X}')}\right)$.

Thus, the kernel function is:

$$K(\mathbf{X},\mathbf{X}') = \sigma_d^2 + \sigma_v^2 \sum_{f=1}^{M} \sum_{j=1}^{(H-k+1)^2} \frac{\sqrt{K_j^0(\mathbf{X},\mathbf{X})K_j^0(\mathbf{X}',\mathbf{X}')}}{\pi} \left(\sin\theta_j + (\pi-\theta_j)\cos\theta_j\right).$$

The kernel function encodes the relationships between input examples based on the network's weights and activations. The training set consists of N examples. Once the kernel $K(X,X')$ is computed, predictions are made by evaluating the required kernel matrices. First, the $N \times N$ kernel matrix K_{xx}, which contains the pairwise kernel values between all training examples, is computed. Then, $K_{xx}^{-1}Y$ is calculated using a linear system solver. For the test set, which contains N_T examples, the matrix K_{x^*x}, representing the kernel between the test and training examples, is evaluated. The final predictions for the test examples are given by $\arg\max(K_{x^*x}K_{xx}^{-1}Y)$, where Y is the matrix of one-hot encoded regression targets used during training. This method reframes the classification task as multi-output regression, with the predicted class determined by maximizing the regression output for each test example.

As for the FNN, because the weight and bias parameters are taken to be i.i.d., the post-activations $\mathbf{Y}_j(\mathbf{X})$, $\mathbf{Y}'_j(\mathbf{X})$ (defined in Eq. 3.17) are independent for $j \neq j'$. Moreover, since $O_k(\mathbf{X})$ is a sum of i.i.d. terms, it follows from the Central Limit Theorem that in the limit of infinite width $N \to \infty$, $O_k(\mathbf{X})$ will be Gaussian distributed. Likewise, from the multidimensional Central Limit Theorem, any finite collection of $\{O_k(\mathbf{X}_1), ..., O_k(\mathbf{X}_n)\}$ will have a joint multivariate Gaussian distribution, which is exactly the definition of a Gaussian process.

3.3 Numerical Experiment

In this section, numerical experiments were conducted to examine the univariate and multivariate properties of the output elements, as described in previous subsections. Specifically, as the number of kernels in the convolutional layer increases indefinitely, a single output element converges to a normal distribution. In contrast, the joint distribution of corresponding output elements from different input images converges to a multivariate normal distribution. These theoretical results are substantiated in Section 3.2.1 and Section 3.2.3.

3.3.1 Univariate Normality Test

To analyze the univariate properties of the output elements, one image from the CIFAR-10 dataset (Krizhevsky 2009) was randomly selected as the input sample to analyze the univariate properties of the output elements. The analysis was performed using the CNN model defined in Lemma 1, with $\sigma_F = 1.0$, $\sigma_b = 0.1$, $\sigma_d = 0.2$, and a large number of kernels (1,000, 5,000, and 10,000). The model was run 1,000 times to generate 1,000 output samples. This sample size was selected to better represent the underlying population distribution, particularly the theoretical asymptotic normal distribution (Eq. 3.20). To assess the alignment of the output distributions with the theoretical asymptotic normal distribution, the Kolmogorov-Smirnov test (Massey 1951) was applied. The effect of increasing the number of kernels was also examined to determine whether the output distribution converges more closely to the theoretical normal distribution. Q-Q plots and histograms were generated to visually assess normality, and the Shapiro-Wilk test (Shapiro and Wilk 1965) was employed to rigorously evaluate the normality of the data.

Q-Q plot and Histogram

A quantile-quantile (Q–Q) plot (Dodge 2008) is a graphical tool used to compare two probability distributions by plotting their quantiles against each other. Each point (x, y) on the plot represents a quantile of the second distribution (y-axis) plotted against the same quantile of the first distribution (x-axis), forming a parametric curve where the parameter corresponds to the quantile interval.

The construction of a Q–Q plot involves two primary steps:

1. Arrange the data $\{x_1, x_2, \ldots, x_n\}$ in increasing order as $x_{[1]} \leq x_{[2]} \leq \cdots \leq x_{[n]}$.
2. Assign to each data point the corresponding quantile q_i of the standard normal distribution. Plot on a graph as ordinates the ordered data x_i and as abscissae the quantiles q_i.

If two variables follow the same distribution, their quantiles, plotted against each other, will align along a line with a slope of 1. In a Q–Q plot, if the distributions being compared are similar, the points will approximately lie on the identity line $y = x$. The further the points deviate from this diagonal line, the more significant the difference between the empirical and theoretical distributions. If the distributions are linearly related but not identical, the Q–Q plot points will still form a straight line but may not necessarily lie on $y = x$.

Different shapes of Q–Q plots provide insights into the relationship between distributions (Dodge 2008). A U-shaped graph indicates that one distribution is skewed relative to the other, while an S-shaped graph suggests that one distribution has a heavier tail (i.e., more extreme values) than the other. These patterns reveal how the empirical distribution deviates from the theoretical one, particularly regarding skewness and the influence of extreme values.

Q–Q plots are useful for visually comparing the location, scale, and skewness of two distributions and can also assist in parameter estimation for distributions in the location-scale family. They're commonly used to assess the similarity between a dataset and a theoretical distribution (Gnanadesikan 1977, Thode 2002), providing a graphical method for evaluating goodness of fit, which is often more informative than a numerical statistic alone. They are also employed to compare two theoretical distributions (Gibbons and Chakraborti 2014). Unlike scatter plots, Q–Q plots do not require data values to be paired, nor must the sample sizes of the two groups be equal.

- $p = \Pr(T \geq t \mid H_0)$ for a one-sided right-tail test-statistic distribution.
- $p = \Pr(T \leq t \mid H_0)$ for a one-sided left-tail test-statistic distribution.
- $p = 2 \min \{\Pr(T \geq t \mid H_0), \Pr(T \leq t \mid H_0)\}$ for a two-sided test-statistic distribution. If the distribution of T is symmetric around zero, then $p = \Pr(|T| \geq |t| \mid H_0)$

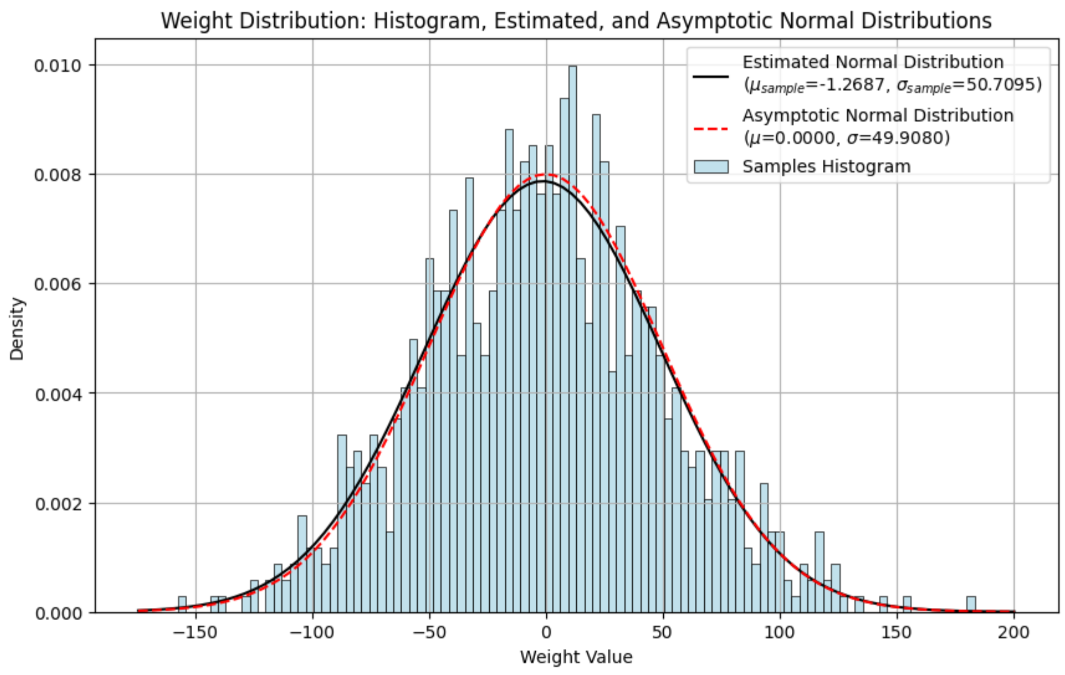


Figure 3.1: Histogram for M = 1000

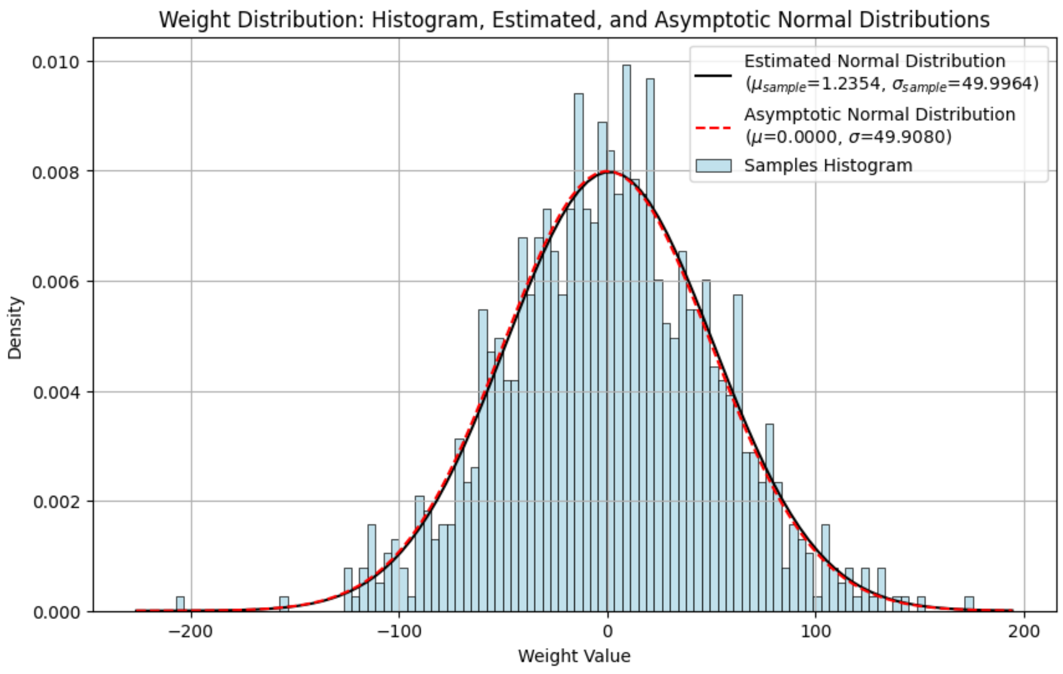


Figure 3.2: Histogram for M = 5000

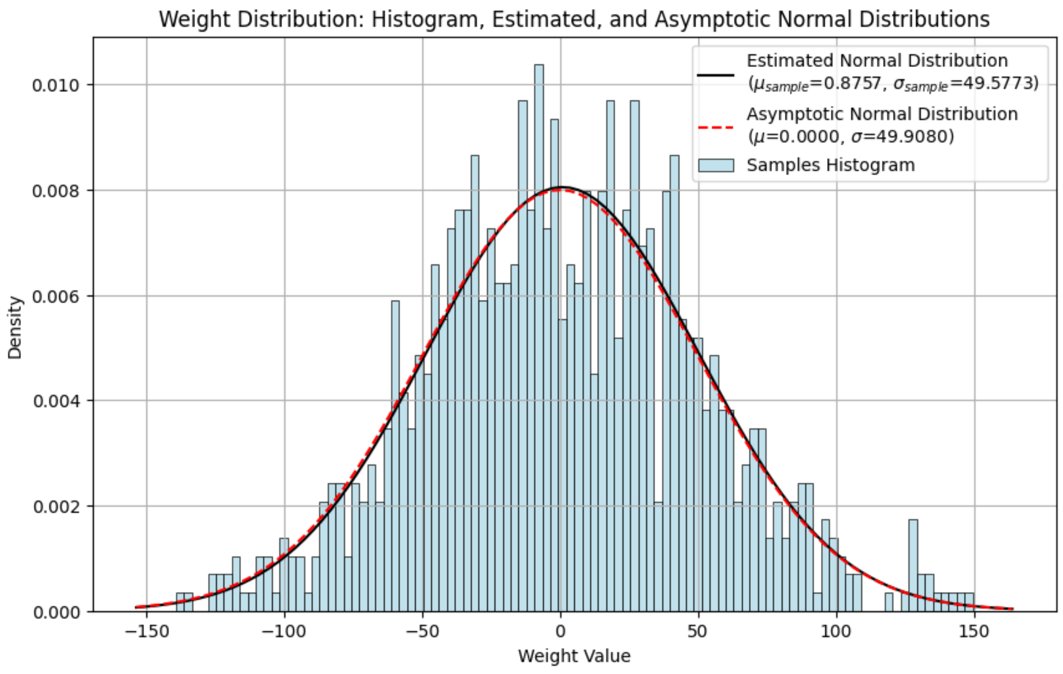


Figure 3.3: Histogram for M = 10,000

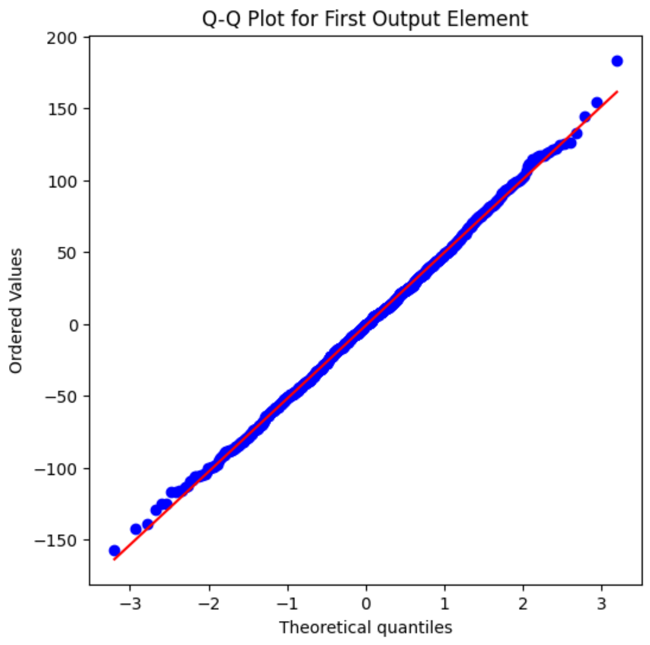


Figure 3.4: Q-Q Plot for M = 1000

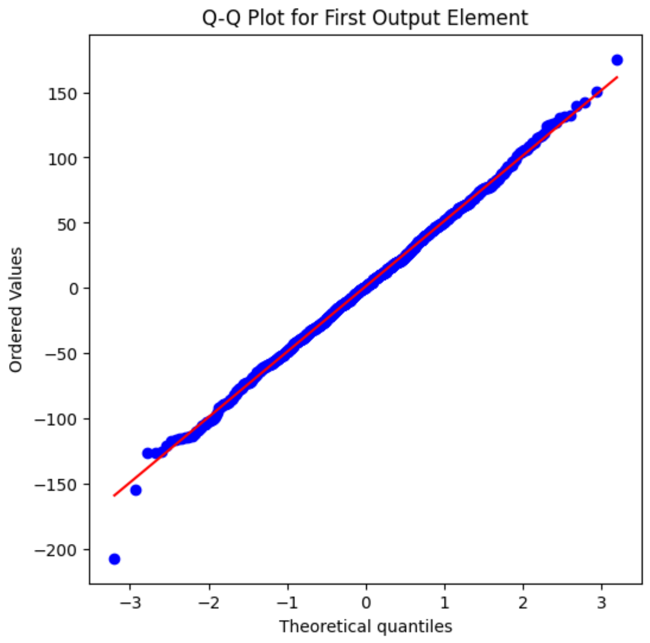


Figure 3.5: Q-Q Plot for M = 5000

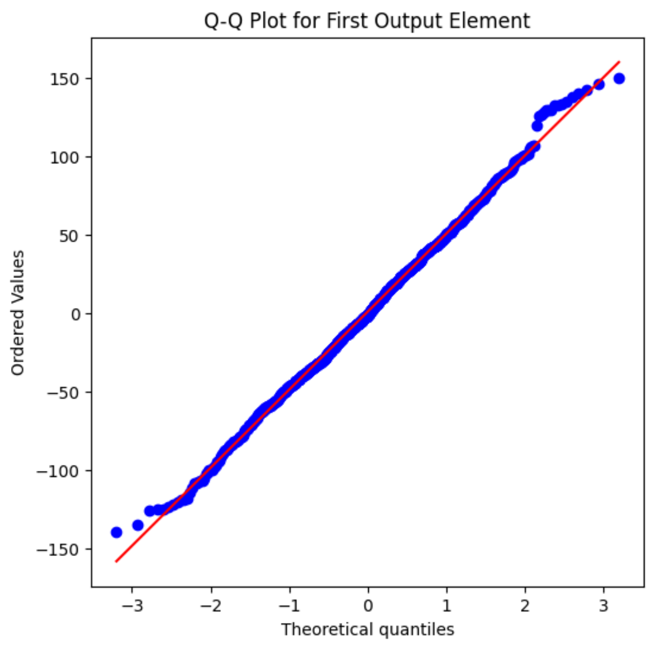


Figure 3.6: Q-Q Plot for M = 10,000

The histograms and Q-Q plots above represent univariate normality tests for three cases: $M = 1000, 5000$, and $10,000$. The output samples O satisfy $\frac{O-\mu}{\sigma} \sim N(0,1)$, so the points in the Q-Q plot form a straight line with intercept μ and slope σ. Theoretically, μ should be 0, as the weights of the fully connected layer are sampled from a normal distribution with zero mean. According to Eq. 3.7, even a slight deviation in the mean of the weights $\mathbf{v}$ can result in a more significant deviation in the mean of the output samples μ. Specifically, Eq. 3.7, $\mu_{f,k} = \sum_{j=1}^{N} \mathbb{E}_{\mathbf{v}}[\mathbf{v}_{f,k}[j]]\mathbb{E}_{\mathbf{Y}_f}[\mathbf{Y}_f[j]]$, indicates that deviations in the weights propagate through the network, affecting the output mean. This discrepancy arises due to sampling variability, where the sample mean from a finite set of random values often differs from the population mean. The law of large numbers ensures that the sample mean will converge to 0 as the sample size increases, but minor deviations are expected with finite samples.

Furthermore, numerical precision limitations inherent in computer systems, such as floating-point arithmetic, can introduce minor rounding errors when generating and processing random numbers. These computational errors can contribute to the observed deviation of the sample mean from 0.

The deviation of the variance σ from its theoretical value is also caused by sampling variability and numerical precision issues.

The histograms demonstrate that the estimated normal distribution from the sample data closely aligns with the theoretical asymptotic normal distribution (Eq. 3.16) and closer as the number of kernels, denoted as M increases. This indicates that the output samples converge toward a normal distribution as M goes to infinity, and the estimated distribution is very similar to the theoretical distribution derived in Lemma 1.

Formal tests for univariate normality

The Shapiro-Wilk test is a statistical method used to assess whether a dataset follows a normal distribution. Developed by Shapiro and Wilk (1965), the test evaluates whether a given sample originates from a normally distributed population. The test is based on the concept that, for a normally distributed sample, the ordered sample values (arranged in increasing order) can be approximated as a linear function of the expected values of the order statistics drawn from a standard normal distribution. Specifically, if the $\{y_i\}$ represent a normal sample, then y_i can be expressed as:

$$y_i = \mu + \sigma x_i, \quad (i = 1, 2, \ldots, n)$$

where x_i are ordered random samples of size n from a standard normal distribution. The test statistic W is constructed by comparing the square of a linear combination of the ordered sample data to the sample variance.

The W statistic is calculated as:

$$W = \frac{(\sum_{i=1}^{n} a_i y_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2},$$

where y_i are the ordered sample values (with y_1 being the smallest), and $\bar{y}$ is the sample mean. The constants a_i are derived from the expected values, variances, and covariances of the order statistics of a sample of size n from a normal distribution and are given by:

$$a_i = \frac{m' V^{-1}}{(m' V^{-1} V^{-1} m)^{1/2}}.$$

Here, V represents the covariance matrix of the sample $x' = (x_1, \ldots, x_n)$, and m is the vector of expected values for $x' = (x_1, \ldots, x_n)$. If the data follows a normal distribution, the test statistic W will be close to 1. Lower values of W indicate greater deviations from normality.

The Kolmogorov-Smirnov (K-S) test (Massey 1951) is a non-parametric test used to compare a sample distribution with a reference probability distribution (one-sample test) or to compare two sample distributions (two-sample test) to assess whether they come from the same distribution. The mathematical foundation of the K-S test lies in measuring the largest difference between the empirical cumulative distribution function (ECDF) of the sample(s) and the theoretical cumulative distribution function (CDF) of the reference distribution.

The empirical distribution function F_n for n independent and identically distributed (i.i.d.) ordered observations X_i is defined as:

$$F_n(x) = \frac{\text{number of (elements in the sample} \le x)}{n} = \frac{1}{n} \sum_{i=1}^{n} 1_{(-\infty, x]}(X_i),$$

where $1_{(-\infty,x]}(X_i)$ is the indicator function, which equals 1 if $X_i \le x$ and 0 otherwise.

In the one-sample case, the Kolmogorov-Smirnov statistic for a given cumulative distribution function $F(x)$ is:

$$D_n = \sup_x |F_n(x) - F(x)|,$$

where $\sup_x$ represents the supremum of the set of distances, $F_n(x)$ is the ECDF of the sample and $F(x)$ is the CDF of the reference distribution. Intuitively, the statistic measures the largest absolute difference between the two distribution functions across all x values.

For the two-sample test, the statistic $D_{n,m}$ is:

$$D_{n,m} = \sup_x |F_n(x) - G_m(x)|,$$

where $F_n(x)$ and $G_m(x)$ are the ECDFs of the two samples.

The K-S test statistic reflects the maximum vertical distance between the ECDF(s) and the theoretical CDF. Suppose the calculated K-S statistic is more significant than a critical value derived from the Kolmogorov distribution. In that case, the null hypothesis that the sample(s) follow the reference distribution is rejected.

The power of the K-S test increases with sample size, making it sensitive to deviations in both the location and shape of the distributions. This test is advantageous as it does not assume a specific distribution form and can be applied to continuous distributions (Massey 1951).

In null-hypothesis significance testing, the p-value is the probability of obtaining test results at least as extreme as the observed result, assuming that the null hypothesis is true. Under the null hypothesis, it represents the probability of observing a real-valued test statistic at least as extreme as the one obtained (Wasserstein and Lazar 2016).

Consider an observed test statistic t from an unknown distribution T. The p-value p is the prior probability of observing a test statistic value at least as extreme as t, assuming the null hypothesis H_0 is true. This is expressed as follows:

- $p = \Pr(T \geq t \mid H_0)$ for a one-sided right-tail test-statistic distribution.
- $p = \Pr(T \leq t \mid H_0)$ for a one-sided left-tail test-statistic distribution.
- $p = 2\min\{\Pr(T \geq t \mid H_0), \Pr(T \leq t \mid H_0)\}$ for a two-sided test-statistic distribution. If the distribution of T is symmetric around zero, then $p = \Pr(|T| \geq |t| \mid H_0)$

Table 3.1: Shapiro-Wilk and Kolmogorov-Smirnov tests for 1000 samples

	M = 1000		**M = 5000**		**M = 10,000**	
Test	Test statistic	p-value	Test statistic	p-value	Test statistic	p-value
Shapiro-Wilk	0.9989	0.7959	0.9987	0.6845	0.9977	0.1813
Kolmogorov-Smirnov	0.0245	0.5780	0.0183	0.8846	0.0176	0.9119

The results of the Shapiro-Wilk test, presented in Table 3.1, suggest that the output samples follow a normal distribution, as the test statistics are all very close to 1. The null hypothesis of the Shapiro-Wilk test posits that the sample distribution is normal. Consequently, the large p-values indicate no significant evidence to reject the null hypothesis, supporting the assumption that the output samples are likely normally distributed.

The Kolmogorov-Smirnov (K-S) test compares the empirical distribution of the sample to a reference distribution. The null hypothesis in this case is that the sample distribution is identical to the reference distribution. As shown in Table 3.1, the test statistics are small, reflecting the maximum absolute difference between the empirical cumulative distribution function (ECDF) and the reference cumulative distribution function (CDF). With p-values well above 0.05, no significant evidence suggests that the sample distribution deviates from the reference distribution.

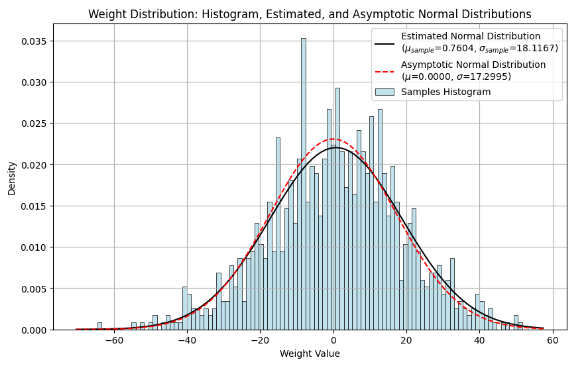
Furthermore, as M increases from 1,000 to 5,000 and then to 10,000, the K-S test statistic steadily decreases (from 0.0245 to 0.0176), demonstrating that the empirical distribution of the sample becomes increasingly closer to the reference distribution. Correspondingly, the p-values increase (from 0.5780 to 0.9119), providing more substantial evidence that the sample does not significantly deviate from the reference distribution as the sample size grows.

Univariate Normality Test for FNNs:

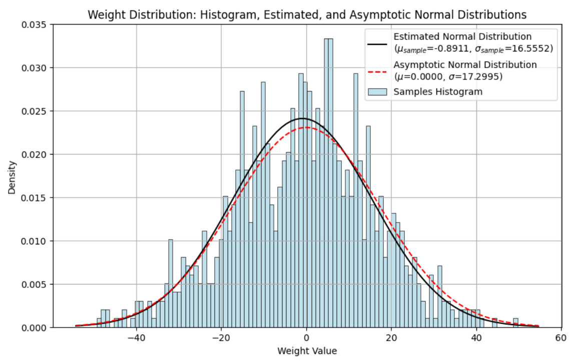
For the one-layer Feedforward Neural Network (FNN), the univariate normality of the output elements was tested using the same methods applied in the CNN test. Specifically, the same image from the CIFAR-10 dataset was used as input, and the FNN model was defined with parameters $\sigma_Q = 1.0$, $\sigma_b = 0.1$, $\sigma_d = 0.2$, and $\sigma_v = 1.0$. The hidden layer of the FNN was evaluated with varying numbers of neurons: 500, 1,000, 5,000, and 10,000. The model was run 1,000 times for each configuration to generate 1,000 output samples.

The Kolmogorov-Smirnov test was employed to assess the univariate normality of the output distribution, specifically, to compare the generated output samples against the theoretical normal distribution. Additionally, Q-Q plots and histograms were generated to visually evaluate the output distribution's normality. The Shapiro-Wilk test was also utilized as a more rigorous test for normality.

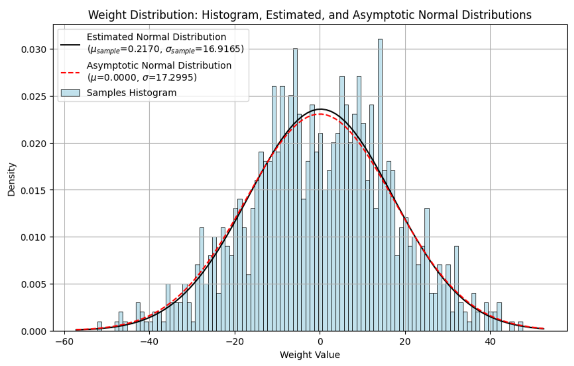
By increasing the number of neurons, the aim was to observe whether the output distribution approached the theoretical asymptotic normal distribution expected for FNNs. This approach allowed for a comprehensive analysis of the normality of the output elements in FNNs, consistent with the methods applied in the CNN case.



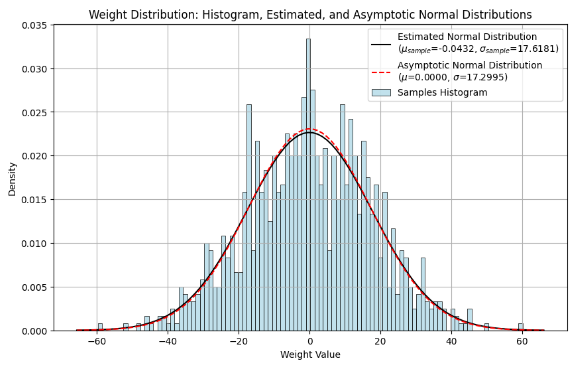
(a) Histogram for FNN with 500 neurons.



(b) Histogram for FNN with 1,000 neurons.

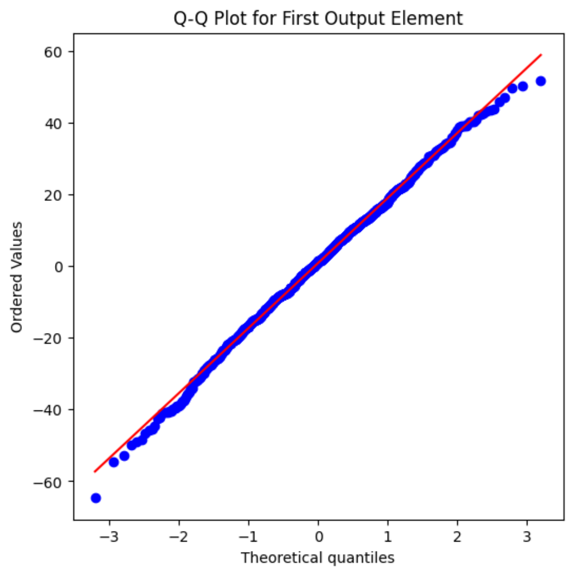


(c) Histogram for FNN with 5,000 neurons.

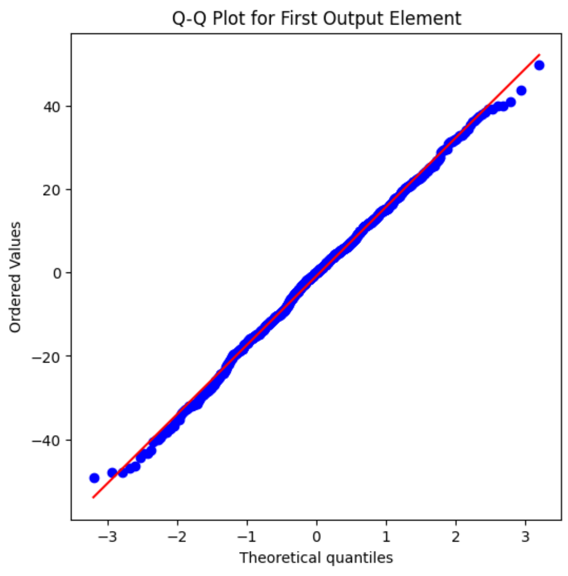


(d) Histogram for FNN with 10,000 neurons.

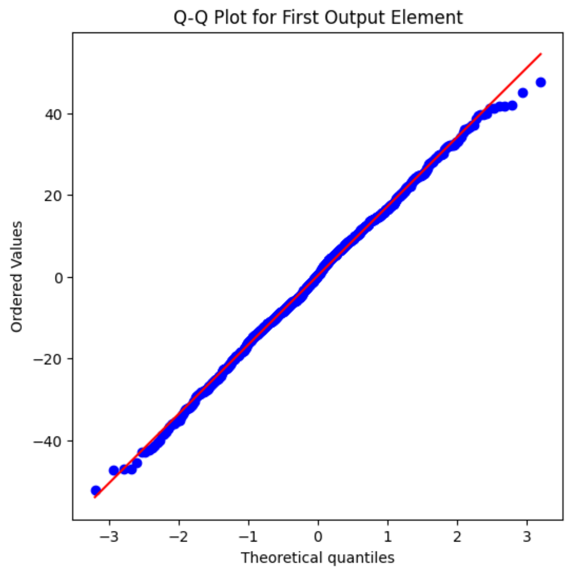
Figure 3.7: Histograms showing the univariate normality test results for FNN output with different numbers of neurons. The tests were conducted using 1,000 output samples generated from 1,000 runs of the model, using CIFAR-10 input images and the specified parameter settings.



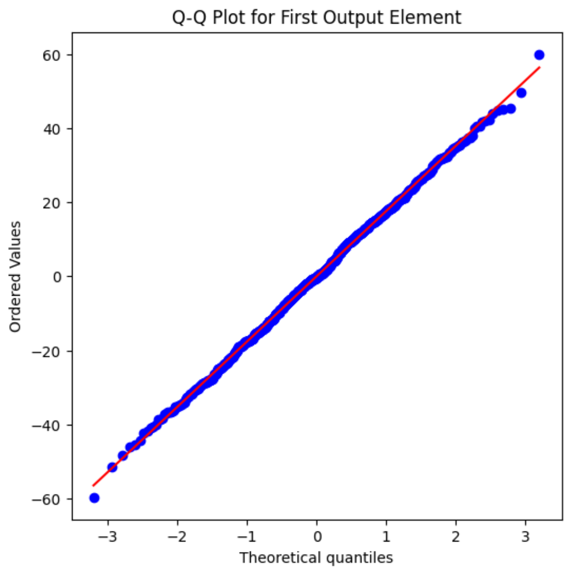
(a) Q-Q plot for FNN with 500 neurons.



(b) Q-Q plot for FNN with 1,000 neurons.



(c) Q-Q plot for FNN with 5,000 neurons.



(d) Q-Q plot for FNN with 10,000 neurons.

Figure 3.8: Q-Q plots for assessing the univariate normality of FNN output with varying neuron numbers. These plots correspond to the histograms shown earlier and provide a visual comparison between the sample quantiles and theoretical quantiles from a normal distribution.

Figure 3.7 and Figure 3.8 above show univariate normality tests for four cases: $N = 500$, 1000, 5000, and $10,000$, where N is the number of neurons in the hidden layer of the FNN.

The output samples O satisfy $\frac{O-\mu}{\sigma} \sim N(0,1)$, leading the points in the Figure 3.8 to form a straight line with an intercept μ and slope σ. Theoretically, μ should be 0, as the weights of the fully connected layer are sampled from a normal distribution with zero mean.

The histograms illustrate that the estimated normal distribution from the sample data closely aligns with the theoretical asymptotic normal distribution (Eq. 3.19) and becomes more accurate as the number of neurons, denoted by N, increases. This indicates that the output samples converge toward a normal distribution as N approaches infinity, with the estimated distribution closely resembling the theoretical distribution.

Table 3.2: Shapiro-Wilk and Kolmogorov-Smirnov tests for 1000 samples with varying neuron counts N.

	N = 1000		**N = 5000**		**N = 10,000**	
Test	Test statistic	p-value	Test statistic	p-value	Test statistic	p-value
Shapiro-Wilk	0.9975	0.1320	0.9980	0.2722	0.9992	0.9431
Kolmogorov-Smirnov	0.0345	0.1810	0.0304	0.3083	0.0250	0.5535

Similar to the results obtained in the CNN case, in Table 3.2, the Shapiro-Wilk test statistic values and corresponding p-values indicate that the first element of the output samples is normally distributed. This conclusion is supported by the fact that all p-values exceed the significance level of 0.05, and the test statistics are consistently close to 1. Furthermore, for the Kolmogorov-Smirnov (KS) test, it is noteworthy that as the number of neurons in the hidden layer increases, the test statistics decrease, while the p-values increase. This trend suggests a closer convergence to the theoretical normal distribution as N approaches infinity.

In addition, it has been observed that the Kolmogorov-Smirnov (KS) test statistics are consistently larger in the FNN case compared to the CNN case. This difference can be explained by the effective number of neurons when the convolutional layer is treated as a fully connected layer, which is given by $MW_{\text{out}}H_{\text{out}}$, as indicated in Eq. 3.18. The increased number of neurons effectively corresponds to a larger sample size, making the output elements closer to the asymptotic normal distribution.

The higher dimensionality in the convolutional representation facilitates more effective averaging, which enhances the convergence of the output distribution to the theoretical normal distribution. This observation aligns with the Central Limit Theorem, which states that the resulting distribution approximates normality more accurately as the number of contributing factors increases. Consequently, the smaller KS test statistics indicate that the empirical distribution of the output aligns more closely with the asymptotic normal distribution as the number of effective neurons increases.

3.3.2 Bivariate Normality Test

In the multivariate case, the bivariate properties of the output elements were analyzed by randomly selecting two images from the CIFAR-10 dataset as input samples. The analysis was conducted using the CNN model described in Lemma 1, with $\sigma_F = 1$, $\sigma_b = 0.1$, $\sigma_d = 0.2$, and 10,000 kernels. The model was run 300 times to generate 300 output samples. A scatter plot of variable pairs, along with contour lines representing the estimated probability density of a multivariate normal distribution, was generated to visualize the sample data. Additionally, a Gamma plot (Chi-squared Q-Q plot) was created to assess bivariate normality, and Mardia's test was applied to statistically evaluate the data.

Scatter plot and Gamma plot

A Gamma plot (chi-squared Q-Q plot) is used to compare a distribution with the chi-squared distribution by plotting its quantiles against the corresponding percentiles of the chi-squared distribution. For a p-dimensional multivariate normal population, let $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n$ be a random sample. The generalized squared distance is given by

$$d_j^2 = (\mathbf{x}_j - \bar{\mathbf{x}})' \mathbf{S}^{-1} (\mathbf{x}_j - \bar{\mathbf{x}}),$$

which follows a chi-squared distribution with p degrees of freedom, denoted χ_p^2 (Hogg et al. 2004). A Gamma plot can be used to determine whether d_j^2 follows a χ_p^2 distribution by plotting the ordered d_j^2 values against the corresponding percentiles of the chi-squared distribution. If the data is bivariate or multivariate normal, the points on the plot should align linearly.

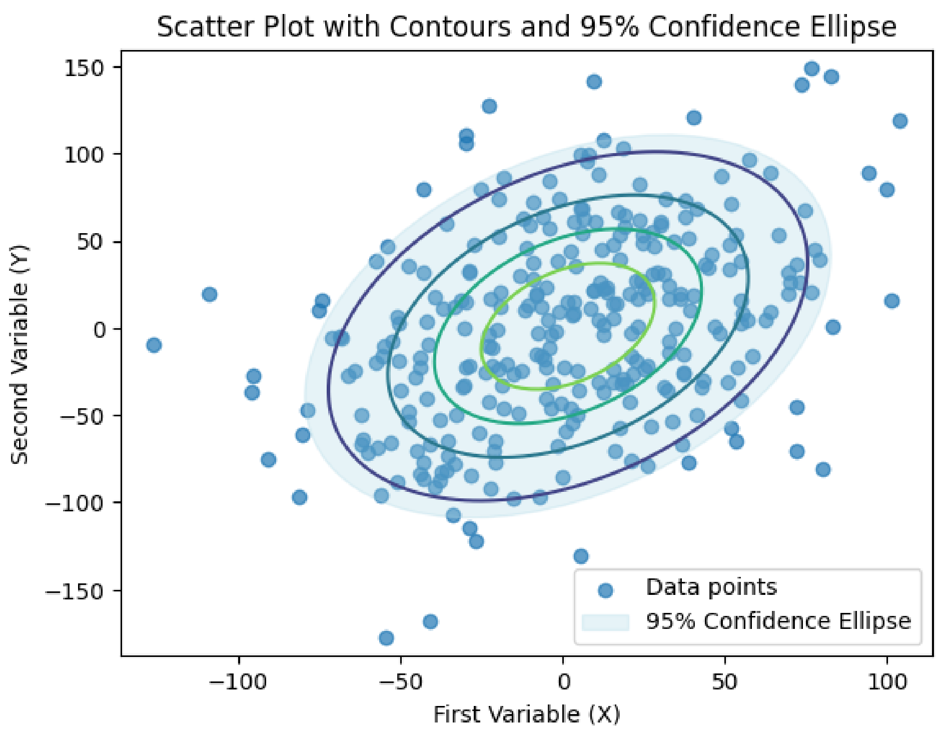


Figure 3.9: Scatter plot of 300 bivariate normal samples, M=10,000

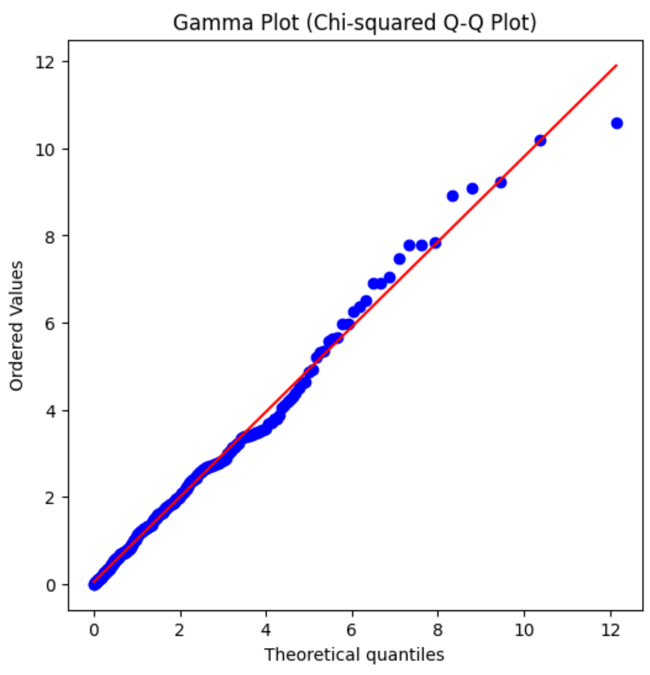


Figure 3.10: Gamma Plot for M = 10,000

For bivariate normally distributed samples, the contours of constant density are expected to form ellipses. The scatter plot should reflect this structure by displaying an overall elliptical pattern (Johnson and Wichern 2007). The light blue region represents the 95% confidence ellipse, indicating where 95% of the data points are expected to lie if the data indeed follows a bivariate normal distribution, as shown in Figure 3.9. This ellipse is centered on the estimated mean of the data and is oriented according to the covariance structure of the variables. In Figure 3.10, the fact that most sample points lie close to the red diagonal line indicates that the Mahalanobis distances of the samples follow a chi-squared distribution, further supporting the assumption of bivariate normality. However, in both plots, some points are distant from the ellipse in the scatter plot and the straight line in the Gamma plot.

Formal tests for bivariate normality

Two common methods are used to test bivariate normality property: Royston's test and Mardia's test. Royston's test is an extension of the Shapiro and Wilk goodness of fit test for univariate normality, while Mardia's test is a multivariate extension of measures of skewness and Kurtosis.

Royston's H test (Royston 1983) is a multivariate extension of the popular test for Univariate Normality (UVN), the Shapiro-Wilk W test (Royston 1982). The Shapiro-Wilk test is generally considered an excellent test for UVN (Mecklin 2000). Let W_j denote the value of the Shapiro-Wilk statistic for the j-th variable in a p-variate distribution.

Then, define

$$R_j = \Phi^{-1}\left(\frac{1}{2}\Phi(1 - W_j)\right)\left(\frac{\lambda - \mu}{\sigma}\right)^2,$$

where λ, μ, and σ are calculated from polynomial approximations given in (Royston 1982), and $\Phi(\cdot)$ denotes the standard normal cumulative distribution function (CDF). Now, if the data follows a multivariate normal distribution (MVN), the Royston's statistic $H = \xi \sum R_j / p$ is approximately $\chi^2_{\hat{\xi}}$ distributed, where

$$\hat{\xi} = \frac{p}{1 + (p-1)\bar{c}},$$

and $\bar{c}$ is an estimate of the average correlation among the R_j's (Royston 1983). This chi-squared distribution is used to obtain critical values for the test. Royston's H test was found to have good power against many different alternative distributions (Mecklin 2000).

Mardia's test (Mardia 1970) is another widely used statistical method for assessing multivariate normality (MVN) by evaluating two aspects of the data: skewness and kurtosis. Mardia's skewness statistic measures the symmetry of the data, while Mardia's kurtosis statistic assesses the heaviness of the tails in the distribution. In Mardia's test, if the skewness and kurtosis are significantly different from the values expected under normality, the data is considered to deviate from multivariate normality. Specifically, Mardia's skewness statistic is asymptotically chi-square distributed, while Mardia's kurtosis statistic follows a normal distribution. If the p-values corresponding to these statistics are greater than a chosen significance level (e.g., 0.05), the null hypothesis that the data follows a multivariate normal distribution is not rejected.

Mardia's skewness statistic is defined as:

$$b_{1,p} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left[\mathbf{d}(\mathbf{x}_i, \mathbf{x}_j)\right]^3,$$

where n is the number of observations (samples), $\mathbf{x}_i$ and $\mathbf{x}_j$ are two multivariate samples, and $\mathbf{d}(\mathbf{x}_i, \mathbf{x}_j)$ is the Mahalanobis distance between the samples:

$$\mathbf{d}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \bar{\mathbf{x}})^\top \mathbf{S}^{-1} (\mathbf{x}_j - \bar{\mathbf{x}}).$$

Here, $\bar{\mathbf{x}}$ is the sample mean vector and $\mathbf{S}^{-1}$ is the inverse of the sample covariance matrix.

Mardia's kurtosis statistic is given by:

$$b_{2,p} = \frac{1}{n} \sum_{i=1}^{n} \left[\mathbf{d}(\mathbf{x}_i)\right]^2,$$

where $\mathbf{d}(\mathbf{x}_i)$ is the Mahalanobis distance of sample $\mathbf{x}_i$ from the sample mean. For a multivariate normal distribution, the expected value of the kurtosis statistic is $p(p+2)$, where p is the number of variables.

If the skewness statistic $b_{1,p}$ is significantly different from zero, it indicates asymmetry in the data, suggesting a departure from multivariate normality. Likewise, if the kurtosis statistic $b_{2,p}$ significantly deviates from $p(p+2)$, the distribution's tails are either heavier or lighter than those of a normal distribution. In practice, the kurtosis statistic in Mardia's test for multivariate normality is adjusted by subtracting the expected kurtosis under normality and scaling the result, which is expressed as $b_{2,p} - p(p+2)$.

Table 3.3: Test for bivariate normality of 1000 samples using Royston's test and Mardia's test

Test	Royston's	Mardia's	
		Skewness (0)	Kurtosis ($b_{2,p} - 6$)
Test statistic	0.1169	0.2872	-0.6329
p-value	0.9437	0.9906	0.5268

Royston's and Mardia's tests were applied to the lumber stiffness data to assess whether the measurements follow a bivariate normal distribution. The hypotheses for these tests are as follows: H_0: The measurements are bivariate normal, versus H_1: The measurements are not bivariate normal. The results from both tests are presented in Table 3.3. At the 5% significance level, the null hypothesis of bivariate normality is not rejected, suggesting that the data is consistent with bivariate normality.

Mardia's test's skewness and kurtosis statistics indicate no significant deviation from bivariate normality. The skewness statistic 0.287 (p-value = 0.991) suggests the data is symmetric. In contrast, the kurtosis statistic of -0.633 (p-value = 0.527) indicates that the distribution's tails do not significantly differ from those expected under normality. These results suggest that the bivariate sample can reasonably be considered bivariate normal. Similarly, Royston's test yields a p-value of 0.944, well above the 0.05 threshold, further supporting the assumption of bivariate normality.

Bivariate Normality Test for FNNs

Following the CNN case, a similar bivariate analysis was performed for the FNN model, with the number of neurons set to 20,000. The parameters were set as $\sigma_F = 1.0$, $\sigma_b = 0.1$, $\sigma_d = 0.2$, and $\sigma_v = 1.0$. As in the previous experiment, two images from the CIFAR-10 dataset were randomly selected as input samples, and the model was run 300 times to generate 300 output samples. To visualize the bivariate properties of the output, a scatter plot of the variable pairs was generated, featuring contour lines representing the estimated probability density of a multivariate normal distribution, along with a 95% confidence ellipse. Additionally, a Gamma plot (Chi-squared Q-Q plot) was created to assess bivariate normality, and Mardia's test was applied to statistically evaluate the normality of the output.

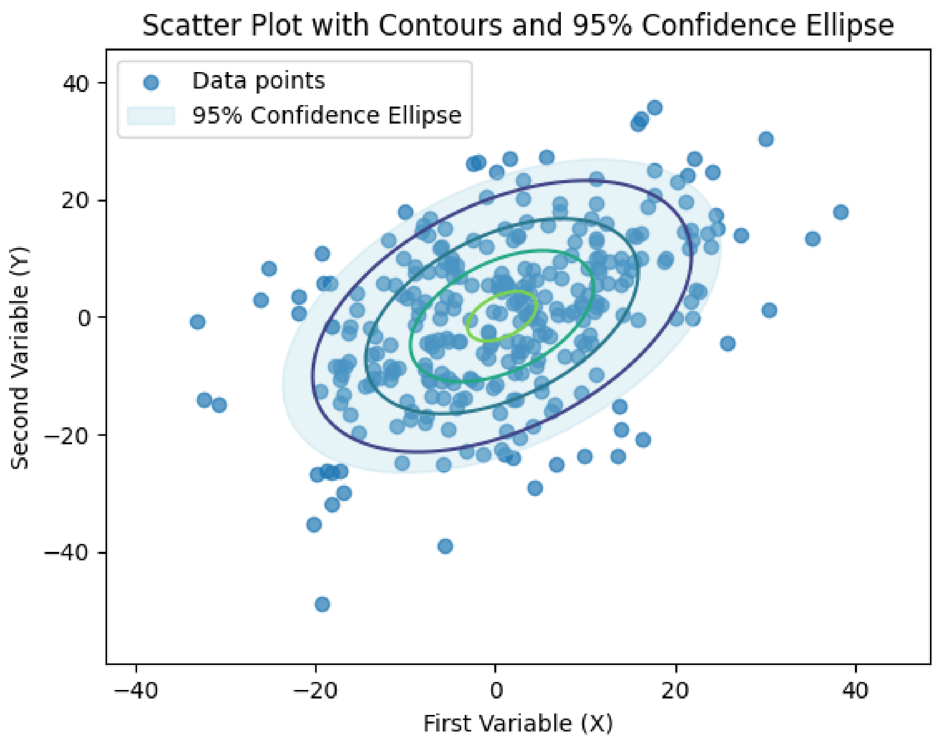


Figure 3.11: Scatter plot of 300 bivariate normal samples, N=20,000

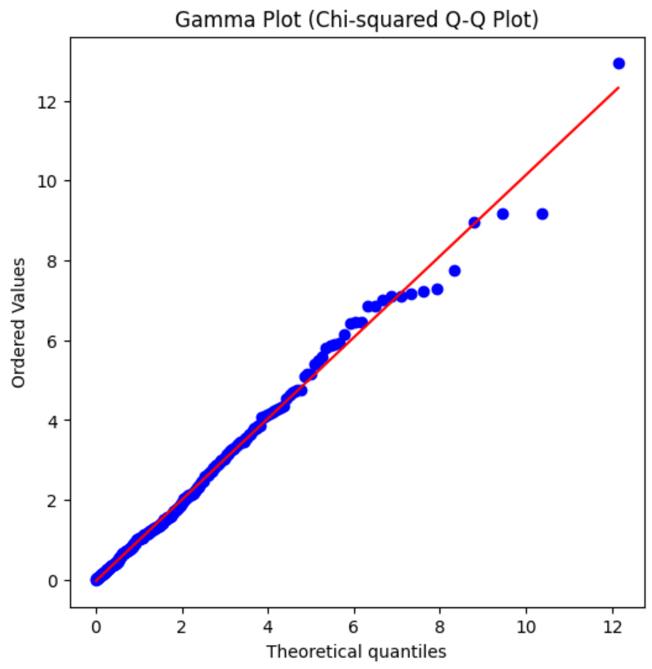


Figure 3.12: Gamma Plot for N = 20,000

As expected for bivariate normally distributed samples, the scatter plot exhibited an elliptical shape, with the light blue region representing the 95% confidence ellipse. This indicates the region where 95% of the data points should fall if the distribution is truly bivariate normal, as shown in Figure 3.11. The orientation and center of the ellipse are determined by the covariance and the mean of the data, respectively.

In Figure 3.12, the proximity of most sample points to the red diagonal line in the Gamma plot suggests that the Mahalanobis distances follow a chi-squared distribution, supporting the assumption of bivariate normality. However, similar to the CNN case, a few points deviate from the ellipse in the scatter plot and the straight line in the Gamma plot, suggesting slight deviations from the expected normality in some areas of the data.

Table 3.4: Test for bivariate normality of 1000 samples using Royston's test and Mardia's test

Test	Royston's	Mardia's	
		Skewness (0)	Kurtosis $(b_{2,p} - 6)$
Test statistic	0.8818	6.3147	0.0448
p-value	0.6457	0.1768	0.9643

The bivariate normality of the 1,000 output samples was evaluated using Royston's and Mardia's tests, with the results summarized in Table 3.4. Royston's test yielded a test statistic of 0.8818 and a p-value of 0.6457, indicating no significant deviation from normality at the 5%

significance level.

Mardia's test further corroborates this finding. The skewness statistic was 6.3147 with a p-value of 0.1768, and the kurtosis statistic was 0.0448 with a p-value of 0.9643. Both statistics fail to reject the null hypothesis of normal skewness and kurtosis, respectively.

Collectively, these test results confirm that the output data adheres to bivariate normality, thereby validating the model's assumptions of normality in this context.

4 Conclusions

This thesis has explored the mathematical equivalence between convolutional and fully connected layers in neural networks and the Gaussian process behavior in infinitely wide neural networks. Through a detailed investigation of CNNs and their transformation into FCNNs, the study has revealed how convolutional operations can be expressed as matrix multiplications, leading to an understanding of the structural differences between these two architectures. This transformation highlights the computational efficiency of CNNs, particularly in tasks involving spatial data, where weight sharing and local receptive fields provide clear advantages over fully connected layers.

Additionally, this thesis has demonstrated that infinitely wide neural networks, including CNNs and FCNNs, exhibit output distributions that converge to normal distributions under prior assumptions. This convergence is supported by the theoretical framework of Gaussian processes, which provides a statistical foundation for understanding the behavior of infinitely wide networks. The numerical experiments conducted in this work further validate these theoretical findings, showing that the output of wide networks aligns closely with Gaussian processes as the network width increases.

The implications of this research are significant for both theory and practice. Understanding the relationship between CNNs and FCNNs offers new insights into neural network design and optimization. Furthermore, the established connection between infinitely wide neural networks and Gaussian processes provides valuable insights into the generalization behavior of neural networks. Specifically, as the width of a neural network layer approaches infinity, the network's behavior converges to that of a Gaussian process due to the Central Limit Theorem. This relationship enables people to apply probabilistic methods to analyze and predict the performance of neural networks under specific prior assumptions.

Future research could extend the Gaussian process approximation to neural architectures beyond fully connected and convolutional networks, offering deeper insights into the probabilistic output distribution of models such as Recurrent Neural Networks (RNNs), Transformers, and Graph Neural Networks (GNNs). This would likely require the development of new theoretical tools to analyze the behavior of these architectures under various assumptions about the weight priors, particularly how their output distributions are influenced by the weight distribution and the input data. Adjusting the weight initialization method based on these insights could improve the performance of models in tasks such as classification and regression. Besides, the impact of various prior assumptions about weight distributions on the probability properties of neural network outputs is also crucial to explore, such as heavy-tailed distributions or structured priors incorporating domain-specific knowledge. Understanding these effects could also improve initialization strategies that enhance model performance. By studying the interplay between input data, weights, and output data, researchers may also develop methods to directly analyze the output distributions of neural networks without processing input data through the networks themselves.

Additionally, researchers could investigate more complex relationships between diverse neural architectures, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Transformers, and Graph Neural Networks (GNNs). This exploration would clarify why certain models are better suited for specific data types—such as CNNs for image data and RNNs for sequential data—and could uncover the underlying factors driving these improvements. By analyzing these transformations, researchers can gain insights into the development of these complex models and identify new strategies to enhance performance, leading to more optimized solutions for various tasks.

Bibliography

Bengio, Y., I. Goodfellow, and A. Courville (2017). *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA.

Guth, F., B. Ménard, G. Rochette, and S. Mallat (2023). "A rainbow in deep network black boxes". In: *arXiv preprint arXiv:2305.18512*.

Lee, J., Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein (2018). *Deep Neural Networks as Gaussian Processes*. arXiv: 1711.00165 [stat.ML]. URL: https://arxiv.org/abs/1711.00165.

Dodge, Y. (2008). *The Concise Encyclopedia of Statistics*. 1st ed. New York, NY: Springer, pp. IX, 622. ISBN: 978-0-387-31742-7. DOI: 10.1007/978-0-387-32833-1. URL: https://doi.org/10.1007/978-0-387-32833-1.

Thode, H. C. (2002). *Testing for Normality*. Section 2.2.2, Quantile-Quantile Plots, p. 21. New York: Marcel Dekker. ISBN: 0-8247-9613-6.

Gibbons, J. and S. Chakraborti (2014). *Nonparametric Statistical Inference, Fourth Edition: Revised and Expanded*. Taylor & Francis. ISBN: 9780203911563. URL: https://books.google.com.hk/books?id=kJbVO2G6VicC.

Shapiro, S. S. and M. B. Wilk (1965). "An analysis of variance test for normality (complete samples)". In: *Biometrika* 52.3/4, pp. 591–611.

Massey, F. J. J. (1951). "The Kolmogorov-Smirnov test for goodness of fit". In: *Journal of the American statistical Association* 46.253, pp. 68–78.

Wasserstein, R. L. and N. A. Lazar (2016). "The ASA statement on p-values: Context, process, and purpose". In: *The American Statistician* 70.2, pp. 129–133. DOI: 10.1080/00031305.2016.1154108. URL: https://doi.org/10.1080/00031305.2016.1154108.

Hogg, R. V., A. T. Craig, and J. W. McKean (2004). *Introduction to Mathematical Statistics*. 6th. Upper Saddle River, NJ: Prentice-Hall.

Johnson, R. A. and D. W. Wichern (2007). *Applied Multivariate Statistical Analysis*. 6th. Upper Saddle River, NJ: Prentice-Hall.

Mecklin, C. (2000). "A comparison of the power of classical and newer tests of multivariate normality". PhD thesis. University of Northern Colorado.

Royston, J. (1983). "Some techniques for assessing multivariate normality based on the Shapiro-Wilk W". In: *Applied Statistics* 32.2, pp. 121–133.

– (1982). "An extension of Shapiro and Wilk's W test for normality to large samples". In: *Applied Statistics* 31.2, pp. 115–124.

Mardia, K. V. (1970). "Measures of multivariate skewness and kurtosis with applications". In: *Biometrika* 57.3, pp. 519–530. DOI: 10.1093/biomet/57.3.519.

Garriga-Alonso, A., C. E. Rasmussen, and L. Aitchison (2019). “Deep Convolutional Networks as Shallow Gaussian Processes”. In: *arXiv preprint arXiv:1808.05587*. URL: `https://arxiv.org/abs/1808.05587`.

Neal, R. M. (1994). “Bayesian Learning for Neural Networks”. PhD thesis. University of Toronto, Dept. of Computer Science.

Billingsley, P. (1995). *Probability and Measure*. 3rd. John Wiley & Sons. ISBN: 0-471-00710-2.

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. Technical Report TR-2009. Toronto, Canada: University of Toronto. URL: `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., pp. 1097–1105.

LeCun, Y., Y. Bengio, and G. Hinton (2015). “Deep learning”. In: *Nature* 521.7553, pp. 436–444. DOI: `10.1038/nature14539`.

Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury (2012). “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups”. In: *IEEE Signal Processing Magazine* 29.6, pp. 82–97. DOI: `10.1109/MSP.2012.2205597`.

Kim, Y. (2014). “Convolutional Neural Networks for Sentence Classification”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pp. 1746–1751. DOI: `10.3115/v1/D14-1181`. URL: `https://www.aclweb.org/anthology/D14-1181`.

Gnanadesikan, R. (1977). *Methods for Statistical Analysis of Multivariate Observations*. New York, NY: Wiley. ISBN: 0-471-30845-5.