Signal to Noise Dynamics and Feature Robustness in Convolutional Neural Networks

A thesis submitted in fulfilment of the requirements for the degree of

Master of Science (MSc) in Mathematics

at

Department of Pure and Applied Mathematics
Faculty of Science
RHODES UNIVERSITY

by

Georgina Bianca Fiorentinos
Student Number: G18F0243

Supervisor: Prof. Marcellin Atemkeng

Abstract

Convolutional neural networks (CNNs) have become foundational to modern computer vision owing to their ability to learn hierarchical feature representations from raw data. Despite their widespread success, the internal relationship between feature quality and the underlying signal to noise ratio (SNR) within these networks remains insufficiently understood. This gap is significant, as imagery data is frequently affected by noise, acquisition artefacts, and heterogeneous backgrounds that challenge the reliability of deep learning models. This thesis presents a systematic investigation into how SNR evolves across the layers of several widely used CNN architectures and examines how these dynamics relate to discriminative performance. Using a dataset of brain tumour Magnetic Resonance Imaging (MRI) slices with tumour masks, the study isolates signal and background components to quantify layer-wise SNR while evaluating representational quality through lightweight linear probe classifiers. Two scenarios are analysed: natural background noise inherent to the dataset and controlled Gaussian noise injected at varying intensities. The results reveal that robustness to noisy inputs is strongly dependent on architectural design. Feed-forward models such as Visual Geometry Group 16-layer network (VGG16) and Visual Geometry Group 19-layer network (VGG19) show a direct correspondence between SNR and classification accuracy, with deeper layers recovering informative features even when early representations degrade. In contrast, architectures employing skip connections or dense feature reuse such as 50-layer Residual Network (ResNet50), 50-layer Residual Network, version 2 (ResNet50V2), and 121-layer Densely Connected Convolutional Network (DenseNet121), achieve strong accuracy despite declining or fluctuating SNR, indicating that discriminative capability does not solely rely on preserving raw signal strength. More advanced models, including EfficientNet-B0 convolutional neural network (EfficientNetB0) and Inception-ResNet-v2 convolutional neural network (InceptionResNetV2), exhibit heightened sensitivity to corruption, while Extreme Inception convolutional neural network (Xception) demonstrates notable resilience, maintaining high accuracy even under substantial intermediate noise. These findings indicate that SNR alone is not a reliable predictor of classification performance. Instead, robustness emerges from architectural mechanisms that transform, re-weight, and filter features to suppress irrelevant variation while enhancing task-specific information. By linking physical signal characteristics with internal feature representations, this thesis provides new insight into how CNNs manage noisy inputs and offers guidance for designing architectures that remain reliable in the real world, noise prone environments.

Contents

Acknowledgments

I want to take a moment to sincerely thank my supervisor, Prof. Marcellin Atemkeng, for his guidance, patience, and thoughtful mentorship. Your insight and academic rigour continually challenged me to think more deeply and strive for clarity in my work. I am truly grateful for the time, care, and trust you invested in me throughout this process.

To my mom, Lorraine Clark-Miller, your love has been the foundation beneath every achievement in my life. Thank you for believing in me, for holding me up, and for reminding me of my own strength whenever I needed it most.

To my brother, Sabro Atakligan, thank you for being my loudest cheerleader and for celebrating every milestone as if it were your own. Your pride in me has always felt like fuel, and your presence in my life is something I treasure deeply.

To my partner, Brad Nelson, you have been my steady ground through the highs and lows of this journey. Through long evenings, looming deadlines, and moments of doubt, you remained calm, patient, and constant. Your quiet strength and unwavering presence made even the hardest days feel manageable. I am so grateful to walk this path alongside you.

To my horse, Ntombi, my loyal companion for over a decade, you have been my sanctuary. In the saddle, the noise fades and the world feels balanced again. You have reminded me to breathe, to trust, and to find joy beyond the desk.

To Vivienne Coetzee, my cherished friend, thank you for your joy, your humour, your support, and for always being in my corner. I'm so grateful for the friendship we share.

And to my cats, Imogen and Titan, thank you for the quiet companionship during long writing sessions. Your gentle presence turned ordinary evenings into moments of comfort.

Finally, I want to thank myself. These past three years have asked a lot of me. Balancing full-time work, my studies, my hobbies, and my social life has not been easy. Yet I showed up, again and again. I pushed through the difficult days, celebrated the small wins, and kept going even when it felt impossible. I am proud of the strength, perseverance, and heart I've poured into this journey.

List of Abbreviations

ANN - Artificial Neural Network

CE-MRI - Contrast Enhanced Magnetic Resonance Imaging

CNN - Convolutional Neural Network

CT - Computed Tomography

DenseNet - Dense Convolutional Network

FLOPs - Floating Point Operations

FN - False Negative

FP - False Positive

LReLU - Leaky Rectified Linear Unit

LRN - Local Response Normalisation

LSVRC - Large-scale Visual Recognition Challenge

MAE - Mean Absolute Error

MBConv - Mobile Inverted Bottleneck Convolution

MNIST - Modified National Institute of Standards and Technology (dataset)

MRI - Magnetic Resonance Imaging

MSE - Mean Square Error

PReLU - Parametric Rectified Linear Unit

ReLU - Rectified Linear Unit

ResNet - Residual Network

SNR - Signal to Noise Ratio

SVM - Support Vector Machine

TN - True Negative

TP - True Positive

Chapter 1

Introduction

1.1 Understanding Convolutional Neural Networks and Signal to Noise Ratio

Convolutional Neural Networks (CNNs) have revolutionised computer vision by attaining cutting edge performance in tasks including image classification, segmentation, and object detection [1]. Their hierarchical structure allows for the gradual extraction of intricate features from unprocessed data. Following the demonstration by AlexNet that deep architectures could effectively facilitate large-scale image recognition [2], CNNs emerged as an important component in nearly all visual recognition systems. He et al. [3] later came up with residual learning, which improved gradient flow and performance in very deep networks. In the medical field, CNNs are being used more and more to automate diagnostic operations. They have succeeded in tumor detection and segmentation [4], organ delineation, and disease categorisation, therefore diminishing subjectivity and enhancing reproducibility in diagnosis [5]. Esteva et al. [6] showed that deep learning models can attain classification accuracy equivalent to that of dermatologists for skin cancer, highlighting its promise in therapeutic applications. CNNs have been included into multi-modal diagnostic pipelines, aiding radiologists in detecting subtle visual elements that may be imperceptible to the human eye [7]. This has increased interest in explainable medical Artificial Intelligence (AI) systems, which aim to not just copy but also improve how doctors make decisions. Even with these advances, we still do not know much about how CNNs work on the inside, especially how they handle the quality of information as it moves through layers.

In clinical imaging, scanner artifacts, acquisition noise, and background variability often compromise data integrity [8]. These distortions can complicate the identification of diagnostic features and make models less reliable, which is why it is important to research how networks deal with noisy inputs and how the signal propagates through CNN architectures. The signal to noise ratio (SNR) is a way to determine how much important information (signal) is kept compared to irrelevant variance (noise) [9]. This concept, long established in signal processing and communications [10], serves as an effective method for assessing the quality of features in deep networks. SNR serves as a connection between traditional

statistical signal processing and contemporary methods of feature learning in this context. Measuring the SNR in each layer allows for the assessment of whether the hierarchical abstractions learned by CNNs enhance signal clarity or if certain architectures inadvertently amplify noise. For instance, first convolutional filters may enhance edges and contrasts, hence strengthening the signal. Excessive parameters may inadvertently introduce noise into the learned representation through additional layers. To assess the efficacy of CNNs in extracting salient information from real-world data, one must understand the underlying mechanisms of these processes. Recent studies indicate that CNNs frequently depend on non-semantic or texture based signals that may not align with significant signal characteristics. Geirhos et al. [11] demonstrated that networks trained on ImageNet [12] display a pronounced bias towards texture over shape in their decision making processes. In the same way, Ilyas et al. [13] said that adversarial vulnerability happens when networks use predictive but not very robust characteristics that are hidden in noise. Zhou et al. [14] further shown that internal feature maps exhibit heightened noise levels when models are trained absent explicit noise regularisation. These findings underscore the necessity to delineate the evolution of SNR within CNNs, hence associating representational clarity with architectural design. This thesis thus establishes SNR as a conceptual linkage between the principles of physical signal processing and the representations of deep learning. This study seeks to offer a novel, quantitative viewpoint on information transmission and robustness within CNNs by examining the variation in the ratio of relevant to irrelevant activations between layers. This contributes to the overarching objective of ensuring that deep models function effectively while being comprehensible and grounded in reality.

1.2 The Importance of Robustness and Interpretability

Although CNN performance is commonly evaluated using external metrics such as accuracy or F1-score, these measures alone do not reveal how networks internally transform noisy data into meaningful features. In medical imaging, reliability and interpretability are as important as accuracy because model errors can lead to serious diagnostic consequences. Magnetic Resonance Imaging (MRI) and Computed Tomography Scan (CT) data, for instance, are prone to motion artefacts and noise that can obscure lesions or tissue boundaries [15]. Understanding how CNNs process such disturbances is therefore vital for clinical trust and reproducibility. Small perturbations in input data can drastically alter CNN predictions, as demonstrated by Szegedy et al. [16], who revealed the existence of imperceptible input changes that cause confident network misclassifications. Goodfellow et al. [17] later formalised this phenomenon and showed that these vulnerabilities are inherent to linear regions within deep models. Such sensitivity to small perturbations indicates that networks may not be truly learning robust, generalisable representations. These insights have fuelled a growing recognition that robustness is not simply an optimisation issue but a property of how information flows and is encoded within a network. A model that generalises effectively must retain high SNR throughout its layers, filtering out spurious variation while amplifying meaningful structure. Conversely, when networks are shallow, unregularised, or poorly nor-

malised, noise can accumulate, resulting in unstable or overconfident predictions. From this perspective, SNR becomes a diagnostic lens through which the quality of learned features can be objectively measured. Architectural innovations have sought to address such weaknesses. Residual connections allow information to flow directly between layers, mitigating vanishing gradients and facilitating deeper models [3]. Dense connectivity promotes feature reuse and gradient stability by connecting each layer to all preceding layers [18]. Depthwise separable convolutions, as used in Xception, decouple spatial and channel-wise correlations to improve computational efficiency while maintaining representational capacity [19]. EfficientNet introduced a compound scaling rule to jointly scale depth, width, and resolution for optimal accuracy to complexity trade-offs [20]. However, despite these architectural advances, the effect of such mechanisms on SNR dynamics and information retention remains poorly characterised. Robustness studies have primarily focused on external behaviour by examining how accuracy changes under noise, rather than analysing how the internal flow of information degrades or adapts. A more granular understanding requires investigating layer by layer signal quality. Studying SNR evolution therefore offers an interpretable and architecture agnostic way to identify which designs inherently favour information preservation. This perspective addresses a key gap in the literature: the need for a unified quantitative framework connecting architectural structure, information quality, and robustness. Understanding these internal processes could inform the design of architectures that maintain stability in noisy environments which is especially valuable for medical and industrial applications where input quality cannot always be controlled. Furthermore, by linking measurable signal characteristics with abstract representation quality, this research contributes toward interpretable AI, where model transparency complements predictive power.

1.3 Objectives, Approach, and Methodology

The central objective of this thesis is to analyse how CNN architectures process and transform noisy input data and signal through the lens of SNR. The work seeks to answer the following questions:

- How does SNR evolve across successive layers in CNNs of varying architectural complexity?
- Does higher SNR correspond to greater discriminative or fitting power in learned representations?
- How do architectural mechanisms such as residual and dense connections affect SNR propagation?
- How resilient are CNNs to input perturbations when assessed in terms of both SNR and accuracy?

1.3.1 Baseline Modelling and Validation

The study begins with a controlled baseline experiment using an AlexNet inspired architecture [2] trained on the MNIST dataset. This simple and well characterised benchmark provides a foundation for validating the proposed SNR computation. By monitoring SNR across layers and epochs, the experiment establishes a reference pattern for how signal and noise behave in a straightforward feed-forward CNN during training. This makes sure that the measure is reliable and easy to understand before it is used on more complicated designs and datasets. This phase not only serves as a basic validation, but it also provides an opportunity to begin examining how training phenomena, including convergence and overfitting, show up in changes in SNR. These insights help establish whether SNR can act as an early indicator of representation stability.

1.3.2 Comparative Architectural Analysis on Medical Data

The next phase extends the SNR analysis to established deep architectures that represent milestones in CNN evolution: VGG16 and VGG19 [21], ResNet50 and ResNet50V2 [3], DenseNet121 [22], InceptionV3 [23], Xception [19], InceptionResNetV2 [24], and EfficientNetB0 [20]. Each model is trained on a clinical MRI dataset containing segmented brain tumour images, where tumour regions represent signal and surrounding tissue background noise. SNR is calculated layer by layer by isolating activations corresponding to signal and background regions. This lets one compare the quality of information across different architectures in a quantitative way. Linear probe classifiers are trained on intermediate activations to evaluate the separability of classes at each depth. This provides an independent assessment of the extent to which features can be distinguished based on SNR behaviour. These tests assess whether high SNR representations consistently result in enhanced class separability and whether certain designs inherently excel at eliminating noise while preserving critical information.

1.3.3 Noise Injection and Robustness Evaluation

To enhance comprehension of model robustness, we introduce Gaussian noise of differing intensities to the input data. We analyse the effects of these alterations on classification accuracy and the SNR of each layer. This experiment determines whether robustness is derived by maintaining a high SNR, altering the arrangement of noisy information, or augmenting representational capacity. By examining SNR degradation trends, one can identify the network elements largely responsible for information loss or stability amidst noise. One may easily evaluate the efficacy of a structure in preserving significant signals by comparing the SNR decay rates of various designs under identical noise levels. This is a definitive method to assess the robustness of an architecture.

1.3.4 Significance of the Approach

This approach correlates signal processing principles with practical deep learning analysis by quantifying SNR propagation through layers. It enables the identification of variations in information clarity within a network, elucidating the reasons for strengths or weaknesses. This analytical paradigm enhances theoretical comprehension and provides practical instruments for assessing CNNs trained on real-world noisy datasets. Also, it gives researchers a systematic means to test ideas about how architecture robustness affects interactions, which will help them in their future work on noise aware or self-regularising network designs.

1.4 Thesis Structure

The remainder of this thesis is organised as follows:

- **Chapter 2** presents essential principles of machine learning and deep learning, encompassing biological and artificial neural networks, activation functions, supervised learning, and convolutional architectures.
- **Chapter 3** delineates the categories of noise examined in this work, encompassing class, attribute, and injected noise. This chapter formulates the SNR metric employed in the experiments.
- **Chapter 4** implements the proposed methodology on an AlexNet-like model trained on MNIST, confirming the SNR computation and illustrating its progression over layers and epochs.
- **Chapter 5** expands the analysis to intricate structures and investigates the relationship among architectural design, SNR propagation, and robustness against Gaussian noise disruptions.
- **Chapter 6** concludes the thesis by integrating key findings, analysing theoretical and practical implications, and proposing future research directions for noise aware deep learning models.

In summary, this thesis introduces a quantitative framework for analysing information propagation in CNNs through layer-wise SNR measurements. By systematically evaluating how SNR evolves across architectures and under controlled noise perturbations, the work provides new insight into the relationship between network design, robustness, and internal representation quality, particularly in noisy medical imaging contexts.

Chapter 2

Machine Learning and Deep Learning Concepts

2.1 Introduction

In the rapidly evolving field of AI, understanding the structure of neural networks is crucial for optimising their capabilities. This chapter provides a comprehensive analysis of various neural network topologies, with a particular focus on CNNs, which have revolutionised image processing and recognition applications. Biological neural networks are first examined to motivate the design of artificial ones, focusing on their structure and functionality. The role of activation functions, such as Sigmoid [25], hyperbolic tangent (Tanh) [26], Rectified Linear Unit (ReLU) [25], Leaky Rectified Linear Unit (Leaky ReLU) [26], and Softmax [25], is then discussed in relation to learning and performance. Supervised machine learning is subsequently introduced, differentiating between classification and regression problems, followed by an analysis of how neural networks address these tasks. The feed-forward and backpropagation processes are examined, as they are crucial for understanding how networks acquire knowledge and improve performance over time. CNNs are then considered in detail, encompassing their convolutional and pooling layers. Prominent CNN architectures are reviewed, highlighting their contributions and advancements in the field. Analysing these models provides insights into the progression of CNNs and their influence on modern deep learning applications. This chapter seeks to establish a robust basis in neural network designs, preparing for more sophisticated discussions and applications in later chapters.

2.2 Biological Neural Networks

The human brain is an organ that governs numerous cognitive activities. The nervous system facilitates the reception, integration, and transmission of information, with the brain functioning as the focal hub of these processes. The brain consists of a complex network of over 100 billion neurons that enable learning and adaptation to new information [27].

Each neuron consists of three main parts: the cell body (or soma), dendrites, and the axon as illustrated in Figure 2.1. The soma constitutes the central component of the neuron, housing the nucleus and many specialised organelles. It is encased in a membrane that safeguards it and facilitates interaction with its environment. The axon is an elongated, tail-like structure that links to the soma at the axon hillock. The axon is encased in a lipid material known as myelin, which facilitates the effective conduction of electrical signals. Dendrites are filamentous extensions emanating from the cell body. Dendrites receive and process signals from adjacent neurons via synapses, commonly located on the dendrites or cell body, whilst the axon conveys messages to the synapses of neighbouring neurons. Neurons may possess many sets of dendrites, referred to as dendritic trees [27]. Neurons send signals using action potentials which is a change in the neuron's potential electric energy caused by the flow of charged particles in and out of the membrane. When an action potential is created, it is transported along the axon to the presynaptic ending [28]. This is visualised in Figure 2.1.

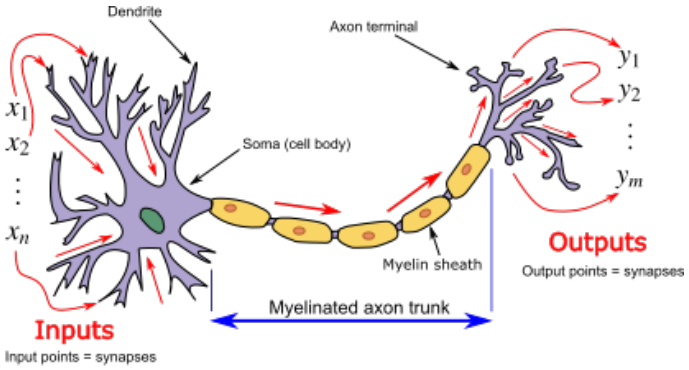

Figure 2.1: Diagram of a biological neuron.[1]

2.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) are designed to emulate the information processing capabilities of the human brain [29]. They consist of three essential components: layers, neurons (or nodes), and connections. The input layer consists of neurons that obtain values from input variables. The layers that follow, called hidden layers, take inputs from their preceding layers. This means that the output from each layer serves as the input for the next layer, continuing until the final output layer is reached [29]. In Figure 2.2, neurons are represented as circles, and their interconnections are illustrated as lines. Each link is allocated a value referred to as a weight, which operates analogously to coefficients in mathematical equations. ANNs learn by repeatedly adjusting these weights to optimise the model's performance. Figure 2.2 shows an example of an ANN with an input layer, one hidden layer, and an output layer. For simplicity, weights are represented by black dots. Here the ANN has two outputs

[1]Image source: *Biological neuron model - Wikipedia*, `https://en.wikipedia.org/wiki/Biological_neuron_model`.

meaning that the ANN is built to classify two classes, or binary classes. The number of outputs represents the number of classes that a particular ANN is discriminating.

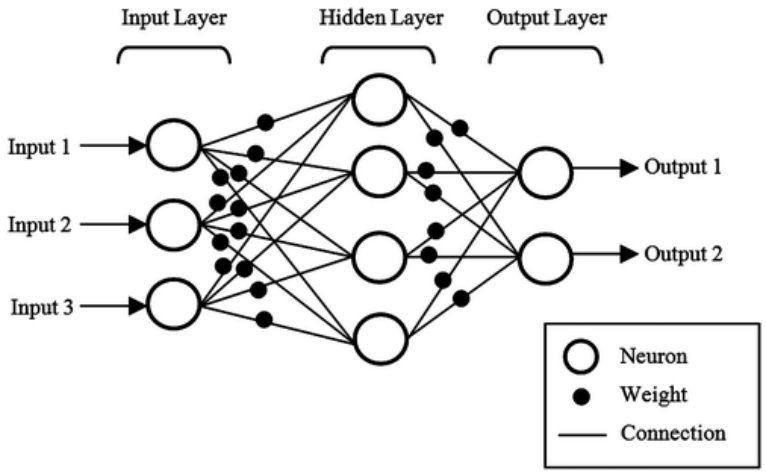

Figure 2.2: An Artificial Neural Network with one hidden layer.

Activation functions are implemented in each hidden layer and at the output layer of an ANN, and they can yield either linear or nonlinear outputs. As illustrated in Figure 2.3, the input to an activation function, σ, is the sum of the weighted inputs plus a bias term, where $\boldsymbol{x}^T = [x_1, x_2, \ldots, x_N]^T$ represents the input, $\boldsymbol{w} = [w_1, w_2, \ldots, w_N]$ denotes the weights, and $\boldsymbol{b} = [b_1, b_2, \ldots, b_N]$ signifies the biases. The bias helps adjust the output by shifting the activation function either positively or negatively. Activation functions are essential for incorporating nonlinearity into the model, allowing it to discern intricate patterns and correlations within the data. Common activation functions encompass ReLU, which alleviates the "vanishing gradient" issue, and Sigmoid or Tanh, which are employed for their distinct characteristics in various circumstances. The choice of activation function at the output layer is contingent upon the model's intended output. A linear activation function is appropriate for regression tasks with continuous outputs, but logistic functions such as Sigmoid or Softmax are optimal for producing probabilistic outputs in classification tasks. The selection of activation function can profoundly influence the training dynamics and overall efficacy of the network.

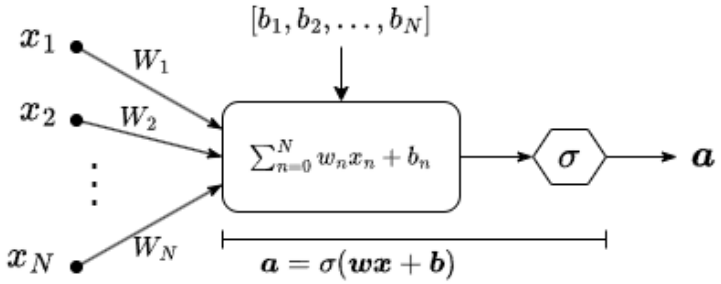

Figure 2.3: An artificial neuron.

2.4 Activation Functions

Activation functions transform the input signal of each layer into an output signal, which then acts as the input for the following layer in ANNs. In the absence of activation functions, the output of each layer would merely be a linear function, severely limiting the network's capacity to learn and discern intricate characteristics within the input. Neural networks are acknowledged as universal function approximators [25]. Activation functions are necessary to add nonlinearity, which lets the network see complex patterns and show complex, non-linear mappings. By including nonlinear activation functions, neural networks achieve the requisite dynamism and capability to extract complex information from input data. Also, activation functions need to be differentiable, or at least substantially differentiable, to make backpropagation optimisation work. This is how mistakes and losses are used to assess the network's performance. The most commonly utilised activation functions include Sigmoid, Tanh, ReLU, Leaky ReLU, and Softmax.

2.4.1 Sigmoid

The Sigmoid function transforms values into a range between 0 and 1 [25]. Although historically popular, it is less commonly used today due to its tendency to cause the "gradient vanishing" problem. When values are transformed to be near 0 or 1, the derivative of the Sigmoid function becomes very small, which impedes learning as the weights do not update efficiently during backpropagation. Additionally, the exponential term e^{-x} in the Sigmoid function is computationally expensive, particularly for larger inputs, which further contributes to its decline in use in favour of more efficient alternatives. The Sigmoid activation function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.1}$$

The derivative of the Sigmoid function is defined as:

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}. \tag{2.2}$$

Figure 2.4 illustrates the Sigmoid function and its derivative.

2.4.2 Tanh

This activation function refers to the hyperbolic tangent (Tanh) function [26]. It is similar to the Sigmoid function, but it transforms values into a range between -1 and 1. Tanh is often preferred over the Sigmoid activation function because its outputs are centered around zero. However, much like the Sigmoid function, while Tanh has this advantage, it still tends to suffer from the "gradient vanishing" problem and is computationally expensive due to the calculations of e^{-x} and e^{x} [26]. The Tanh activation function is defined by:

$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.3}$$

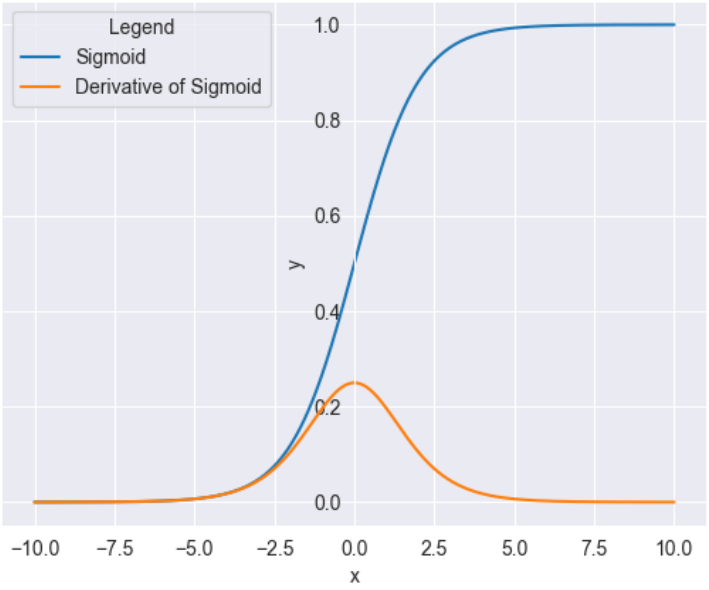

Figure 2.4: The Sigmoid Activation Function.

The derivative of the Tanh function is:

$$\sigma'(x) = \frac{4}{(e^x + e^{-x})^2}. \tag{2.4}$$

Figure 2.5 illustrates the Tanh function and its derivative.

2.4.3 ReLU

ReLU, an abbreviation for Rectified Linear Unit [25], is the most widely used activation function in deep learning applications. It is an approximately linear function, differentiable at all points except at zero, where it is conventionally considered to be zero for computational purposes. The ReLU activation function offers several advantages, including simple and fast computation, the ability to output a true zero value instead of approximate zero, and linear behaviour, which makes it easy to optimise [30]. ReLU also avoids the "vanishing gradient" problem common with Sigmoid and Tanh for positive values due to its non-saturating nature. However, a primary disadvantage is that all negative values are mapped to zero, which may not suit all use cases and can lead to the "dying ReLU" problem. This occurs when neurons receive only negative inputs, causing their outputs to be perpetually zero, which can lead to the network struggling to fit to the data. Variants like Leaky ReLU [26] address this by allowing small, nonzero outputs for negative inputs, keeping neurons active. Additionally, ReLU outputs are not zero centered, which can lead to imbalance in weight updates. The

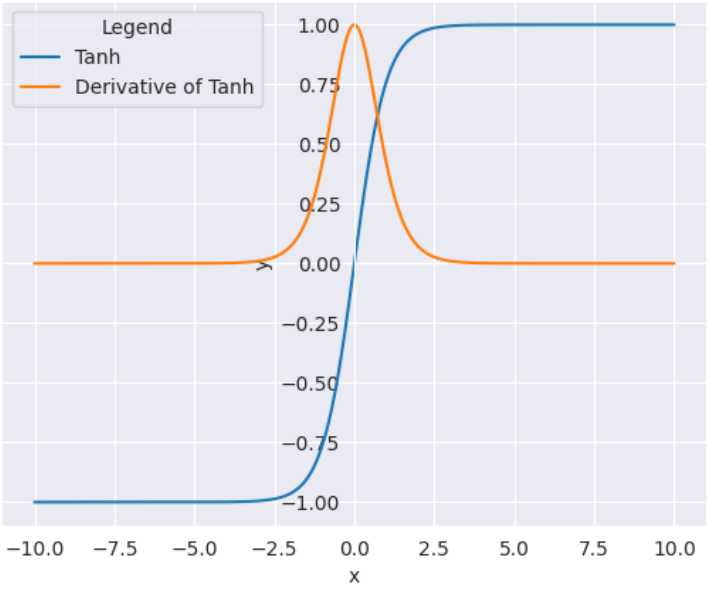

Figure 2.5: The Tanh Activation Function.

ReLU activation function is defined by:

$$\begin{aligned}\sigma(x) &= \max(0, x) \\ &= \begin{cases} 0 & \text{if } x_i \leq 0 \\ x_i & \text{if } x_i > 0 \end{cases}.\end{aligned} \tag{2.5}$$

The derivative of the ReLU function is:

$$\sigma'(x) = \begin{cases} 0 & \text{if } x_i \leq 0 \\ 1 & \text{if } x_i > 0 \end{cases}. \tag{2.6}$$

Figure 2.6 illustrates the ReLU function and its derivative.

2.4.4 Leaky ReLU

Leaky ReLU (LReLU) modifies the ReLU function by introducing a small slope for negative values. The coefficient for this slope, α, is predefined before the training of a neural network, typically set to a small value like 0.01, though it can also be tuned as a hyperparameter. This adjustment aims to address the "dying ReLU" problem [26] by maintaining activity in neurons that would otherwise output zero. Although Leaky ReLU often does not result in significant performance improvements compared to ReLU, it can prevent neurons from becoming inactive, which may improve convergence in certain models. A related variant, Parametric ReLU (PReLU) [31], allows the slope α to be learned during training, offering greater flexibility. The LReLU activation function is defined by:

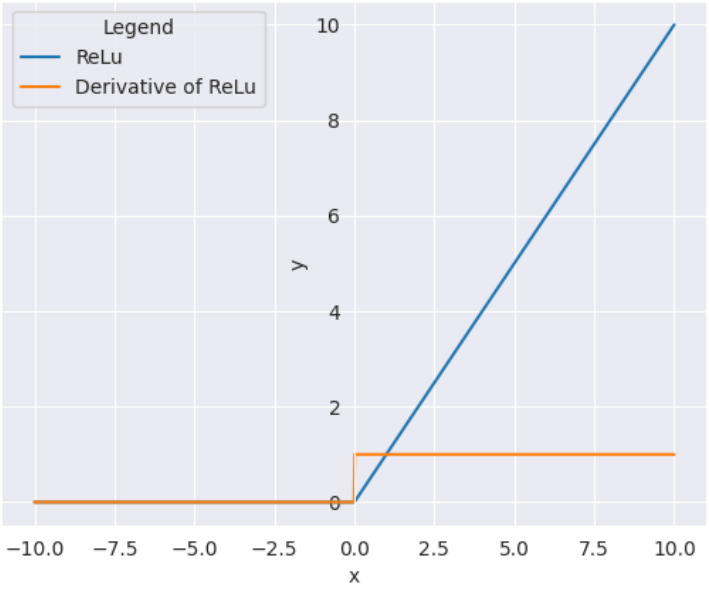

Figure 2.6: The ReLU Activation Function.

$$\begin{aligned}\sigma(x) &= \max(\alpha x, x) \\ &= \begin{cases} \alpha x, & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}\end{aligned} \tag{2.7}$$

The derivative of the LReLU function is:

$$\sigma'(x) = \begin{cases} \alpha & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}. \tag{2.8}$$

Figure 2.7 illustrates the LReLU function and its derivative, with $\alpha = 0.1$.

2.4.5 Softmax

The Softmax activation function converts the input into a probability distribution, ensuring that the output values fall between 0 and 1, with their sum equaling 1. It achieves this by exponentiating the inputs and then normalising them by the sum of the exponentials. Consequently, it is frequently employed in the final layer of a neural network for multiclass classification tasks, where the probabilities indicate the likelihood of each class [25]. The output can then be used to evaluate the model's performance, often in conjunction with the cross-entropy loss function to optimise classification accuracy [30]. Due to its ability to handle multiple classes effectively, Softmax is commonly utilised in tackling multiclass

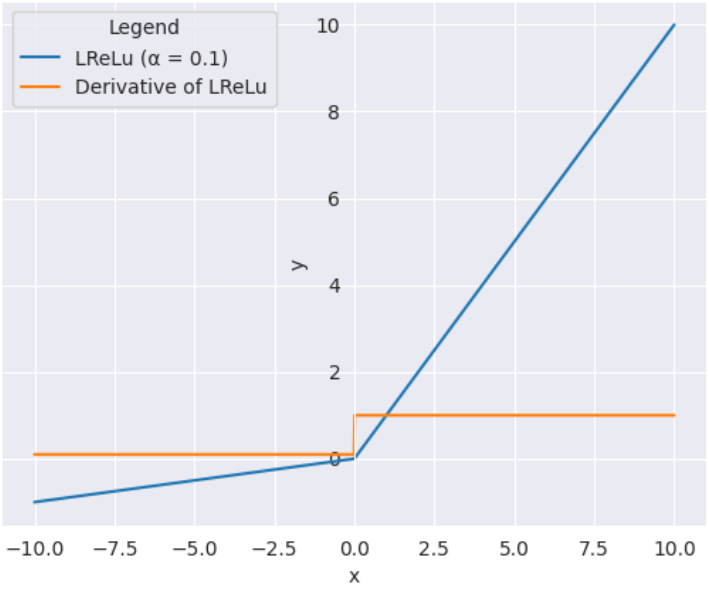

Figure 2.7: The LReLU Activation Function.

classification problems. The Softmax activation function is defined by:

$$\sigma(\boldsymbol{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{N} e^{x_k}}, j = 1, 2, \cdots, N, \tag{2.9}$$

where $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$. Let $S(x_j)$ denote the Softmax function applied to the j-th element of $\boldsymbol{x}$. The derivative of the Softmax function with respect to the k-th input, x_k is given by:

$$\frac{\partial S(x_j)}{\partial x_k} = S(x_j) \times (\delta_{jk} - S(x_k)), \tag{2.10}$$

where δ_{jk} is the Kronecker delta, which is 1 if $j = k$ and 0 otherwise [30]. Figure 2.8 illustrates the Softmax function.

2.5 Supervised Machine Learning

Let $\mathcal{X}$ denote the input space where each vector $\boldsymbol{x}_i = \{x_1, x_2, \ldots, x_n\} \in \mathcal{X}$ and n represents the number of features for each observation. In supervised learning, each observation $\boldsymbol{x}_i$ is associated with a unique output variable $y_i \in \mathcal{Y}$. For classification problems, $\mathcal{Y}$ may be defined as $\{c_1, c_2, \ldots, c_j\}$ where each c_j corresponds to a distinct class, or for regression problems, $\mathcal{Y} \in \mathbb{R}$. This establishes a domain of N input-output pairs defined as:

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}, 1 \leq i \leq N. \tag{2.11}$$

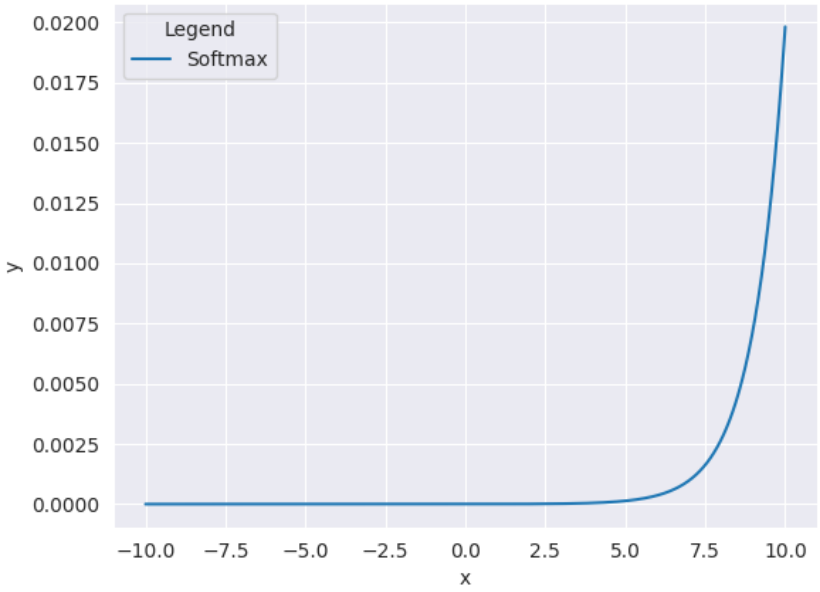

Figure 2.8: The Softmax Activation Function.

Assume one has a family of parametric functions, $\mathcal{F}$ from which the model is chosen. The objective of supervised machine learning is to determine a function (or model) $f \in \mathcal{F}$ that predicts y_i from a given $\boldsymbol{x}_i$,

$$\begin{aligned} f : \mathcal{X} &\to \mathcal{Y} \\ \boldsymbol{x}_i &\mapsto \hat{y}_i \end{aligned} \tag{2.12}$$

and presents the expected risk

$$\mathcal{R}(f) = E_p[f, \mathcal{L}, \mathcal{X}, \mathcal{Y}] \tag{2.13}$$

the smallest, where $\mathcal{L}$ is the loss function and p is the unknown distribution. Since $\mathcal{R}(f)$ is theoretical during training, the empirical risk is instead minimised in the place of the expected risk:

$$\mathcal{R}_{\mathcal{D}}(f) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(x_i, y_i, f). \tag{2.14}$$

The empirical risk, $\mathcal{R}_{\mathcal{D}}$ is an optimistic estimate of the expected risk, $\mathcal{R}(f)$. Assume $f^*_{\mathcal{D}}$ is the model that minimises the empirical risk, $\mathcal{R}_{\mathcal{D}}(f^*_{\mathcal{D}})$, and f^* is the model that minimises the expected risk, $\mathcal{R}(f^*)$. Theoretically, there exists a threshold model, f_{min}, that also minimises the expected risk, ie.:

$$\mathcal{R}(f_{min}) \leq \mathcal{R}(f^*). \tag{2.15}$$

We can write:

$$\mathcal{R}(f^*_{\mathcal{D}}) = \mathcal{R}(f_{min}) + [\mathcal{R}(f^*) - \mathcal{R}(f_{min})] + [\mathcal{R}(f^*_{\mathcal{D}}) - \mathcal{R}(f^*)] \tag{2.16}$$

where $\mathcal{R}(f^*) - \mathcal{R}(f_{min})$ is the approximation error which is theoretical since one cannot calculate $\mathcal{R}(f_{min})$ and $\mathcal{R}(f^*)$ practically. However, it is minimal since f^* is the error that minimises the expected risk. $\mathcal{R}(f^*_{\mathcal{D}}) - \mathcal{R}(f^*)$ is the estimation error and should be zero or at least very close to zero if the model that minimises the empirical risk is also the model that minimises the expected risk. Since $\mathcal{R}(f^*_{\mathcal{D}})$ and $\mathcal{R}(f^*)$ are theorectical, in practice they are approximated by $\mathcal{R}_{\mathcal{D}}(f^*_{\mathcal{D}})$ as:

$$\begin{aligned} \lim_{N\to\infty} \mathcal{R}_{\mathcal{D}}(f^*_{\mathcal{D}}) &= \mathcal{R}(f^*) \\ &= \mathcal{R}(f^*_{\mathcal{D}}) \end{aligned} \tag{2.17}$$

since the model is generalising, $\mathcal{R}(f^*_{\mathcal{D}}) - \mathcal{R}(f^*) = 0$ or ≈ 0. In practice, $\mathcal{R}(f^*_{\mathcal{D}})$ is approximated using the model error on the test dataset while $\mathcal{R}_{\mathcal{D}}(f^*_{\mathcal{D}})$ is approximated by the error of the model on the training dataset.

There are several different loss functions to choose from to suit their specific needs.

2.5.1 Loss for Classification Problems

In the following sections, let N denote the number of samples, M denote the number of output classes, y_{ij} denote the true class for sample i, and $\hat{y}_{ij}$ denote the predicted class/probability distribution for sample i for a classification problem.

Cross-Entropy Loss

The Cross-Entropy Loss (or Log Loss) is the most popular loss function used for classification problems. It compares the predicted probability distribution to the expected or true class and then penalises based on the magnitude of deviation [32]. It is defined in Equation 2.18.

$$\mathcal{L} = -\frac{1}{N} \sum_{i}^{N} \sum_{j}^{M} y_{ij} \log(\hat{y}_{ij}). \tag{2.18}$$

Figure 2.9 shows the cross-entropy.

Hinge Loss

An alternative to the Cross-Entropy Loss is the Hinge Loss. This loss function aims to penalise wrong predictions, as well as predictions that are not confident. However, this particular loss function was primarily created for Support Vector Machines (SVM) [33] that usually have their class labels as -1 and 1 [32]. The Hinge Loss is defined in Equation 2.19.

$$\mathcal{L} = max(0, 1 - y_{ij}\hat{y}_{ij}). \tag{2.19}$$

Figure 2.10 shows the Hinge Loss function.

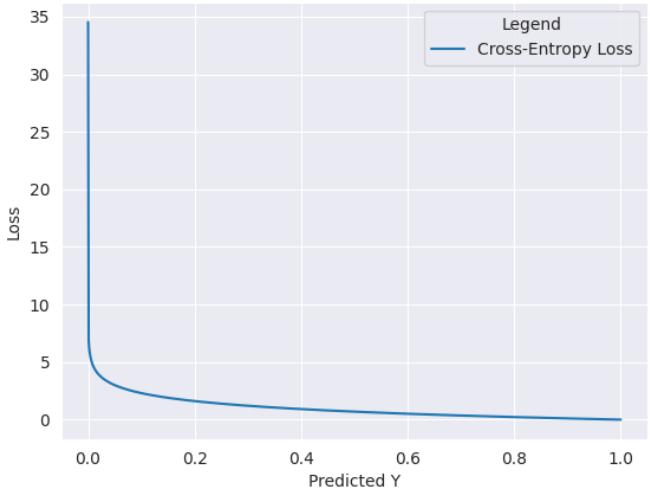

Figure 2.9: The Cross-Entropy Loss Function.

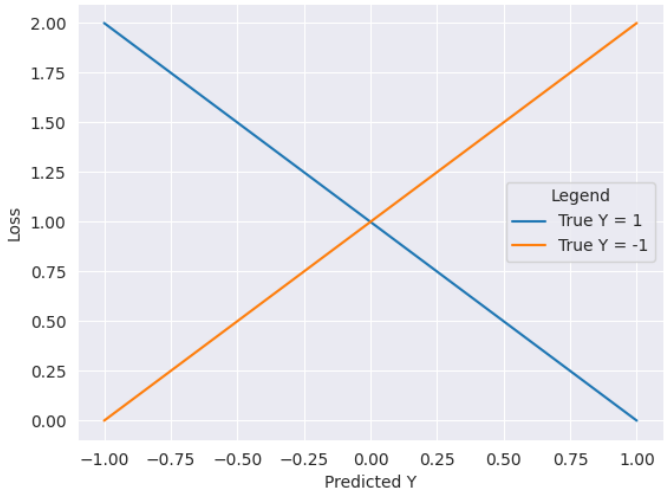

Figure 2.10: The Hinge Loss Function.

2.5.2 Loss for Regression Problems

Mean Square Error Loss

The Mean Square Error (MSE), also known as Quadratic Loss or L2 Loss, is a commonly used loss function for regression problems. It is calculated by taking the average of the squared differences between the actual and predicted values. This characteristic means that large errors are heavily penalised because they are squared, making the MSE very sensitive to outliers in the dataset. The MSE loss function is defined:

$$\mathcal{L} = \frac{1}{N}\sum_{n}^{N}(\hat{y}_n - y_n)^2. \tag{2.20}$$

Figure 2.11 shows the Mean Square Error Loss.

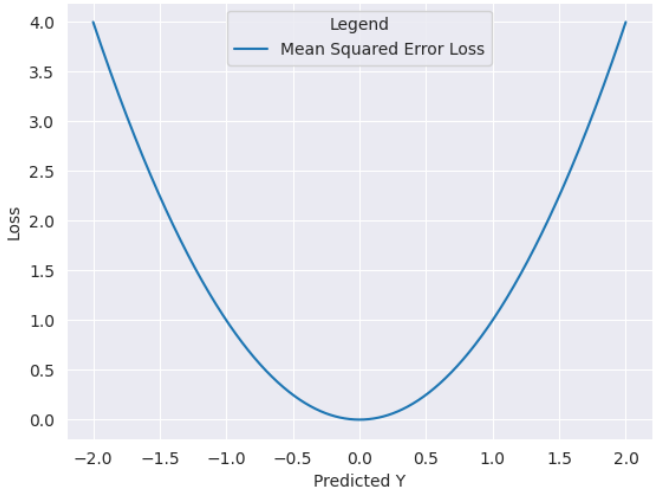

Figure 2.11: The MSE Loss Function.

Mean Absolute Error Loss

An alternative to the MSE is the Mean Absolute Error (MAE), also known as the L1 loss. It is the average of the absolute deviation between the actual and predicted values. The MAE is not as sensitive to outliers as the MSE and is therefore chosen for datasets prone to outliers. However, a drawback of the MAE is that it is non-differentiable at 0, when the predicted value and actual value are equal. The MAE loss function is defined as:

$$\mathcal{L} = \frac{1}{N}\sum_{n}^{N}|\hat{y}_n - y_n|. \tag{2.21}$$

Figure 2.12 shows the Mean Absolute Error Loss.

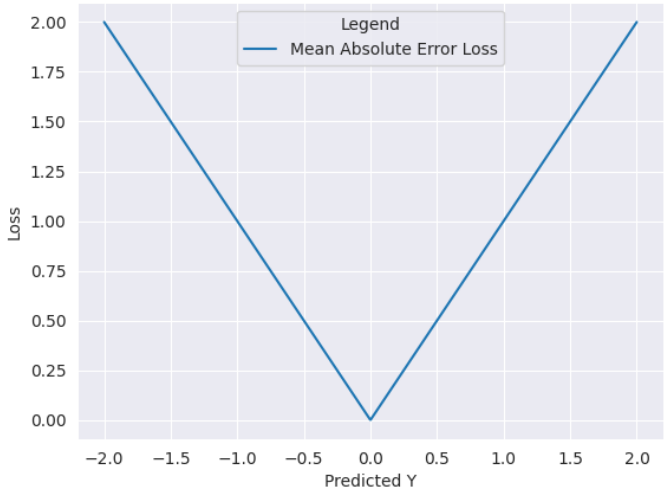

Figure 2.12: The MAE Loss Function.

2.6 The Feed-Forward Process

As discussed in Section 2.3, an ANN learns by repeatedly updating each neuron's weight and bias until the performance reaches an optimal state. Consider the training dataset $(\boldsymbol{X}, \boldsymbol{y})$. Calculations, as defined by Equation 2.22, are performed on $\boldsymbol{X}$ within every neuron of each layer in the ANN. The output of the calculation of each layer serves as the input for the subsequent layer until finally $\hat{\boldsymbol{y}}$, the prediction of $\boldsymbol{y}$, is produced by the last layer. The weights and biases of the first layer, $\boldsymbol{W}^1$ and $\boldsymbol{b}^1$, are initialised and used to calculate $\boldsymbol{Z}^1$, the linear combination of inputs defined by $\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1$. This $\boldsymbol{Z}^1$ is then passed through an activation function, σ, to obtain $\boldsymbol{y}^1$,which serves as the input for the second layer. This process is repeated in all layers [34]. The output of the final layer, L, is $\boldsymbol{y}^L$ is then $\hat{\boldsymbol{y}}$. The mathematical representation of the feed-forward process is defined as follows:

$$\hat{\boldsymbol{y}}_{\boldsymbol{k}} = \boldsymbol{y}_k^L = \sigma(\boldsymbol{W}_k^L \boldsymbol{y}_i^{L-1} + \boldsymbol{b}_k^L), \tag{2.22}$$

where $\boldsymbol{W}_k^L$ denotes the normalised weight that links neurons of layer $L-1$ to neuron k of layer L, $\boldsymbol{y}_i^{L-1}$ is the output of neuron i of layer $L-1$ after applying the activation function, and $\boldsymbol{b}_k^L$ is the bias of neuron k of layer L. The prediction $\hat{\boldsymbol{y}}$, together with the actual $\boldsymbol{y}$ can be used to calculate the empirical risk, outlined in Section 2.5, denoted by $\mathcal{R}_D$. This is outlined in more detail in Section 2.7. The feed-forward process can be visually represented in Figure 2.13.

2.7 The Backpropagation Process

To evaluate the precision of a particular iteration of the feed-forward process in predicting $\boldsymbol{y}$, the backpropagation process takes $\hat{\boldsymbol{y}}$ and passes it into a loss function, $\mathcal{L}(.)$, defined

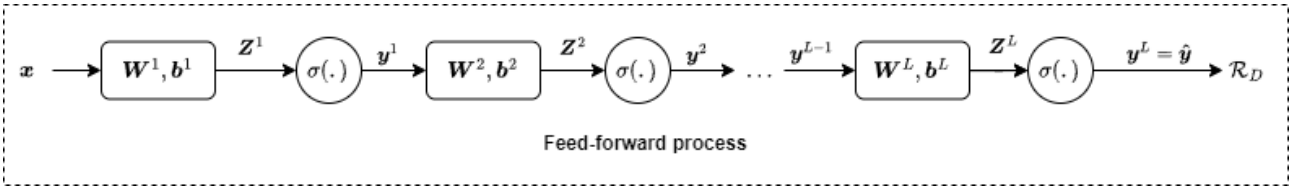

Figure 2.13: The Feed-Forward Process.

in Section 2.5, to calculate the risk, $\mathcal{R}_D$. Using the gradient descent algorithm, successive differentiation calculations are performed with respect to the weights and biases of each layer [34]. This is expressed as

$$\frac{\partial \mathcal{R}_D}{\partial \boldsymbol{W}^L_{k,j}} = \frac{\partial \mathcal{R}_D}{\boldsymbol{y}^L_i} \frac{\partial \boldsymbol{y}^L_i}{\partial \boldsymbol{Z}^L_i} \frac{\partial \boldsymbol{Z}^L_i}{\partial \boldsymbol{W}^L_{k,j}}. \tag{2.23}$$

When applying the chain rule, we can simplify this further:

$$\begin{aligned} \frac{\partial \mathcal{R}_D}{\partial \boldsymbol{W}^L_{k,j}} &= \frac{\partial \sum_{q=1}^{N_{L-1}} (\boldsymbol{W}^L_{k,q} \boldsymbol{y}^{L-1}_q + \boldsymbol{b}^L_k)}{\partial \boldsymbol{W}^L_{k,j}} \\ &= \boldsymbol{y}^{L-1}_j \end{aligned} \tag{2.24}$$

$$\frac{\partial \boldsymbol{y}^L_i}{\partial \boldsymbol{Z}^L_i} = \sigma^1(\boldsymbol{Z}^L_i) \tag{2.25}$$

and knowing that $\boldsymbol{y}^L_i = \hat{\boldsymbol{y}}_i$,

$$\frac{\partial \mathcal{R}_D}{\partial \boldsymbol{y}^L_i} = -(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i). \tag{2.26}$$

Therefore, we can write:

$$\begin{aligned} \frac{\partial \mathcal{R}_D}{\partial \boldsymbol{W}^L_{k,j}} &= -(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)\sigma(\boldsymbol{Z}^L_i) \\ &= \boldsymbol{y}^{L-1}_j. \end{aligned} \tag{2.27}$$

The same differentiation strategy can be applied to the biases. The weights and biases are then updated using the following:

$$\boldsymbol{W}^L_{t+1} = \boldsymbol{W}^L_t - \lambda \Delta_t \boldsymbol{W}^L \tag{2.28}$$

$$\boldsymbol{b}^L_{t+1} = \boldsymbol{b}^L_t - \lambda \Delta_t \boldsymbol{b}^L \tag{2.29}$$

where λ represents the learning rate, $\Delta_t \boldsymbol{W}^L$ and $\Delta_t \boldsymbol{b}^L$ denote the differentiation with respect to all the weights and biases at iteration t. The backpropagation process is visualised in Figure 2.14.

2.8 Convolutional Neural Networks (CNN)

CNNs are highly adaptable neural networks capable of addressing various pattern recognition challenges. Similar to traditional ANNs, CNNs consist of neurons that self-optimise

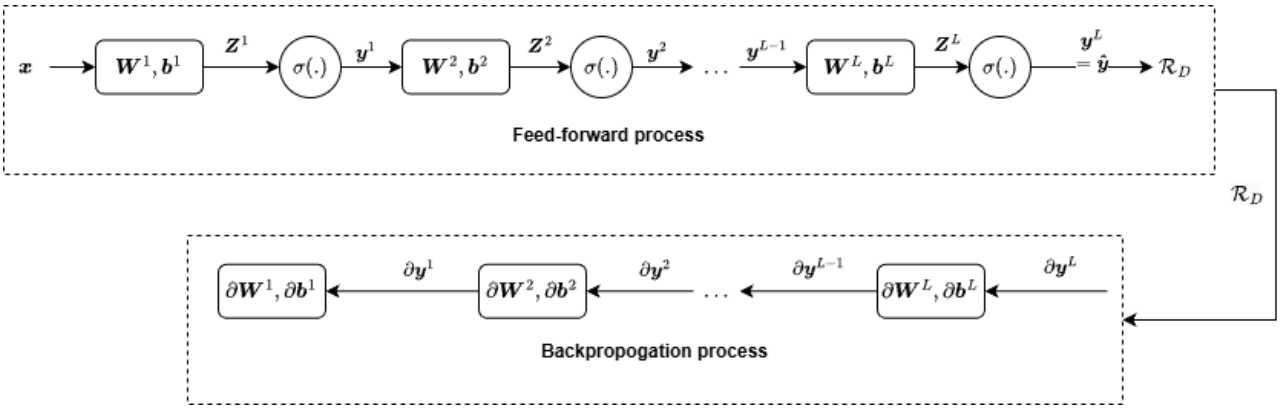

Figure 2.14: The Backpropagation Process.

through the feed-forward and backpropagation algorithms described in Sections 2.6 and 2.7. However, CNNs are particularly effective for pattern recognition in image data, overcoming the limitations that traditional ANNs encounter with high-dimensional inputs such as images [35].

The overall architecture consists of three distinct types of layers: convolutional layers, pooling layers, and fully connected layers. The input layer holds the pixel values for each image. The convolutional layers compute the output of neurons connected to specific regions of the input by calculating the dot product between their weights and the corresponding region. Pooling layers downsample along the spatial dimensions of the input, reducing the number of features within the activation. Finally, the fully connected layers, which form the ANN portion of this architecture, generate predicted classes from the activations for classification purposes [35].

2.8.1 Convolution Layer

A convolutional layer uses a kernel to extract features from the input. This kernel can be thought of as a window that is slid across the input image and multiplied with the values inside the window visible to the kernel to enhance features encompassed by that portion of the input [36]. Suppose the input for the i^{th} layer in the CNN is a three-dimensional matrix with dimensions $N^i \times M^i \times D^i$. The convolutional kernel is also a three-dimensional matrix of size $N \times M \times D^i$. The kernel will then overlap the input matrix at spatial location $(0, 0, 0)$. The products of corresponding elements in all the D^i channels and the sum of the $N \times M \times D^i$ products is calculated to get the convolution result at this spatial location. The kernel is then shifted along the input image to get the next convolutional result at the next spatial location [36]. Once this process is completed across all spatial locations, the convolution is complete. The output of a convolutional layer is called a feature map [35]. Figure 2.15 shows a convolution with a 2×2 filter on an input.

In a convolutional layer, there are usually multiple convolution kernels. Let $\boldsymbol{k}$ be a set of D kernels. Thus, $\boldsymbol{k}$ is four-dimensional: $\boldsymbol{k} \in \mathbb{R}^{N \times M \times D^i \times D}$. As shown in Figure 2.15, the output

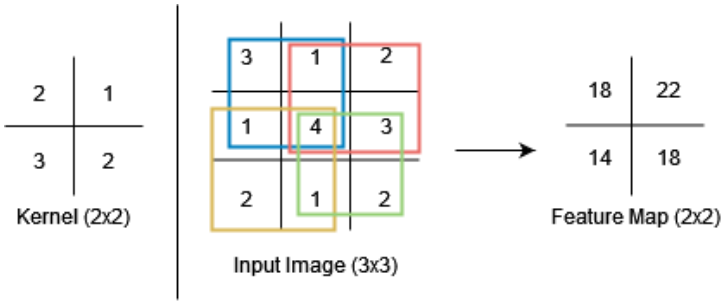

Figure 2.15: Example of a kernel moving through the input image.

from the convolution is smaller than the input image so long that the size of the kernel is more than 1×1. Therefore, if the dimensions of the input image are $N^i \times M^i \times D^i$, and the kernel size is $N \times M \times D^i \times D$, then the convolution output will have size $(N^i - N + 1) \times (M^i - M + 1) \times D$ [36]. In Figure 2.15, the kernel convolves at every spatial location of the input. This corresponds to a stride, denoted s, of 1. If s were to be greater than 1, the kernel would skip $s - 1$ spatial locations for every movement and thus the convolution would be performed once every s pixels horizontally and vertically [36]. In this section we will consider only a stride of 1. Let the output of layer i, with input $\boldsymbol{x}^i$, be denoted as $\boldsymbol{y}^i$. This output is used as the input into the next layer and thus can also be thought of as $\boldsymbol{x}^{i+1}$. Hence, $\boldsymbol{y}^i \in \mathbb{R}^{N^{i+1} \times M^{i+1} \times D^{i+1}}$ with $N^{i+1} = N^i - N + 1$, $M^{i+1} = M^i - M + 1$, and $D^{i+1} = D$ [36]. Therefore, the convolution procedure is expressed as:

$$\boldsymbol{y}_{n^{i+1},m^{i+1},d} = \sum_{n=0}^{N} \sum_{m=0}^{M} \sum_{d^i=0}^{D^i} \boldsymbol{k}_{n,m,d^i,d} \times \boldsymbol{x}^i_{n^{i+1}+n,m^{i+1}+m,d^i}. \tag{2.30}$$

Equation 2.30 is repeated for all $0 \leq d \leq D = D^{i+1}$, and for any spatial location (n^{i+1}, m^{i+1}) which satisfies $0 \leq n^{i+1} < N^i - N + 1 = N^{i+1}$ and $0 \leq m^{i+1} < M^i - M + 1 = M^{i+1}$ [36]. A bias, denoted as b_d is usually added to $\boldsymbol{y}_{n^{i+1},m^{i+1},d}$ but has been omitted for clarity's sake.

2.8.2 Pooling Layer

Let $\boldsymbol{x}^i \in \mathbb{R}^{N^i \times M^i \times D^i}$ be the input to the i^{th} layer which is a pooling layer. Let the spatial extent of the pooling is denoted as $N \times M$. Assume that N divides N^i and M divides M^i and the stride equals the pooling spatial extent. Therefore, the strides in the vertical and horizontal directions are N and M respectively. Thus, the output of the pooling layer, denoted $\boldsymbol{y}^i$, is an order 3 tensor with dimensions $N^{i+1} \times M^{i+1} \times D^{i+1}$ where $N^{i+1} = \frac{N^i}{N}$, $M^{i+1} = \frac{M^i}{M}$, and $D^{i+1} = D^i$ [36].

A pooling layer operates on each channel of $\boldsymbol{x}^i$ independently. Within a channel, the matrix with $N^i \times M^i$ elements are divided into $N^{i+1} \times M^{i+1}$ non-overlapping sub-regions. Each of these sub-regions are of size $N \times M$ [36]. A sub-region is then mapped to a single number as a result of the pooling operator.

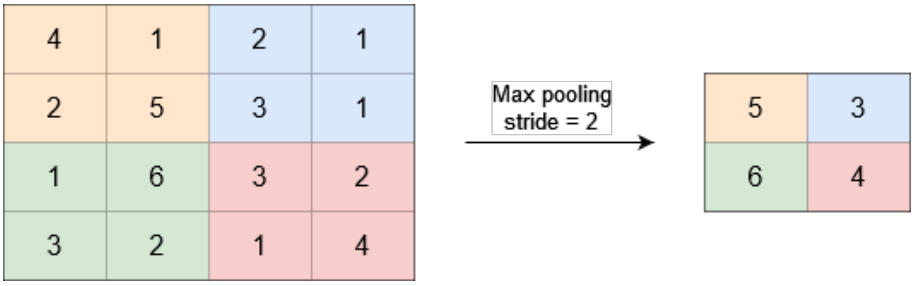

Figure 2.16: Max pooling with stride = 2.

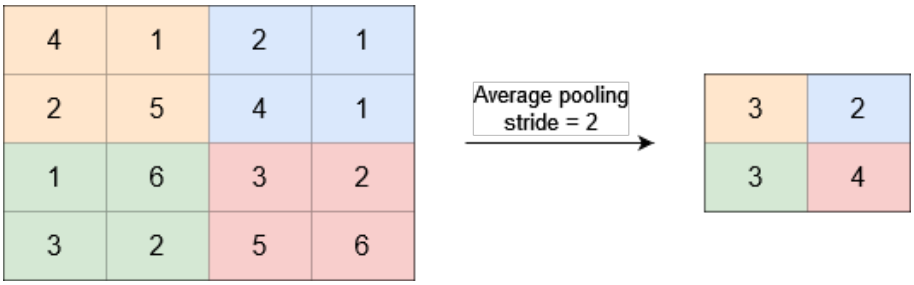

Figure 2.17: Average pooling with stride = 2.

The two types of pooling operators that are used most frequently are max pooling and average pooling. Max pooling maps a sub-region to its maximum value:

$$\boldsymbol{y}_{n^{i+1},m^{i+1},d} = \max_{0 \le n < N, 0 \le m < M} \boldsymbol{x}^i_{n^{i+1} \times N + n, m^{i+1} \times M + m, d}, \tag{2.31}$$

Figure 2.16 depicts max-pooling with stride of 2.

Average pooling maps a subregion to its average value:

$$\boldsymbol{y}_{n^{i+1},m^{i+1},d} = \frac{1}{NM} \sum_{0 \le n < N, 0 \le m < M} \boldsymbol{x}^i_{n^{i+1} \times N + n, m^{i+1} \times M + m, d}, \tag{2.32}$$

Figure 2.17 depicts average-pooling with stride of 2 where $0 \le n^{i+1} < N^{i+1}, 0 \le m^{i+1} < M^{i+1}$, and $0 \le d < D^{i+1} = D^i$ [36].

Max pooling captures the most dominant feature in a region, making it useful for detecting sharp transitions like edges or corners. In contrast, average pooling provides a more stable representation, reducing sensitivity to small shifts in the input image. Max pooling often enhances feature extraction in tasks like object detection, while average pooling helps maintain spatial consistency, which can be beneficial in scenarios where fine details matter, such as texture analysis or medical imaging. The choice depends on whether highlighting key features or preserving overall feature integrity is more important for the task.

2.8.3 Different Architectures of CNNs

CNNs have evolved through a series of increasingly sophisticated architectures, each introduced to overcome the limitations of its predecessors and to improve performance across

a range of computer vision tasks. Over the past decade, research has shifted from shallow models with simple convolutional stacks to deeper and more optimised networks that incorporate new structural components, improve training dynamics, and achieve higher accuracy on large-scale datasets. Each major architecture reflects a step forward in understanding how depth, connectivity, feature reuse, and computational efficiency can be balanced to build more complex neural networks. In this thesis, several of the most recent CNN architectures are explored, namely AlexNet, VGG, ResNet, DenseNet, Inception, Xception, and EfficientNet. These models were selected because they represent key milestones in the evolution of deep learning for image classification. The model developed for the initial experimentation, presented in Chapter 4, was designed to reflect the fundamental structure of AlexNet and served as a baseline for evaluating the effects of SNR on network performance. Subsequent experiments, discussed in Chapter 5, extend this investigation by applying and comparing a range of established architectures including VGG, ResNet, DenseNet, Inception, Xception, and EfficientNet. These models were trained and evaluated under varying noise conditions to assess their robustness and generalisation capability. This transition from a custom baseline model to sophisticated, existing architectures facilitates a thorough evaluation of the impact of architectural complexity and feature reuse on performance in the presence of noisy picture data.

AlexNet

The AlexNet model won the large-scale Visual Recognition Challenge (LSVRC) competition in 2012, marking a significant milestone in the development of deep learning. The model was proposed by Alex Krizhevsky and his colleagues in their paper ImageNet Classification with Deep Convolutional Neural Networks [2]. The network consists of eight trainable layers, including five convolutional and three fully connected layers, arranged to progressively extract and refine features from the input data. A key innovation in AlexNet is the use of the ReLU activation function, applied at the end of all layers except the final one. ReLU helps to combat the "vanishing gradient" problem and enables faster, more effective training. The final layer uses the Softmax activation function to output class probabilities for image classification. To improve generalisation and reduce overfitting, dropout layers [37] are applied to the first two fully connected layers. Max-pooling layers come after the first, second, and fifth convolutional layers. They keep important information while making the feature maps smaller in space. To improve performance and stability, further methods were included, such as GPU acceleration, data augmentation, and local response normalisation (LRN). These design choices made it possible for AlexNet to do large-scale classification jobs with never before seen accuracy and opened the door for deeper, more complicated architectures. The structural configuration of AlexNet is illustrated in Figure 2.18 [2].

VGG (VGG16 and VGG19)

The VGG16 architecture, proposed by K. Simonyan and A. Zisserman [21], is notable for its simplicity, depth, and effectiveness, achieving excellent performance in tasks such as object recognition and image classification. One of the key aspects of VGG16 is its use of very small

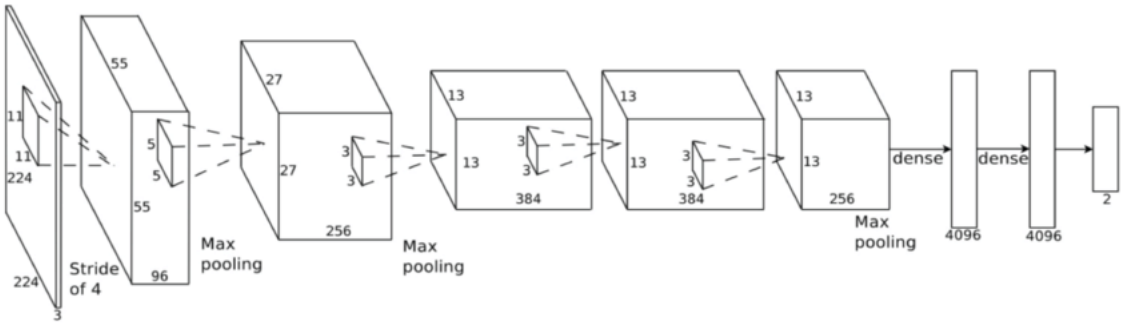

Figure 2.18: AlexNet Architecture: Original image published in paper [2].

(3×3) convolutional filters throughout the network. This design choice enables the model to build deeper architectures than earlier CNNs while keeping the number of parameters manageable and allowing for the extraction of increasingly complex features at each layer. The network consists of 16 trainable layers: thirteen convolutional layers, five max-pooling layers, and three fully connected layers, arranged sequentially to progressively transform input images into high-level feature representations. Despite its strong performance, VGG16 has some limitations. Its depth and large number of parameters lead to long training times and high computational costs, requiring substantial memory and processing power. In addition, it lacks more modern techniques such as batch normalisation, which can improve training stability and convergence speed. Nevertheless, the model's straight forward, uniform structure made it an influential baseline for subsequent CNN architectures and it remains widely used in research and practical applications today. A visual summary of its structure appears in Figure 2.19.

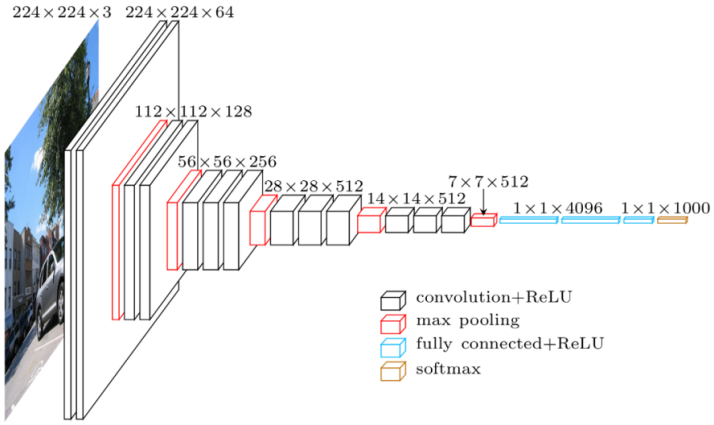

Figure 2.19: VGG16 Architecture: Original image published in paper [38].

A deeper variant, VGG19, follows the same core design principles as VGG16 but extends the network to nineteen trainable layers by adding three additional convolutional layers. The increased depth allows VGG19 to capture more complex hierarchical representations of visual data, which can lead to acceptable performance improvements in certain tasks. However, these gains come at the cost of an even larger number of parameters and longer training times,

further increasing computational demands. Like VGG16, VGG19 retains the strictly uniform architecture based on 3×3 convolutional filters and 2×2 max-pooling layers, demonstrating the continued effectiveness of depth and architectural simplicity in deep CNNs. The general architecture remains consistent with VGG16, as depicted in Figure 2.20.

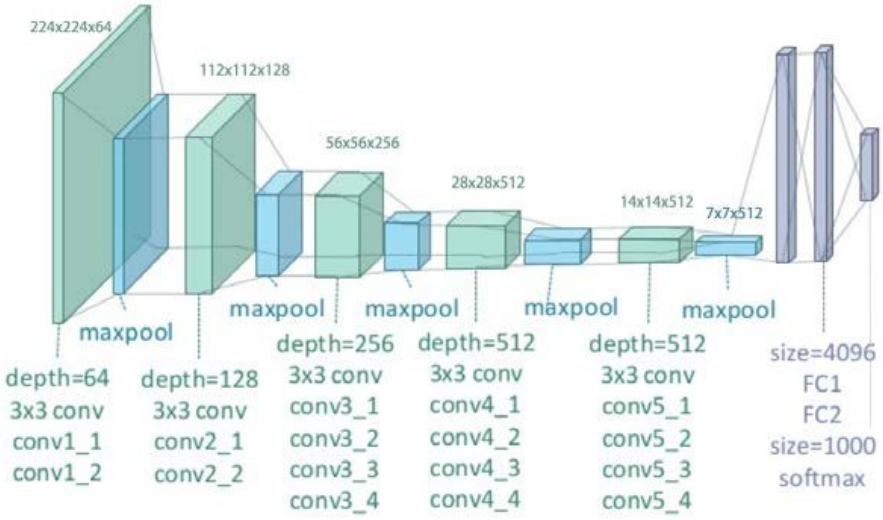

Figure 2.20: VGG19 Architecture: Original image published in paper [39].

ResNet (ResNet50 and ResNet50V2)

Residual Network (ResNet), introduced by Kaiming He et al. [3] in 2015, was a major advancement in deep learning architectures, addressing the "vanishing gradient" problem. As networks became deeper, their performance often plateaued or even degraded because gradients diminished as they propagated backward through many layers. ResNet solved this problem by introducing residual connections, which allow layers to learn residual functions relative to their inputs rather than trying to learn the underlying mapping directly. At the heart of ResNet are residual blocks, shown in Figure 2.21, where the input to a block is added directly to its output. This creates a shortcut, allowing information to bypass certain layers if no significant transformation is required. By doing so, the network preserves important features, improves gradient flow during backpropagation, and stabilises training. This approach mitigates the degradation problem and makes it possible to train networks with hundreds of layers while still improving performance. As a result, ResNet architectures have become widely used for tasks such as image classification, object detection, and feature extraction. In this thesis, the focus is on the 50 layer variant, ResNet50, one of the most widely adopted versions due to its balance of depth, computational efficiency, and predictive performance. ResNet50 consists of a series of stacked residual blocks, each containing multiple convolutional layers followed by batch normalisation and activation functions. Its success has made it a standard backbone in many computer vision models. A later improvement, ResNet50V2 [40], builds on the original design by changing the internal structure of the residual blocks. Instead of applying batch normalisation and ReLU activation after each convolution, known as post activation, ResNet50V2 uses a pre-activation approach, where

normalisation and activation occur before the convolution. This subtle adjustment improves gradient flow, speeds up convergence, and often leads to slightly better generalisation, particularly in deeper networks. Additionally, ResNet50V2 removes the final ReLU activation at the end of the residual branch, which helps preserve more precise feature information across layers. The introduction of residual learning fundamentally changed CNN design, inspiring a new generation of deeper, more robust architectures. ResNet's influence extends beyond classification tasks, and its design principles are now integral to many modern computer vision models. The architecture of a typical residual block and the overall structure of ResNet are shown in Figures 2.21 and 2.22.

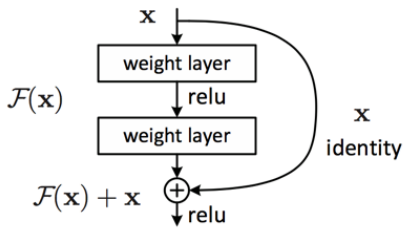

Figure 2.21: Residual Block: Original image published in paper [3].

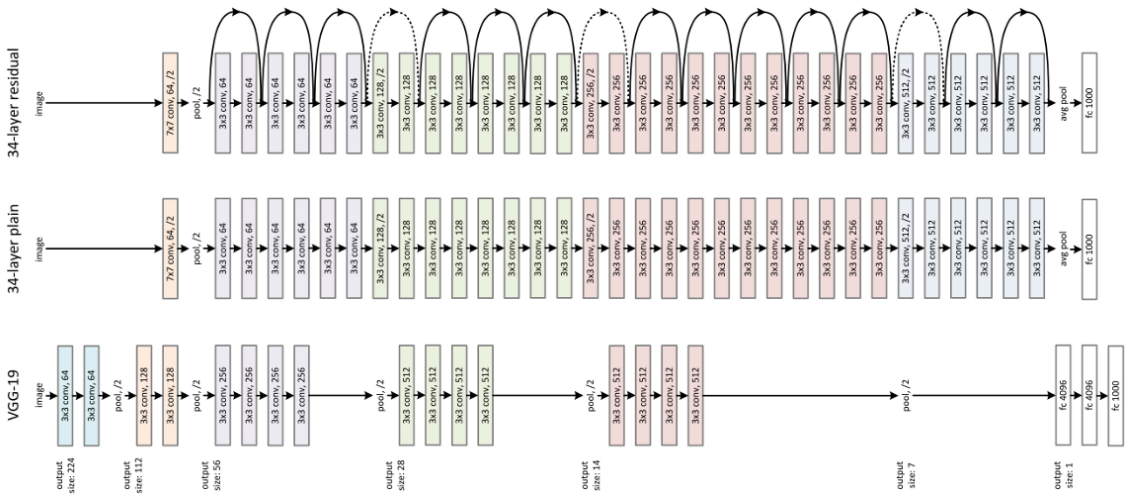

Figure 2.22: ResNet Architecture: Original image published in paper [3].

DenseNet

Dense Convolutional Networks (DenseNet), introduced by Gao Huang et al. [22] in 2017, represent another major evolution in CNN architecture, extending the concept of skip connections introduced by ResNet. While ResNet allows information to bypass certain layers through residual connections, DenseNet takes this idea further by creating dense connectivity between layers: each layer receives the feature maps of all preceding layers as additional

inputs. In other words, instead of summing features as in ResNet, DenseNet concatenates them, ensuring that each layer has direct access to the outputs of all earlier layers. This design offers several key advantages. Firstly, it promotes extensive feature reuse, as earlier feature maps continue to contribute to later computations, reducing redundancy and improving representational efficiency. Secondly, it significantly improves gradient flow during backpropagation, which helps mitigate vanishing gradients and makes training deeper networks more stable and effective. Finally, by leveraging previously computed features rather than relearning them, DenseNet achieves high performance with fewer parameters compared to many architectures of similar depth. One of the most widely used models in this family is DenseNet121, which consists of 121 layers organised into four dense blocks separated by transition layers. Each dense block contains multiple convolutional layers, and because their outputs are concatenated rather than summed, the dimensionality of the feature maps grows progressively deeper into the network. The transition layers, which include 1×1 convolutions followed by 2×2 average pooling, control this growth by reducing spatial dimensions and computational cost while preserving feature richness. A key strength of DenseNet121 is its exceptional parameter efficiency. Despite being very deep and capable of strong performance on complex image classification tasks, it typically requires significantly fewer parameters than comparably deep networks. Moreover, the dense connectivity provides a form of implicit deep supervision, as gradients from the loss function propagate directly to earlier layers, enabling more effective feature learning from shallow to deep stages. The primary trade-off of this design is increased memory usage, since feature maps from all previous layers must be stored for concatenation. Nevertheless, DenseNet121 remains a popular and influential architecture thanks to its strong balance of performance, training stability, and computational efficiency. A pictorial depiction of the architecture of DenseNet121 is provided in Figure 2.23.

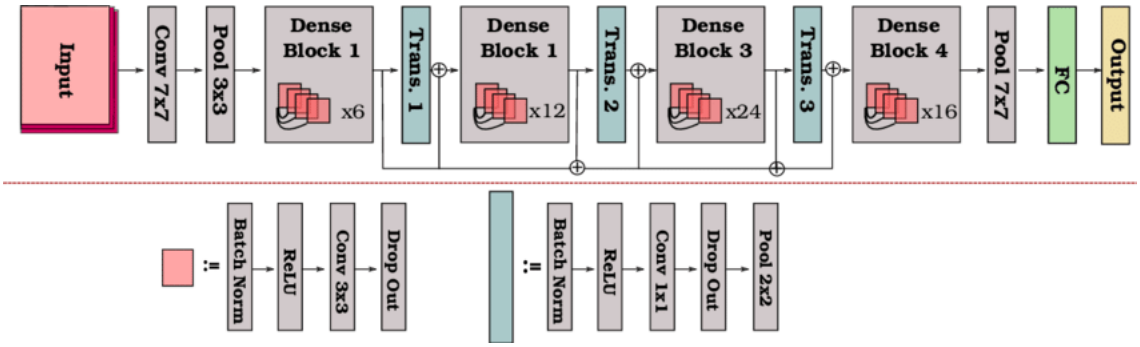

Figure 2.23: An illustration of the DenseNet121 architecture: Original image published in paper [41].

InceptionV3

The Inception family of networks, first introduced by Szegedy et al. [23] in 2015, marked a significant shift in CNN architecture design by focusing on computational efficiency while increasing representational power. Rather than simply stacking more layers or increasing filter sizes, Inception introduced a novel module based approach: multiple convolutional operations

of different kernel sizes are performed in parallel within the same layer, and their outputs are concatenated along the channel dimension. This design enables the network to capture features at multiple spatial scales simultaneously, improving its ability to learn both fine-grained details and more abstract patterns. InceptionV3, one of the most widely used variants, introduced several refinements to this original concept, further improving both accuracy and computational efficiency. One key innovation is the use of factorised convolutions, where larger kernels, such as 5×5, are replaced by multiple smaller ones (e.g. two 3×3 convolutions), reducing the number of parameters while maintaining receptive field size. Additional improvements include label smoothing and auxiliary classifiers, both of which help regularise the network and improve gradient flow during training. These enhancements enable deeper and more complex networks to be trained without incurring prohibitive computational costs. The InceptionV3 architecture is composed of a series of these inception modules, interleaved with pooling layers, to progressively build higher level feature representations. Its strength lies in balancing expressiveness and efficiency, by processing multiple receptive field sizes in parallel, the model can learn a diverse range of features without significantly increasing computational demand. However, this modular design is more complex and less uniform than simpler architectures such as VGG, which can make implementation and optimisation more challenging. Despite this, InceptionV3 remains a widely used and influential model in both research and industry. Moreover, its architectural principles paved the way for hybrid models such as Inception-ResNet, which combine multiscale feature extraction with residual learning. Figure 2.24 depicts the full architecture of InceptionV3.

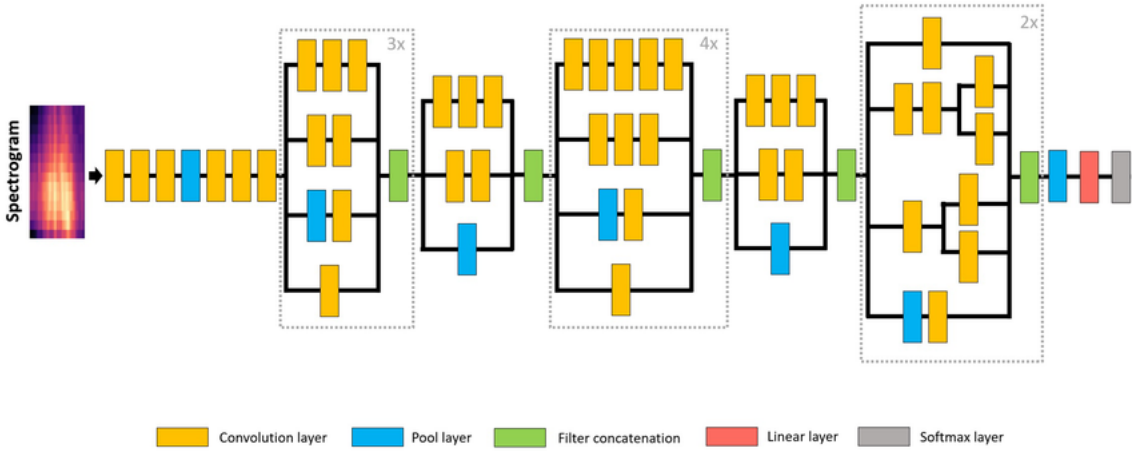

Figure 2.24: InceptionV3 Architecture: Original image published in paper [42].

InceptionResNetV2

InceptionResNetV2, introduced by Szegedy et al. [24] in 2017, builds upon the design principles of the Inception family while integrating key ideas from residual networks to further enhance training dynamics and performance. Whereas standard Inception modules concatenate the outputs of parallel convolutional branches, InceptionResNetV2 introduces residual connections to identify shortcut paths that bypass each block and allow feature information

and gradients to flow directly through the network. This modification helps alleviate training difficulties that can arise in very deep models, reduces the risk of vanishing gradients, and enables faster convergence during optimisation. The architecture retains the powerful scalable feature extraction capability of Inception modules but combines it with the stability and efficiency of residual learning. It is typically structured into three main stages: the stem, which performs initial feature extraction; the Inception-ResNet-A/B/C blocks, which form the core of the network and capture increasingly abstract feature representations; and the reduction modules, which downsample feature maps to control computational complexity as depth increases. An additional technique called residual scaling is also introduced which means the output of the residual branch is scaled by a small constant before being added to the main path. This prevents large residuals from destabilising training and contributes to more stable optimisation in deeper networks. By combining the complementary strengths of Inception and ResNet, InceptionResNetV2 achieves faster convergence and often superior performance compared to non-residual Inception networks. Its architecture is highly versatile and widely adopted as a backbone for a range of tasks beyond image classification, including object detection, segmentation, and feature extraction for transfer learning. Importantly, the success of this hybrid approach also laid the foundation for subsequent architectures such as Xception, which further simplify and extend the concept of modular feature extraction. Figure 2.25 presents an overview of the hybrid design that is InceptionResnetV2.

Xception

The Xception architecture, proposed by François Chollet in 2017 [19], represents a natural progression from the Inception family by fully embracing the concept of depthwise separable convolutions. While Inception modules aim to capture multiscale spatial information by applying filters of different sizes in parallel, Xception simplifies and generalises this approach by decomposing standard convolutions into two distinct operations: a depthwise convolution, which applies a single spatial filter to each input channel independently, followed by a pointwise convolution (1×1) that combines information across channels. This separation allows the model to learn spatial and cross-channel correlations independently, resulting in improved representational efficiency and significantly reducing the number of parameters. The architecture of Xception is organised into three main components: the entry flow, middle flow, and exit flow. The entry flow is responsible for capturing low-level spatial features using initial convolution and pooling layers. The middle flow, composed of multiple depthwise separable convolutional blocks with residual connections, learns increasingly abstract and complex feature representations. Finally, the exit flow refines these features and prepares them for classification using additional convolutional layers and global average pooling. The integration of residual connections throughout the network enhances gradient propagation and contributes to more stable and efficient training. One of the most notable advantages of Xception is its strong balance between accuracy and computational cost. By decoupling spatial and cross-channel feature learning, the architecture achieves state-of-the-art performance with fewer parameters than similarly deep conventional CNNs. Empirical results demonstrate that Xception consistently outperforms InceptionV3 on large-scale benchmarks such as ImageNet [12], despite having a comparable parameter count. However, the model

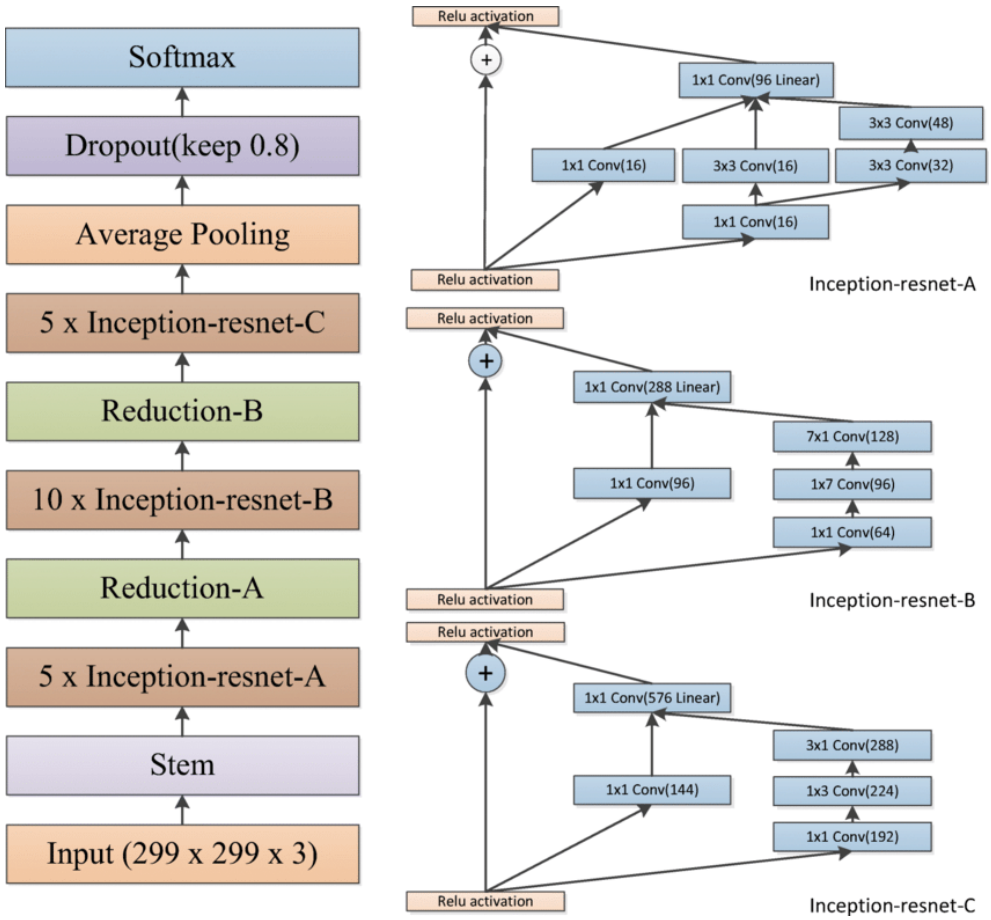

Figure 2.25: InceptionResNetV2 Architecture: Original image published in paper [43].

can be sensitive to hyperparameters and requires careful tuning to achieve optimal performance. Overall, Xception showcases the effectiveness of depthwise separable convolutions as a fundamental building block for deep learning architectures and has influenced the design of many subsequent models. Xception's architecture is outlined in Figure 2.26.

EfficientNet

EfficientNet, introduced by Mingxing Tan and Quoc V. Le in 2019 [20], represents a new generation of convolutional neural network architectures designed to optimise both accuracy and computational efficiency. Earlier approaches to improving model performance often involved scaling up one aspect of a network such as its depth, width, or input resolution while keeping the others fixed. However, this manual scaling approach is suboptimal and can lead to diminishing returns. EfficientNet addresses this limitation with a compound scaling method, which uniformly scales depth, width, and resolution according to a set of carefully determined coefficients. This balanced scaling strategy ensures that all three dimensions grow together in a way that maximises model performance while keeping computational cost under control. EfficientNetB0 serves as the baseline model in the EfficientNet family and

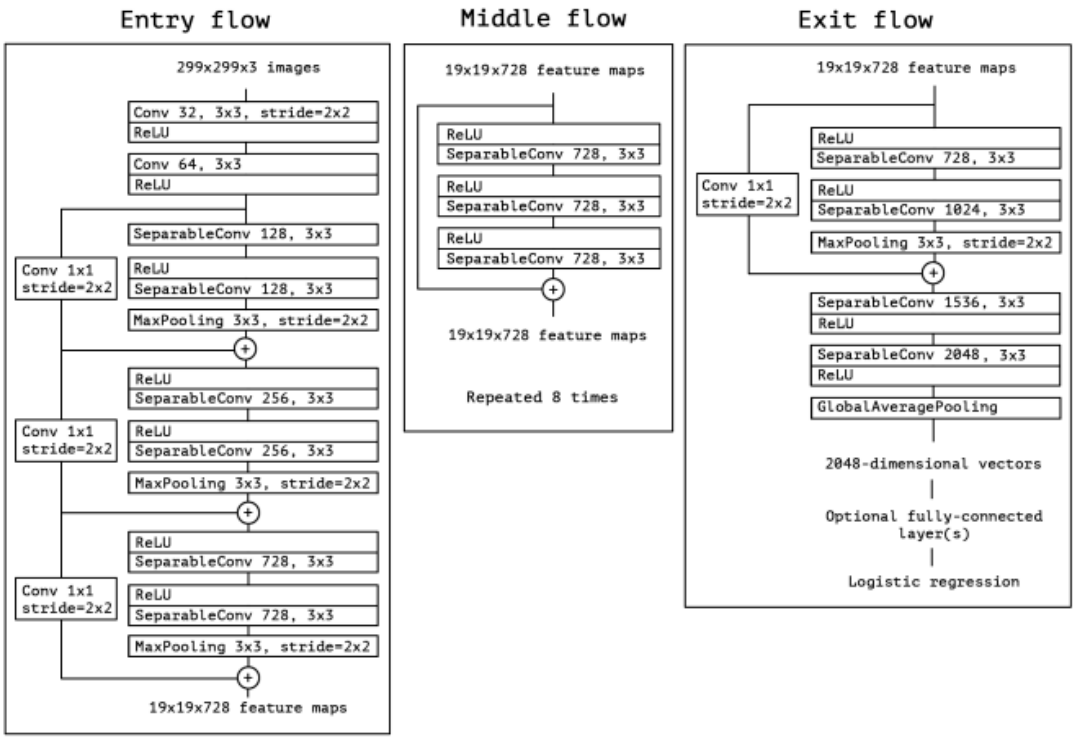

Figure 2.26: Xception Architecture: Original image published in paper [19].

forms the foundation for the larger variants (B1-B7). It is built using a series of Mobile Inverted Bottleneck Convolution (MBConv) blocks, originally introduced in MobileNetV2 [44], which combine depthwise separable convolutions with squeeze and excitation layers to adaptively recalibrate channel-wise feature responses. These design choices, along with the use of depthwise convolutions, greatly reduce the number of parameters and floating point operations (FLOPs) required, while preserving strong representational capacity. Despite being the smallest and most lightweight model in the EfficientNet series, EfficientNetB0 achieves impressive performance on large-scale image classification tasks, often outperforming much deeper and wider architectures. A key advantage of EfficientNetB0 is its exceptional parameter efficiency. It achieves high accuracy with orders of magnitude fewer parameters and computational requirements than models such as ResNet50 or InceptionV3. This makes it particularly well suited for deployment in resource constrained environments, including mobile devices and embedded systems. However, the reliance on compound scaling also introduces certain challenges: EfficientNet models can be more sensitive to hyperparameter choices, and their design is less intuitive than that of more traditional architectures. Nonetheless, EfficientNetB0 and its scaled variants represent an important milestone in CNN design, demonstrating that carefully balanced scaling strategies and lightweight building blocks can achieve state-of-the-art performance with minimal computational overhead. The structural composition of EfficientNetB0 is visualised in Figure 2.27.

[2]Image source: Ahmed & Sabab (2020), Classification and Understanding of Cloud Structures via Satellite Images with EfficientUNet, ESSOAr preprint. DOI: 10.1002/essoar.10507423.1

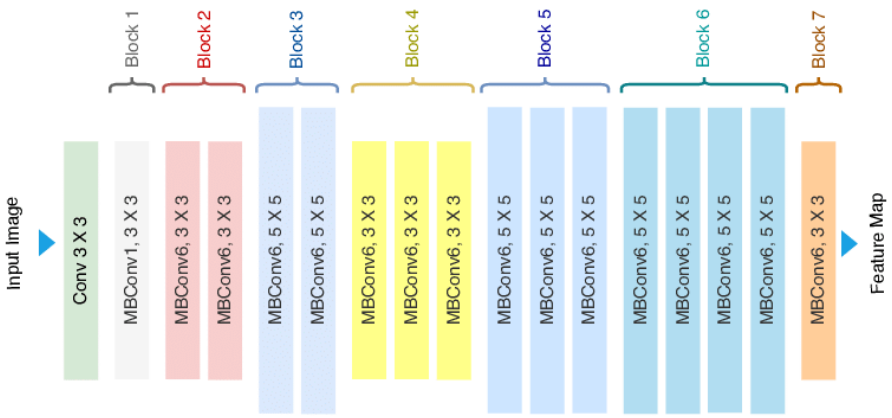

Figure 2.27: EfficientNetB0 Architecture.[2]

2.9 Model Evaluation Metrics

Assessing the performance of a classification model extends beyond simply observing its predictions, it requires a rigorous and quantitative evaluation of how effectively those predictions align with the true class labels. Evaluation metrics provide the mathematical foundation for this analysis, enabling us to measure the model's predictive capability, compare alternative approaches, and determine whether its outputs meet the requirements of the problem domain. In classification tasks, such metrics play a critical role in capturing different aspects of performance. Certain metrics, such as accuracy, furnish a broad assessment of predictive efficacy, whilst others, including precision, recall, and the F1-score, yield more nuanced understanding of a model's performance in addressing specific issues like class imbalance or asymmetric misclassification costs. This section focuses exclusively on metrics relevant to classification problems. It presents their mathematical formulations, discusses their interpretations, and highlights the contexts in which each is most informative. Establishing this foundation is essential for understanding not only how models are compared and optimised but also how their outputs should be interpreted within the broader scope of the application.

2.9.1 Common Classification Metrics

Before defining specific evaluation metrics, it is essential to introduce the confusion matrix [30] which a fundamental tool for analysing the performance of classification models. The confusion matrix provides a complete view of how a classifier's predictions compare to the true class labels by summarising them into four key components:

- True Positives (TP): Instances correctly predicted as belonging to the positive class.
- True Negatives (TN): Instances correctly predicted as belonging to the negative class.
- False Positives (FP): Instances incorrectly predicted as positive (type I error).
- False Negatives (FN): Instances incorrectly predicted as negative (type II error).

From these components, a variety of performance metrics can be derived. Each captures a different perspective on model behaviour and is informative in specific contexts. Figure 2.28 graphically depicts a confusion matrix.

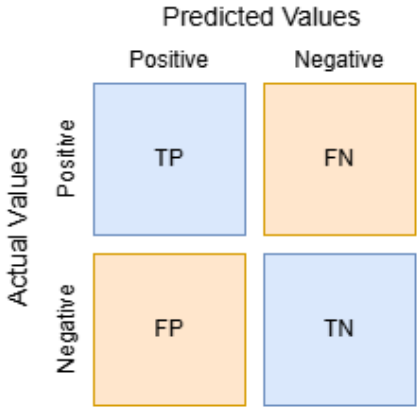

Figure 2.28: A Confusion Matrix.

Accuracy

Accuracy is one of the most commonly used metrics for evaluating classification models due to its simplicity and intuitive interpretation. It measures the proportion of correctly classified instances relative to the total number of predictions made by the model. Mathematically, accuracy is defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.33}$$

where TP and TN denote the number of true positives and true negatives, respectively, and FP and FN represent false positives and false negatives. A higher accuracy value generally indicates better overall performance, as it reflects the model's ability to correctly identify both positive and negative instances. However, despite its popularity, accuracy can be misleading in situations where the dataset is imbalanced. In such cases, a model that simply predicts the majority class may achieve high accuracy while failing to capture minority class patterns. Therefore, while accuracy serves as a useful baseline indicator, it is often complemented by additional metrics such as precision, recall, and the F1-score to provide a more comprehensive understanding of model performance.

Precision

Precision is a key way to judge how reliable a classifier's positive predictions are. It tells you what percentage of all the times a positive class was predicted were really correct. Precision is formally defined as

$$Precision = \frac{TP}{TP + FP} \tag{2.34}$$

where TP represents the number of true positives and FP the number of false positives. A high precision score indicates that the model makes relatively few false positive errors since this means that FP tends towards 0. In other words, when it predicts a positive class, it is usually correct. This property makes precision particularly important in applications where the cost of false positives is high, such as disease diagnosis, where incorrectly labeling a negative instance as positive could have significant consequences. However, precision alone does not provide a complete picture of model performance, as it does not account for false negatives. A model could achieve high precision by being overly conservative in its positive predictions, potentially missing many actual positives. For this reason, precision is often interpreted alongside recall to provide a more balanced understanding of a classifier's effectiveness.

Recall

Recall, also known as sensitivity or the true positive rate, measures a classifier's ability to correctly identify all instances of the positive class. It is defined as the proportion of actual positive examples that are correctly predicted by the model. Mathematically, recall is expressed as

$$Recall = \frac{TP}{TP + FN} \tag{2.35}$$

where TP denotes the number of true positives and FN the number of false negatives. A high recall score indicates that the model successfully captures the majority of positive instances, making it particularly valuable in scenarios where missing a positive case carries significant consequences. For example, in medical diagnostics, failing to identify a true positive (i.e. a false negative) could result in severe outcomes. However, focusing solely on recall can lead to an increase in false positives, as a model may label more instances as positive to avoid missing any actual positives. This trade-off illustrates why recall is often analysed in conjunction with precision: while precision evaluates the quality of positive predictions, recall assesses their completeness. Together, they provide a more comprehensive view of a model's classification performance.

F1-score

The F1-score is a prevalent metric for assessing classifier performance by integrating precision and recall into a single value. The F1-score is the harmonic mean of precision and recall, which means that it only gets a high value when both of these metrics are high. Formally, it is calculated as

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{2.36}$$

The use of the harmonic mean, rather than the arithmetic mean, penalises extreme imbalances between precision and recall, meaning that a model with very high precision but low recall (or vice versa) will not achieve a strong F1-score. This property is especially beneficial when both false positives and false negatives have serious effects, and a balance between finding as many positive cases as possible and making sure those forecasts are true is needed. The

F1-score is particularly advantageous when dealing with imbalanced datasets, as accuracy alone may not effectively indicate performance. The F1-score is a comprehensive metric for evaluating a model's performance in classification tasks, as it illustrates the balance between precision and recall.

In summary, evaluation metrics such as accuracy, precision, recall, and the F1-score form the foundation for assessing the performance of classification models. Each provides a distinct perspective on predictive behaviour: accuracy offers an overall measure of correctness, precision reflects the reliability of positive predictions, recall evaluates the model's ability to capture all positive instances, and the F1-score balances these two complementary objectives. Understanding the trade-offs among these metrics is essential when interpreting model performance, particularly in applications where class imbalance or asymmetric misclassification costs play a critical role. By selecting and interpreting metrics in a manner aligned with the problem context, practitioners can make more informed decisions regarding model selection, optimisation, and deployment. This framework for evaluation provides an essential basis for the deeper analysis of model performance that follows, particularly in the context of convolutional neural networks, where architectural design can significantly influence classification outcomes.

2.9.2 Linear Probe Accuracy

Linear probe accuracy is a widely used diagnostic tool for assessing the quality of internal feature representations learned by deep neural networks. Instead of training a full classification head, a linear probe freezes the parameters of the network and trains a lightweight linear classifier, such as logistic regression or a linear SVM [33], on the activations extracted from each layer [45]. This methodology isolates the intrinsic representational power of a layer from the optimisation dynamics of the full model, offering a clearer view of how easily separable the features are at different depths. Linear probes have become standard in representation learning research, where they are commonly used to analyse both supervised and self-supervised CNNs [46, 47]. High probe accuracy at a given layer indicates that the features are linearly separable, suggesting that the network has already organised the data into a discriminative structure. In contrast, low probe accuracy implies that the representations require additional non-linearly driven refinement to support classification. Beyond measuring representational quality, linear probes have also been used to assess robustness in real-world conditions. Liang et al. [48] showed that simple episodic linear probes can achieve strong visual recognition performance under distribution shift, reinforcing their value as a stability diagnostic. In contexts involving noisy or corrupted inputs, probe accuracy provides a complementary view to physical measures such as the SNR. While SNR reflects the clarity and intensity of activation patterns, linear probe accuracy directly evaluates the discriminative utility of those patterns. In this thesis, linear probes are applied to the intermediate layers of several CNN architectures to characterise how feature separability evolves with depth under both natural background noise and injected Gaussian noise. This allows for a detailed comparison of how architectural components such as residual blocks, dense connectivity, and depthwise separable convolutions influence the formation, preservation, and degradation of

linearly decodable feature structure.

2.10 Conclusion

In this chapter, we examined the fundamental principles underpinning neural networks, beginning with their biological foundations and extending to the artificial models that emulate their behaviour. We explored the structure and functionality of ANNs, emphasising the role of activation functions in introducing nonlinearity and enabling networks to model complex patterns. Several activation functions were reviewed in detail such as Sigmoid, Tanh, ReLU, Leaky ReLU, and Softmax. Each of them offer distinct mathematical properties and advantages depending on the problem context. Building on this foundation, we discussed supervised learning, outlining the differences between classification and regression tasks and the optimisation processes that allow neural networks to learn from data. The feed-forward and backpropagation mechanisms were analysed to illustrate how weights and biases are iteratively updated to minimise prediction error and improve model performance over time.

The discussion then advanced to CNNs, a specialised class of ANNs. We described the convolutional and pooling layers that form the backbone of these networks and explored how they enable hierarchical feature extraction while reducing computational complexity. This was followed by a review of several key CNN architectures such as AlexNet, VGG, ResNet, DenseNet, Inception, Xception, and EfficientNet. These architectures represent major milestones in the development of deep learning with each architecture introducing innovative mechanisms, such as residual and dense connections or compound scaling strategies, that improved depth, efficiency, and robustness. Collectively, these models encapsulate the evolution of modern CNN design and establish the theoretical framework for understanding how structural choices influence learning dynamics and predictive capability.

The findings articulated in this chapter establish the foundation for examining one of the most crucial factors affecting CNN performance: data quality. The next chapter talks about the SNR, which is a way to quantify the amount of relevant information in visual data compared to the amount of noise. To evaluate the robustness and generalisability of a model, one must understand the interplay between signal and noise. This shift from architectural design to noise analysis signifies the subsequent phase in linking theory to experiments, wherein the susceptibility of deep learning models to diverse noise environments will be systematically evaluated.

Chapter 3

Signal To Noise Ratio (SNR)

3.1 Introduction

This chapter examines signal to noise theories in the field of signal processing. A key aspect of this analysis will involve the SNR, which offers valuable insight into the relationship between the signal and noise present in input images. Noise, originally defined in signal processing, refers to unwanted variations introduced during the capture, storage, transmission, or processing of a signal. In machine learning, noise is typically associated with anomalies or corruptions within a dataset [49]. The SNR quantifies the balance between the meaningful information in the data (the signal) and the unwanted elements (the noise). This ratio is crucial for assessing the quality and usefulness of the data. The following sections will provide an overview of the SNR and highlight its relevance to the objectives of this study.

3.2 Types of Noise

In this section, we explore various types of noise commonly encountered in the machine learning field and examine their visual appearance in image data. There are three main types: class, attribute, and injected noise.

3.2.1 Class Noise

Class noise occurs due to labeling errors within a dataset [50]. These errors may arise from human mistakes during data collection, subjectivity, or insufficient information for labeling an example. Within class noise, one can distinguish between two types: contradictory class noise, where an example appears multiple times in the dataset with different labels, and misclassification class noise, where examples are mislabeled. This concept is visualised in Figure 3.1. In the figure, we can see that the first row (or instance) and the third row of data have identical features/attributes. This should, in turn, result in the label being identical. However, we can see that in the first row the label is 0, while in the third row the label is 1.

This would confuse the model and is therefore classified as class noise.

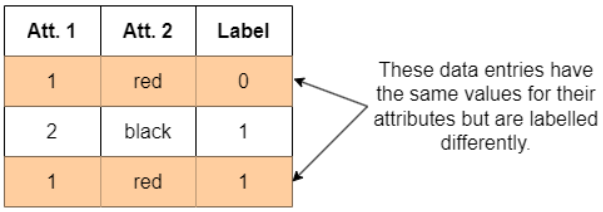

Figure 3.1: Example of class noise, where identical feature vectors are associated with conflicting class labels, resulting in contradictory supervision.

3.2.2 Attribute Noise

Attribute noise refers to errors or inconsistencies within the attributes or features of a dataset [50]. Common examples include missing attribute values, incorrect values, or attributes that are irrelevant to the target label and thus add unnecessary complexity to the learning process. An illustration of attribute noise is provided in Figure 3.2. In this figure, we can observe that attribute 2 has a missing value in the second row of data, which contributes to attribute noise and could hinder the model's ability to learn effectively.

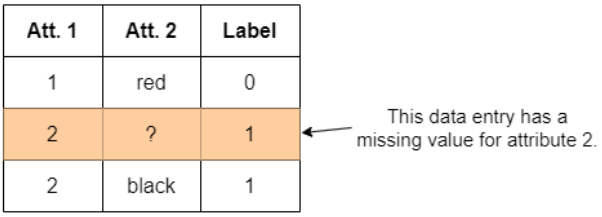

Figure 3.2: Example of attribute noise caused by a missing attribute value in the second data instance, introducing inconsistency in the feature space.

3.2.3 Injected Noise

Noise can be added to a dataset that is otherwise free from noise. This injected noise can affect either the attributes or the classes. Researchers might do this to control the amount or type of noise presented to the dataset [50].

Attribute Noise Injection

A common form of injected attribute noise is Gaussian noise, also known as white noise [50]. Gaussian noise introduces small, random variations into the dataset, which can give images a static-like appearance. The probability density function of a Gaussian random variable x is expressed as:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{3.1}$$

where μ is the mean and σ is the standard deviation. These parameters can be adjusted to control the level of noise. Figures 3.3, 3.4, and 3.5 illustrate different levels of injected Gaussian attribute noise. As the standard deviation σ increases, the level of static-like noise also rises, making the image increasingly difficult to discern. This can be considered analogous to how a model reacts to noise. Excessive noise obscures important features, hindering the model's ability to correctly identify and learn from the data.

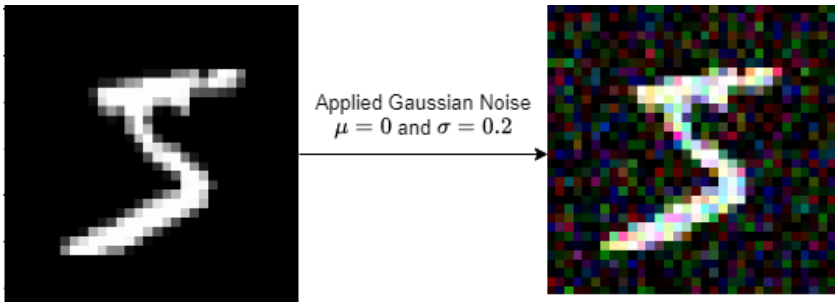

Figure 3.3: Injected Gaussian attribute noise applied to an image with $\mu = 0, \sigma = 0.2$, producing mild static-like perturbations.

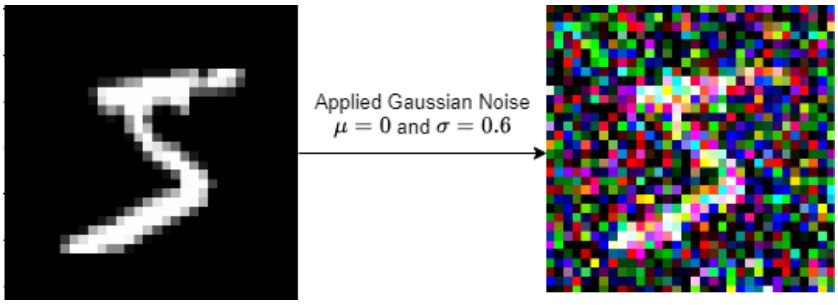

Figure 3.4: Injected Gaussian attribute noise applied to an image with $\mu = 0, \sigma = 0.6$, producing moderate static-like perturbations.

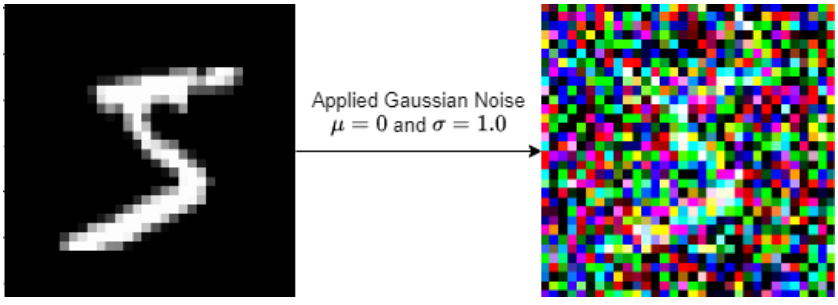

Figure 3.5: Injected Gaussian attribute noise applied to an image with $\mu = 0, \sigma = 1.0$, producing severe static-like perturbations.

Class Noise Injection

Class noise injection involves randomly altering the class labels of a portion of the data to different possible class labels. This method introduces intentional mislabeling, simulating real-world noise that could arise from human error or ambiguity in data labeling. The degree of class noise can be adjusted by modifying the percentage of labels that are altered. However, we are not interested in this type of noise for this work.

3.3 SNR Formulation for This Work

While CNNs excel at extracting meaningful features from structured data, they are also known to be sensitive to noise and perturbations. Even small, imperceptible changes to input images can lead to significant drops in classification accuracy. In real-world settings such as medical imaging, noise often arises naturally from acquisition artefacts, patient movement, or device variability, making robustness a practical necessity. Efforts to improve robustness include data augmentation with noisy inputs, adversarial training, and architectural changes such as noise resilient normalisation layers or denoising front ends. However, most of these approaches focus on improving end to end performance rather than understanding the internal dynamics of noise propagation. This thesis takes a complementary approach by analysing how signal and noise evolve across layers and how these dynamics relate to classification performance.

Several alternative measures have been proposed to characterise information flow in deep neural networks, including information-theoretic quantities such as mutual information between layer activations and input or output variables. Mutual information has been used to assess the quality of learned representations by quantifying the dependence between inputs and internal features, as demonstrated in representation learning frameworks such as Deep InfoMax [51]. However, practical estimation of mutual information in deep networks is challenging due to the high dimensionality of activation spaces and the need for approximation or variational estimation strategies, which can reduce interpretability in applied settings. In contrast to information-theoretic metrics, SNR provides a direct and physically interpretable measure of representational quality that does not require estimation of complex probability distributions. In the context of this work, SNR is particularly suitable because it enables layer-wise analysis of signal preservation and noise amplification using quantities that are readily computable from network activations. Moreover, SNR aligns naturally with medical imaging applications, where foreground structures of interest and background tissue or acquisition artefacts can be explicitly separated, allowing signal and noise components to be defined in a meaningful and task relevant manner.

SNR is defined as the ratio of signal power to noise power. Drawing from the work of Mitson et al. [52] in fisheries acoustics, the received sound, composed of both signal and noise, is captured by a transducer and processed by an echo sounder. This process yields the filtered signal, denoted as S, and the filtered noise, denoted as N. The noise component N can be further broken down into background noise, $N_{\text{background}}$, and thermal noise, N_{thermal},

where the latter typically follows a Gaussian distribution. The SNR can be estimated using equation 3.2:

$$\begin{aligned} \text{SNR} &\approx \frac{S}{N} \\ &\approx \frac{S}{N_{\text{background}} + N_{\text{thermal}}}, \end{aligned} \tag{3.2}$$

where the SNR is expressed as a power ratio, rather than in decibels. To illustrate the effect of noise on the SNR, consider a scenario where the signal remains constant at 1. As the noise level increases, the SNR will decrease, as shown in Figure 3.6. Now, consider an image

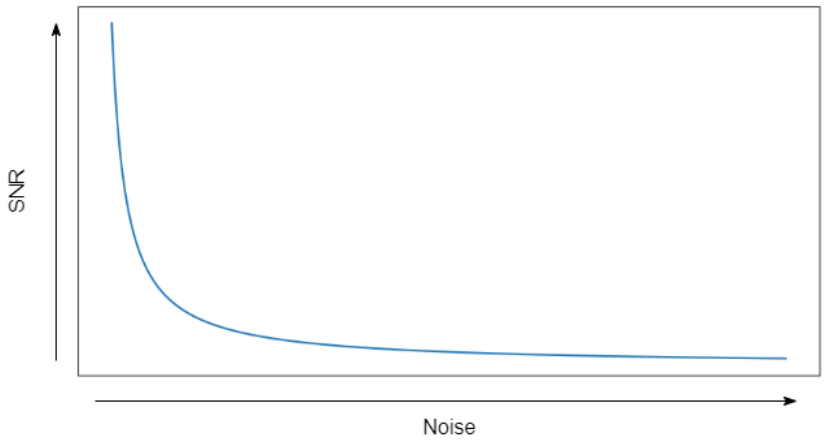

Figure 3.6: Illustration of the inverse relationship between noise magnitude and SNR for a fixed signal amplitude.

with dimensions $[N \times M \times Q]$. The SNR in decibels for a single image, and therefore for a single neuron, is computed as follows:

$$\begin{aligned} \text{SNR} &= 10 \log_{10} \left(\frac{\text{mean}(S)}{\text{std}(N)} \right) \\ &= 10 \log_{10} \left(\frac{\text{mean}(S)}{\text{std}(N_{\text{background}} + N_{\text{thermal}})} \right). \end{aligned} \tag{3.3}$$

The SNR for a layer is defined as the average SNR across all input images. This formulation is illustrated conceptually in Figure 3.7, which shows how signal and background activations are separated within a convolutional feature map.

3.4 Conclusion

In this chapter, we examined the importance of the SNR in evaluating the influence of noise on input data, especially within the framework of machine learning. We looked at numerous kinds of noise, such as class, attribute, and injected noise, and spoke about how they affect

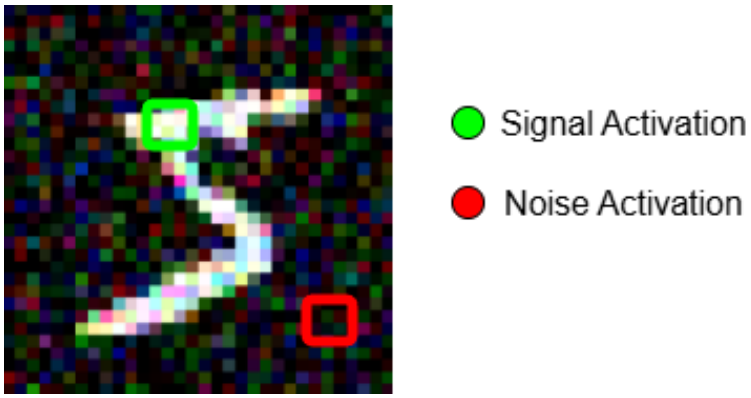

Figure 3.7: Illustration of signal (green) and noise (red) activations used for SNR computation.

data quality and model performance. The SNR formulation, based on the research of Mitson et al. [52], established a framework for assessing the equilibrium between signal and noise in picture data. We showed how noise alters the SNR under different conditions and discussed how higher noise levels affect model learning. By adding Gaussian noise and simulating how it affects input data, we learned a lot about how models react to different levels of noise at different levels of deep learning. To improve data preprocessing methods and, in the end, the accuracy of deep learning models, it is important to know how SNR affects model performance. The ideas presented here lay the groundwork for more research into how noise affects deep learning and why it is important to keep a high SNR for the best model training and generalisation.

Chapter 4

Analysis of SNR Propagation in AlexNet Using the MNIST Dataset

4.1 Introduction

The field of image classification has achieved remarkable progress with the rise of deep learning, particularly through the use of CNNs. One of the most widely used benchmark datasets in this domain is the Modified National Institute of Standards and Technology (MNIST) dataset, which has been instrumental in training and evaluating CNNs. While MNIST is inherently clean and largely free from noise, real-world image data often contains varying degrees of corruption, which can significantly degrade model performance. Consequently, understanding how a model inspired by the AlexNet architecture responds to signal degradation in the presence of noise is essential for developing robust deep learning systems. This chapter investigates the effects of SNR propagation in a simple AlexNet CNN architecture. Gaussian noise was systematically injected into the MNIST dataset during training to simulate different noise conditions. The analysis focuses on how alternative pooling configurations influence SNR behaviour across convolutional layers and how these dynamics evolve when models are trained over multiple epochs under varying levels of noise.

4.2 Dataset

The dataset used in this initial experiment is the well-known MNIST dataset [53], which consists of images of handwritten digits. Created in 1994, the MNIST dataset was developed by combining two databases from the National Institute of Standards and Technology (NIST): Special Database 1, containing samples of handwritten digits from employees of the U.S. Census Bureau, and Special Database 3, which includes digitized figures written by high school students [54]. The complete dataset comprises $60,000$ training examples and $10,000$ test examples. Each image represents a digit in a 28×28 pixel grayscale format, making this dataset an ideal benchmark for image classification tasks. The MNIST dataset is inherently

free from noise whether thermal, background, or otherwise. This characteristic makes it well suited for controlled experiments, as it allows for the systematic introduction of artificial noise while maintaining a well defined baseline. In this study, Gaussian noise is added to simulate real-world distortions that can degrade model performance. By introducing varying noise levels, we can assess the model's robustness and examine how pooling layers mitigate noise effects. This, in turn, provides insights into the relationship between noise levels and SNR in image classification tasks. Figure 4.1 shows five randomly selected examples from the raw dataset.

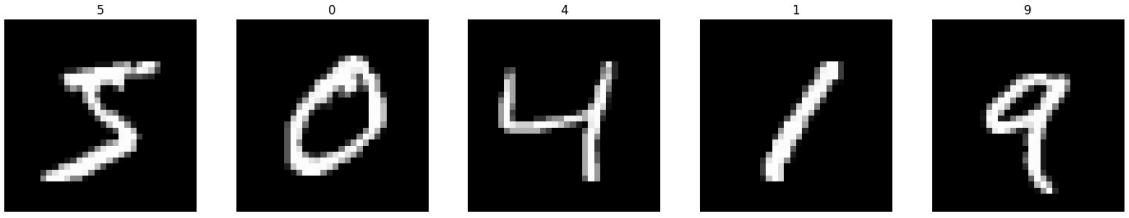

Figure 4.1: Random sample of five MNIST handwritten digit images, illustrating the baseline visual quality prior to noise injection.

4.3 Methodology

The objective of this initial investigation is to understand how the SNR is influenced by both the number of pooling layers and the level of noise injected into the input data. In addition, we aim to examine how the SNR propagates and evolves across different layers within the AlexNet CNN. The experiment is structured in two parts. The first part investigates how the SNR, calculated after each convolutional layer, behaves across a single training epoch for a range of model architectures. The second part focuses on a subset of these architectures and analyses how SNR dynamics evolve over multiple epochs. Together, these experiments address the central question posed in the introduction: how different pooling configurations affect SNR propagation in CNNs, and how this behaviour changes under varying noise conditions. By systematically varying both noise levels and pooling strategies, we aim to gain deeper insight into the role of pooling in mitigating noise and preserving signal integrity throughout the convolutional hierarchy. Figure 4.2 illustrates how increasing the noise standard deviation progressively distorts the input image.

4.3.1 SNR Calculation Across Layers for 1 Epoch

To establish a baseline, the first model was designed with ten convolutional layers [55], followed by fully connected layers. This depth was chosen to ensure sufficient hierarchical feature extraction while maintaining computational feasibility. Each convolutional layer applies a 3×3 kernel with 128 filters, as this kernel size effectively captures local spatial patterns while keeping parameter complexity manageable for this type of controlled experiment. The number of filters was set to 128 to balance expressive power and computational efficiency. A

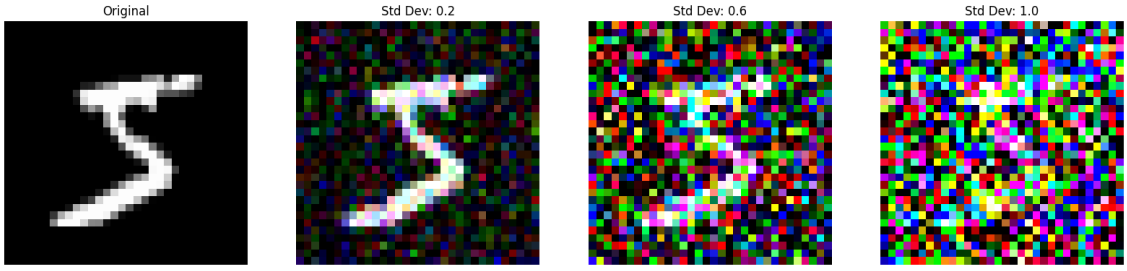

Figure 4.2: Example MNIST digit with injected Gaussian noise at increasing standard deviation (Std Dev), illustrating the qualitative effect of noise severity on input visibility.

padding of "same" was used to preserve spatial dimensions across layers, facilitating deeper networks without excessive downsampling. The default stride of 1 was applied to maintain fine-grained feature extraction. Figure 4.3 outlines the architecture of this base model.

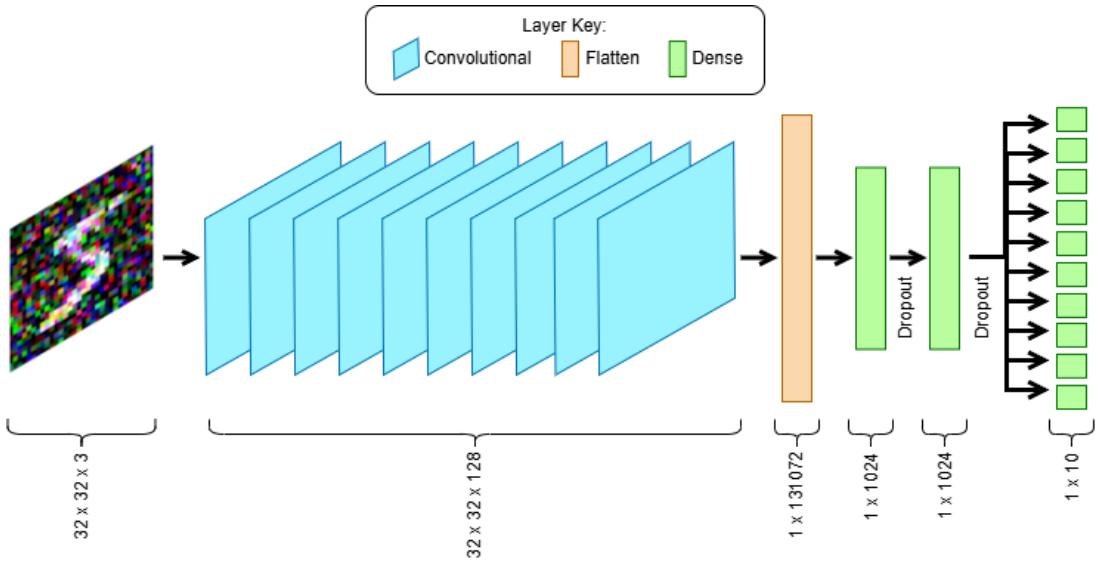

Figure 4.3: Architecture of the Base Model Without Pooling Layers.

The second model incorporates a single pooling layer to examine its impact on feature extraction. To systematically evaluate this effect, a series of models was created by placing the pooling layer at various positions within the architecture. For instance, Figure 4.4 illustrates the model where the pooling layer is introduced after the first convolutional layer, using a pooling size (or kernel size) of 2×2 with a stride of 1. The 2×2 kernel was chosen as it is a common configuration that aggregates local features while preserving spatial resolution. To assess the influence of pooling placement, additional models were developed by positioning the pooling layer after the second convolutional layer, the third convolutional layer, and so on. Furthermore, the pooling size was varied to 4×4 and 6×6 to explore how different levels of spatial reduction affect performance. This systematic variation resulted in 10 models with a pooling size of 2×2, 10 models with a pooling size of 4×4, and 10 models with a pooling size of 6×6, leading to a total of 30 additional models. Importantly, the number of times the SNR is computed was kept consistent across models to facilitate a fair comparison of

their effects on noise levels in the training data.

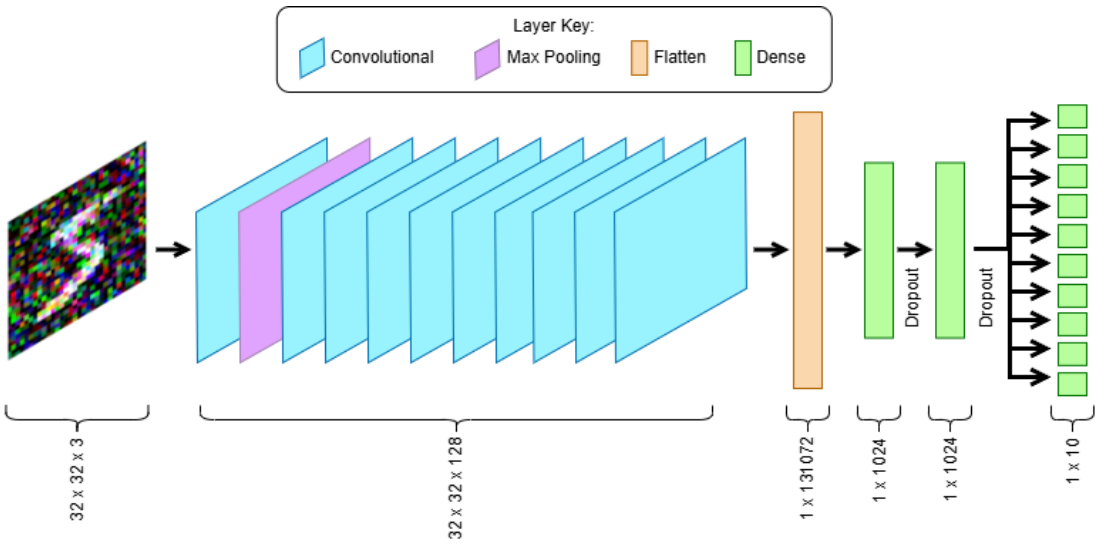

Figure 4.4: Architecture of the Model with One MaxPooling Layer.

The third and final model incorporates two pooling layers to examine their impact on feature aggregation and noise suppression. Similar to the previous models, a variety of configurations was explored by positioning the pooling layers at different locations within the architecture. For illustration, Figure 4.5 presents a model where the pooling layers are placed after the second and fifth convolutional layers. Additional models in this series were systematically developed by repositioning the pooling layers. For instance: after the first and second convolutional layers, the third and sixth convolutional layers, and so forth. Since "same" padding is used, the spatial dimensions are preserved, ensuring that pooling primarily functions as a local feature extractor rather than a downsampling mechanism. The pooling sizes were varied to 4×4 and 6×6 to investigate how different levels of local feature aggregation influence the network's ability to retain essential information while suppressing noise. This structured approach resulted in 45 models with a pooling size of 2×2, 45 models with a pooling size of 4×4, and 45 models with a pooling size of 6×6, culminating in a total of 135 additional models. As with the previous experiments, the number of times the SNR is computed was kept consistent to enable a meaningful comparison of noise effects across models.

For each of the models described above, comprising one base model, 30 models with a single pooling layer, and 135 models with two pooling layers, Gaussian attribute noise was introduced into each sample of the $60,000$ training images. This noise was characterised by $\mu = 0$ and $\sigma = 0.2$, in accordance with the formulas detailed in Section 3.2.3. Figure 4.6 shows an example of Gaussian noise of $\mu = 0$ and $\sigma = 0.2$. The SNR was computed following each convolutional layer, or, in instances where a convolutional layer was succeeded by a pooling layer, the calculation was performed after the pooling layer. The relevant formulas used to calculate the SNR are outlined in Section 3.3. Table 4.1 illustrates the layers in the model with two pooling layers where the SNR was calculated, specifically after convolutional layer 2 and after convolutional layer 5. The SNR values for each model were recorded and then plotted.

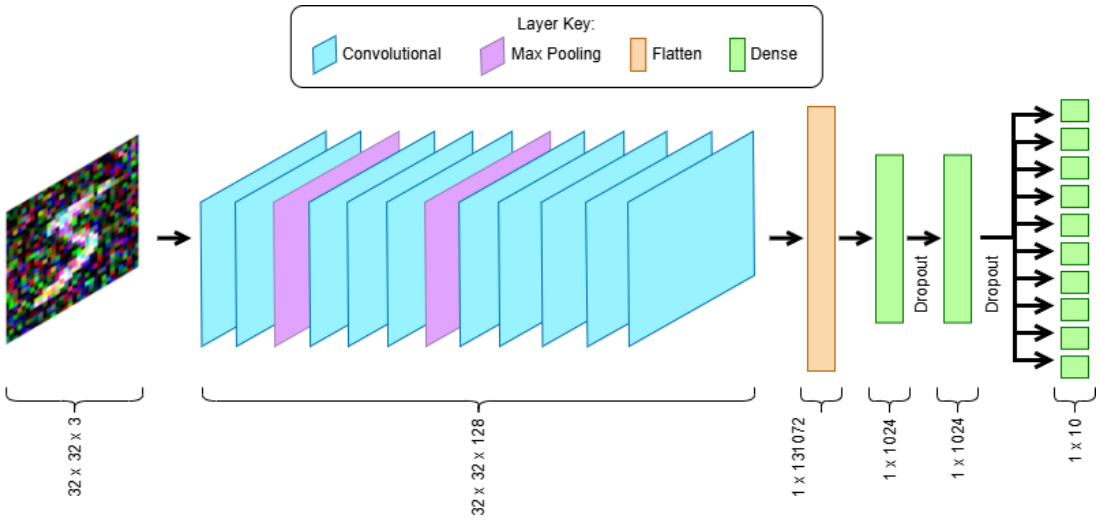

Figure 4.5: Architecture of the Model with MaxPooling Layers After the 2nd and 5th Convolutional Layers.

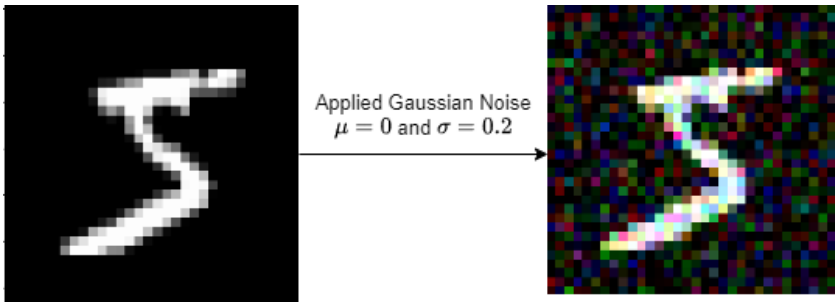

Figure 4.6: Example MNIST digit after Gaussian noise injection ($\mu = 0, \sigma = 0.2$), used as the low-noise condition for SNR propagation experiments.

4.3.2 SNR Calculation Across Layers for 10 Epochs

A selection of the models mentioned above was utilised for this part of the experiment to assess the relationship between SNR and model accuracy under different noise conditions. Each model was trained for 10 epochs, a duration chosen to capture initial learning trends while balancing computational efficiency. During training, the SNR was computed for every layer using the techniques outlined in Section 3.3, ensuring a layer-wise analysis of noise propagation. Additionally, the accuracy was recorded at each epoch to facilitate a comparative analysis of how accuracy changes in response to variations in SNR.

To evaluate model robustness, Gaussian noise was injected into the training images at three different levels:

- $\mu = 0, \sigma = 0.2$: A low-level of noise, simulating minor perturbations.
- $\mu = 0, \sigma = 0.6$: A moderate noise level, representing more noticeable distortions.
- $\mu = 0, \sigma = 1.0$: A high-level of noise, testing the model's capacity to learn in a really

Table 4.1: Layer Types and SNR Calculations.

Layer Type	**SNR Calculation**
Convolutional Layer 1	Calculate SNR
Convolutional Layer 2	-
MaxPooling Layer 1	Calculate SNR
Convolutional Layer 3	Calculate SNR
Convolutional Layer 4	Calculate SNR
Convolutional Layer 5	-
MaxPooling Layer 2	Calculate SNR
Convolutional Layer 6	Calculate SNR
Convolutional Layer 7	Calculate SNR
Convolutional Layer 8	Calculate SNR
Convolutional Layer 9	Calculate SNR
Convolutional Layer 10	Calculate SNR
Flatten Layer	-
Dense Layer	-
Dropout Layer	-
Dense Layer 1	-
Dropout Layer 1	-
Dense Layer 2	-

distorted environment.

These noise levels were picked to see how higher noise levels affect the SNR and accuracy of different model architectures. The chosen models for this experiment exemplify significant architectural variations:

- Base model with no pooling: This serves as a control to evaluate the efficacy of convolutional layers in managing noise independently.
- Model with one pooling layer (after the 10^{th} convolutional layer): Examines the effect of late-stage feature aggregation on noise suppression.
- Model with two pooling layers (after the 1^{st} and 3^{rd} convolutional layers): Evaluates whether early-stage feature aggregation provides benefits in mitigating noise effects.

By analysing these configurations, the experiment aims to uncover patterns in how pooling influences SNR and accuracy in noisy environments.

4.4 Results

4.4.1 SNR Calculation Across Layers for 1 Epoch

For the models in this Subsection, the x-axis labels in the corresponding SNR plots, specifically from Figure 4.7 onwards, were changed from "Conv" to "Layer". This adjustment reflects the fact that the SNR was calculated after each layer in the network, whether convolutional or pooling. When a pooling layer is inserted after a convolutional layer, for example after the first convolution, the SNR measurement for that stage represents the combined effect of both the convolution and the subsequent pooling operation. Because the position and number of pooling layers differ between model configurations, using the more general label "Layer" ensures that the plots remain accurate and comparable across models.

The results of the SNR calculation for our baseline model, without pooling layers, are presented in Table 4.2 and Figure 4.7. These show the SNR for the raw input images and after each subsequent convolutional layer. For the case of Gaussian noise with $\sigma = 0.2$, the SNR starts high and exhibits an overall decreasing trend. This indicates that as the input passes through the convolutional layers, noise increases relative to the signal. However, after convolutional layer 2, the SNR briefly improves before continuing to decline, suggesting that the model may reach its optimal performance with only two convolutional layers. Interestingly, when the input is instead corrupted with heavier Gaussian noise $\sigma = 1.0$, a different trend emerges. In this case, the raw image begins with a much lower SNR relatively speaking, but as it progresses through the convolutional layers, the SNR steadily increases. This suggests that the convolutional layers are effectively acting as filters to suppress noise while enhancing features thereby recovering the signal over time. Overall, the convolutional stack slightly degrades SNR for relatively clean inputs, but improves SNR for heavily corrupted inputs, indicating that convolutional layers can act as mild distortions in low-noise settings and as effective denoising filters when noise dominates.

Table 4.2: SNR values across different layers.

Layer	**Noise with $\sigma = 0.2$**	**Noise with $\sigma = 1.0$**
Raw Image	5.780421	-1.268568
After Layer 1	4.061133	-0.689908
After Layer 2	4.621880	-0.253581
After Layer 3	4.029476	-0.505440
After Layer 4	3.598565	-0.622276
After Layer 5	3.729382	-0.280470
After Layer 6	3.166172	-0.736925
After Layer 7	3.316945	-0.548160
After Layer 8	3.813275	0.088287
After Layer 9	3.754750	0.175707
After Layer 10	3.046274	-0.512048

The results of the SNR calculation for each model with a single pooling layer and Gaussian

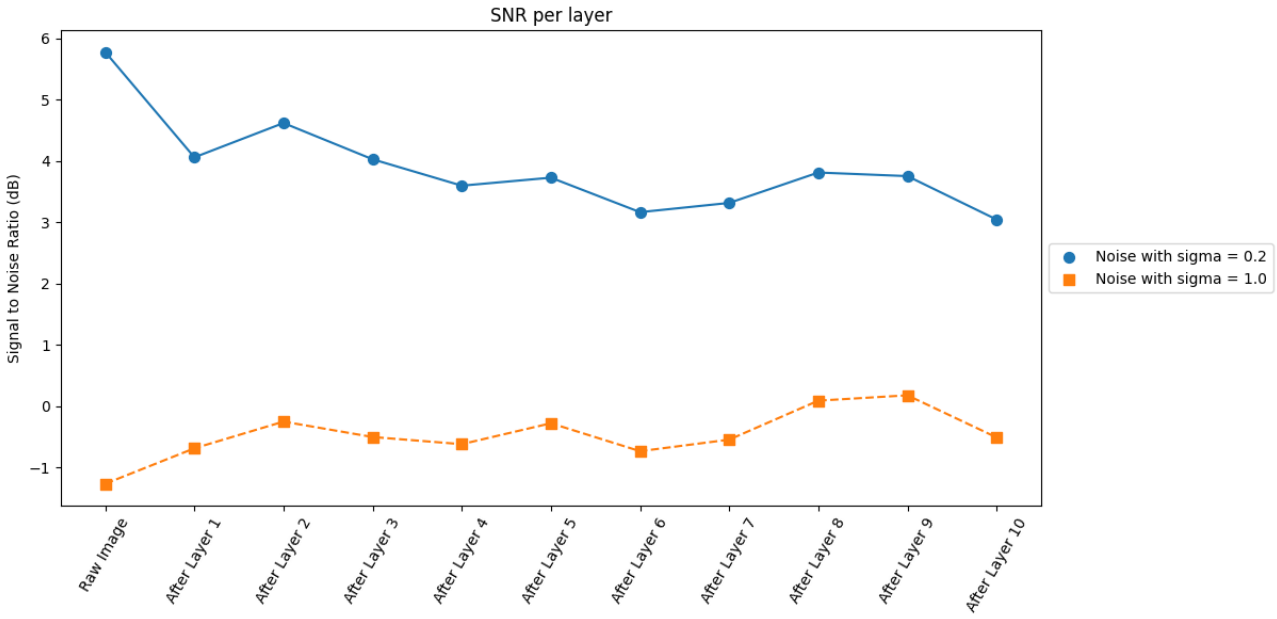

Figure 4.7: SNR Per Layer for Baseline Model with Gaussian Noise ($\mu = 0, \sigma = 0.2$) and Gaussian Noise ($\mu = 0, \sigma = 1.0$).

noise added at $\mu = 0, \sigma = 0.2$ are presented in Table 4.3 and Figure 4.8. The model with the pooling layer placed after the second convolutional layer shows a sharp increase in SNR immediately following layer 2, corresponding to the location of the pooling operation. This post-pooling SNR increase is observed across all configurations, although its magnitude decreases as pooling is positioned deeper in the network. Overall, the SNR tends to decline steadily with depth, suggesting a cumulative effect of noise as signals progress through the convolutional stack. This trend implies that pooling can help suppress noise and enhance the clarity of feature maps, but its impact varies depending on where it is introduced. When pooling occurs early in the network, it has a stronger influence on downstream representations. In such cases, useful detail may be discarded prematurely, impairing the model's ability to reconstruct or enhance important signal components in later layers. This effect is particularly pronounced in the model with pooling immediately after the first convolution, where SNR drops sharply and shows little sign of recovery. By contrast, placing the pooling layer toward the end of the network, as in the model with pooling after the tenth convolution, yields SNR values closely aligned with those of the baseline model, indicating minimal interference with the signal prior to classification. These findings underscore the importance of carefully selecting the position of pooling layers in convolutional neural networks. This highlights the trade-off between early noise suppression and the preservation of fine-grained spatial detail, which should be tailored to the specific task and data characteristics. It is worth noting that with higher noise levels, the benefits of early pooling may become more apparent, as stronger filtering could be required to maintain signal integrity. This possibility is explored further in Subsection 4.4.2

The results of the SNR calculation for each of the models with two pooling layers and

Table 4.3: SNR Per Layer for Models with One Pooling Layer.

	After Conv 1	**After Conv 2**	**After Conv 3**	**After Conv 4**
Raw Image	5.780421	5.780421	5.780421	5.780421
After Layer 1	5.135354	4.061133	4.061133	4.061133
After Layer 2	3.269352	5.217929	4.621880	4.621880
After Layer 3	2.965889	3.195196	4.746973	4.029476
After Layer 4	2.562884	2.805596	2.694072	4.370337
After Layer 5	2.726815	2.917060	2.841867	3.038345
After Layer 6	2.208713	2.373076	2.271984	2.357254
After Layer 7	2.271374	2.667337	2.563437	2.595408
After Layer 8	2.887028	3.093463	2.982200	3.065337
After Layer 9	2.826417	3.095257	2.899440	3.048267
After Layer 10	2.136876	2.425657	2.237883	2.375266

	After Conv 5	**After Conv 6**	**After Conv 7**	**After Conv 8**
Raw Image	5.780421	5.780421	5.780421	5.780421
After Layer 1	4.061133	4.061133	4.061133	4.061133
After Layer 2	4.621880	4.621880	4.621880	4.621880
After Layer 3	4.029476	4.029476	4.029476	4.029476
After Layer 4	3.598565	3.598565	3.598565	3.598565
After Layer 5	4.413700	3.729382	3.729382	3.729382
After Layer 6	2.674945	3.826175	3.166172	3.166172
After Layer 7	2.793531	2.697428	3.939150	3.316945
After Layer 8	3.320397	3.230895	3.388587	4.329132
After Layer 9	3.258034	3.199852	3.250582	3.277406
After Layer 10	2.554431	2.501111	2.584043	2.682145

	After Conv 9	**After Conv 10**
Raw Image	5.780421	5.780421
After Layer 1	4.061133	4.061133
After Layer 2	4.621880	4.621880
After Layer 3	4.029476	4.029476
After Layer 4	3.598565	3.598565
After Layer 5	3.729382	3.729382
After Layer 6	3.166172	3.166172
After Layer 7	3.316945	3.316945
After Layer 8	3.813275	3.813275
After Layer 9	4.214774	3.754750
After Layer 10	2.629038	3.490384

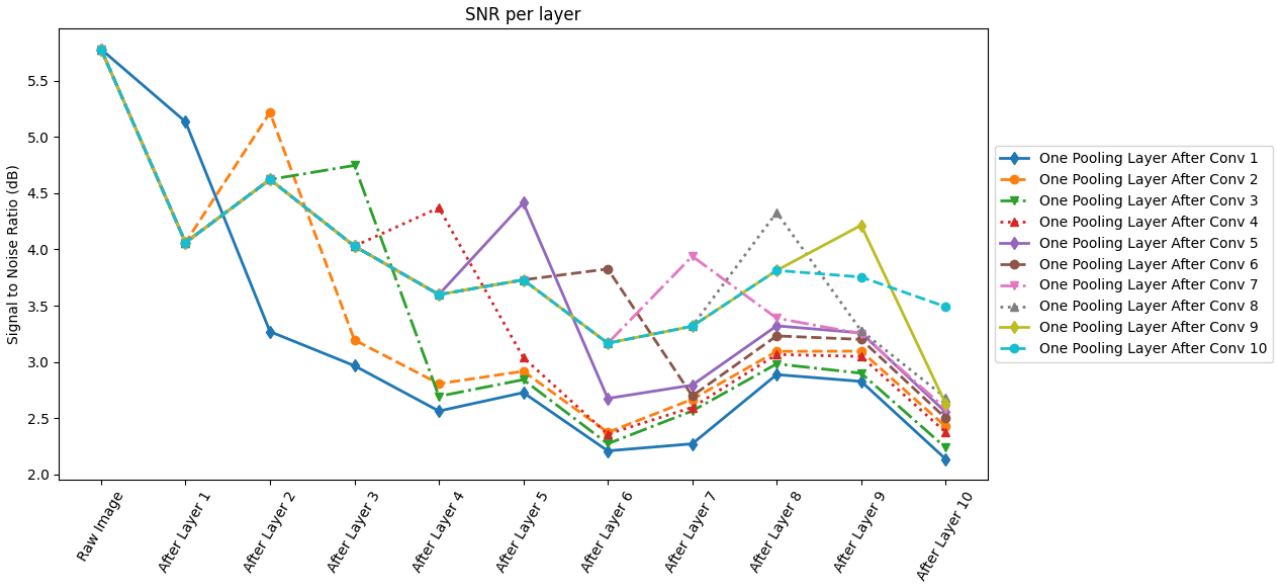

Figure 4.8: SNR Per Layer for Models with One Pooling Layers - Gaussian Noise ($\mu = 0, \sigma = 0.2$).

Gaussian noise added at $\mu = 0, \sigma = 0.2$ are shown in Figure 4.9. Due to the large number of model variations, the legend has been omitted for clarity. To facilitate interpretation, ten models were selected at random, and their results are shown in Table 4.4 and Figure 4.10. Despite the variety in pooling configurations, a consistent trend of decreasing SNR across layers is observed in all models, reaffirming the cumulative impact of noise as signals propagate through deeper layers of the network. Differences between models are primarily driven by the rate of SNR degradation, which is strongly influenced by pooling placement. For instance, the model with pooling after convolutional layers 1 and 3 exhibits a steep and continuous decline in SNR, ultimately resulting in the lowest signal quality among the selected examples. This pattern indicates that early, closely spaced pooling layers can prematurely discard informative features, whereas staggered pooling allows for better signal preservation. By contrast, the model with pooling after convolutional layers 6 and 7 retains a higher SNR throughout the network, indicating better preservation of signal despite the overall downward trajectory. This positioning likely allows earlier layers to extract more informative features before the resolution is reduced, thereby enhancing the model's ability to maintain signal integrity. Most notably, the model with pooling placed after layers 4 and 10 achieves the highest final SNR among the ten examined. This outcome suggests that a more balanced or staggered pooling strategy where one pooling layer is positioned near the middle and the other closer to the end may provide an effective compromise between noise reduction and information retention. Such a configuration allows the network to first capture rich representations before compressing them and again avoids excessive disruption immediately before classification. While early pooling may be advantageous in highly noisy conditions by filtering irrelevant variations, its benefits appear limited in the present setting with moderate noise levels. It is possible that under higher noise conditions, earlier pooling could offer greater benefits.

Table 4.4: SNR Per Layer for Models with Two Pooling Layers (random sample of 10).

	After Conv 1 & 3	**After Conv 1 & 4**	**After Conv 1 & 7**
Raw Image	5.781339	5.781339	5.781339
After Layer 1	5.132318	5.132318	5.132318
After Layer 2	3.264412	3.264412	3.264412
After Layer 3	3.219509	2.960245	2.960245
After Layer 4	1.886301	2.957867	2.556961
After Layer 5	2.080872	2.175654	2.720695
After Layer 6	1.583743	1.637805	2.202206
After Layer 7	1.753021	1.783771	2.656053
After Layer 8	2.313470	2.389915	2.475984
After Layer 9	2.229722	2.344713	2.370084
After Layer 10	1.539915	1.655996	1.684740

	After Conv 1 & 9	**After Conv 2 & 4**	**After Conv 2 & 7**
Raw Image	5.781339	5.781339	5.781339
After Layer 1	5.132318	4.060428	4.060428
After Layer 2	3.264412	5.211975	5.211975
After Layer 3	2.960245	3.189209	3.189209
After Layer 4	2.556961	3.147408	2.800195
After Layer 5	2.720695	2.400262	2.911273
After Layer 6	2.202206	1.752637	2.366396
After Layer 7	2.264887	2.137624	2.985303
After Layer 8	2.880193	2.520210	2.681772
After Layer 9	3.102131	2.605112	2.625881
After Layer 10	1.764229	1.920129	1.992816

	After Conv 2 & 9	**After Conv 3 & 8**	**After Conv 4 & 10**	**After Conv 6 & 7**
Raw Image	5.781339	5.781339	5.781339	5.781339
After Layer 1	4.060428	4.060428	4.060428	4.060428
After Layer 2	5.211975	4.619283	4.619283	4.619283
After Layer 3	3.189209	4.739905	4.024864	4.024864
After Layer 4	2.800195	2.688184	4.363438	3.593205
After Layer 5	2.911273	2.835777	3.031757	3.723164
After Layer 6	2.366396	2.264998	2.349912	3.817993
After Layer 7	2.659453	2.556239	2.587278	2.948634
After Layer 8	3.085142	3.267410	3.056932	2.906446
After Layer 9	3.344173	2.532942	3.039351	2.859216
After Layer 10	2.021005	1.931277	2.613423	2.169257

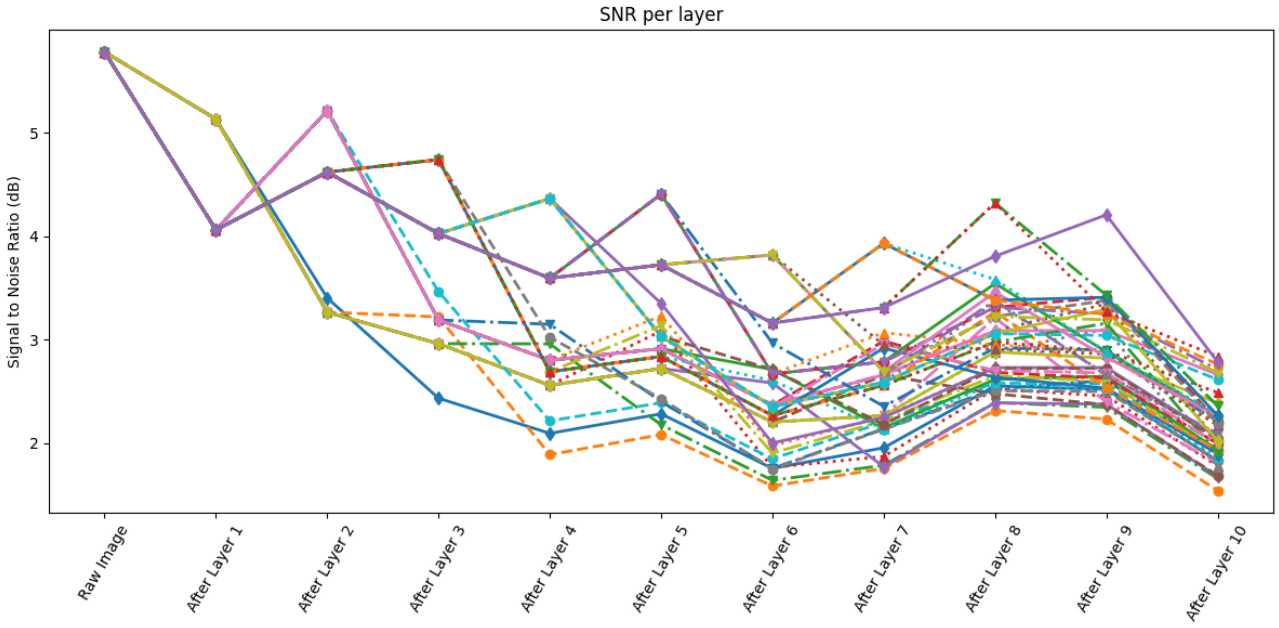

Figure 4.9: SNR Per Layer for Models with Two Pooling Layers - Gaussian Noise ($\mu = 0, \sigma = 0.2$).

All of the preceding calculations were carried out using pooling layers with a kernel size of 2×2. To investigate how the choice of pooling kernel size might affect signal retention, the same calculations were repeated for models with two pooling layers, this time using larger kernel sizes of 4×4 and 6×6.

The results for the 4×4 pooling kernels are presented in Table 4.5 and Figure 4.11. Although the general pattern of declining SNR across layers remains, some differences emerge compared to the 2×2 case. One key observation is that models with 4×4 pooling exhibit higher SNR values in the earlier layers of the network. This implies that larger pooling kernels may initially be effective at suppressing noise, leading to a cleaner signal in the immediate layers that follow. However, this comes at a potential cost. The broader aggregation window of a 4×4 kernel removes more spatial detail than smaller kernels, which can make it harder for the network to recover fine-grained features in deeper layers. This trade-off is evident in the varied performance of the different pooling configurations. Models such as those with pooling after layers 1 and 9, 2 and 9, 3 and 8, 4 and 10, and 6 and 7 exhibit a clear SNR spike immediately following the first pooling operation, consistent with the denoising effect of larger pooling kernels. These spikes are particularly prominent in the early and middle layers, reinforcing the idea that large kernels offer strong initial noise suppression. Yet, the persistence of this benefit varies. While some models maintain relatively high SNR throughout, others experience a steeper decline in the later layers. Once again, the model with pooling layers placed after layers 4 and 10 achieved the highest SNR across all ten layers, supporting the earlier observation that staggered pooling, especially with one layer late in the network, can offer a balance between noise reduction and information preservation. In contrast, the model with pooling after layers 1 and 3 and the model with pooling after layers

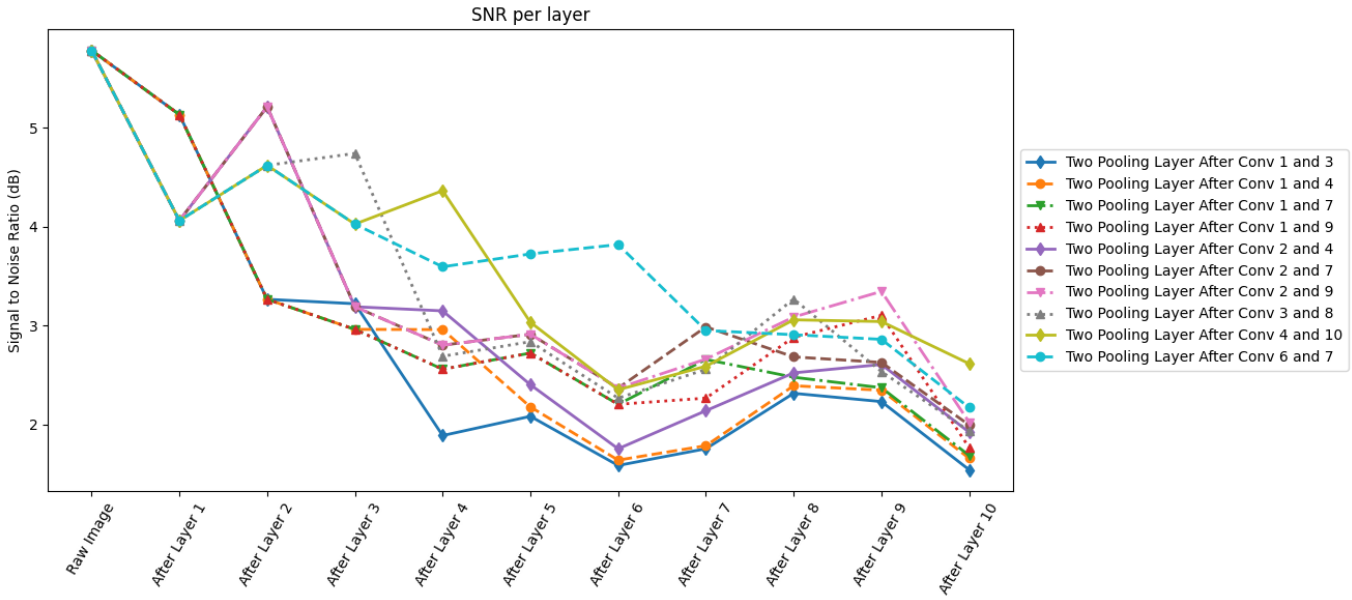

Figure 4.10: SNR Per Layer for Models with Two Pooling Layers (random sample of 10) - Gaussian Noise ($\mu = 0, \sigma = 0.2$).

1 and 7 showed the lowest final SNR values. These results reinforce the idea that early and concentrated pooling, even with large kernels, can lead to irreversible signal degradation, especially if no later pooling is available to further refine the representation. Overall, these findings suggest that larger pooling kernels can be more aggressive in filtering out noise in the early-stages, but this must be balanced against their potential to discard important spatial information. The ideal configuration likely depends on the nature and severity of the input noise. While 4×4 kernels may be beneficial under moderate noise levels, their advantages could become more pronounced in scenarios where noise is more dominant.

Table 4.5: SNR Per Layer for Models with Two Pooling Layers with Pooling kernel size 4×4 (random sample of 10).

	After Conv 1 & 3	**After Conv 1 & 4**	**After Conv 1 & 7**
Raw Image	5.781339	5.781339	5.781339
After Layer 1	6.999315	6.999315	6.999315
After Layer 2	2.585814	2.585814	2.585814
After Layer 3	2.571208	2.319059	2.319059
After Layer 4	1.040868	2.340410	1.894155
After Layer 5	1.471121	1.538862	2.249462
After Layer 6	0.897163	0.996741	1.669707
After Layer 7	0.979580	1.046322	2.022309
After Layer 8	1.796574	1.907776	1.783713
After Layer 9	1.707828	1.900413	1.670583
After Layer 10	0.989632	1.195890	0.988196

	After Conv 1 & 9	**After Conv 2 & 4**	**After Conv 2 & 7**
Raw Image	5.781339	5.781339	5.781339
After Layer 1	6.999315	4.060428	4.060428
After Layer 2	2.585814	6.316109	6.316109
After Layer 3	2.319059	2.617207	2.617207
After Layer 4	1.894155	2.567522	2.276143
After Layer 5	2.249462	1.734474	2.406010
After Layer 6	1.669707	1.111255	1.875006
After Layer 7	1.613149	1.484430	2.510890
After Layer 8	2.431281	1.864955	2.017868
After Layer 9	2.600091	2.030182	1.920188
After Layer 10	1.102877	1.290865	1.309860

	After Conv 2 & 9	**After Conv 3 & 8**	**After Conv 4 & 10**	**After Conv 6 & 7**
Raw Image	5.781339	5.781339	5.781339	5.781339
After Layer 1	4.060428	4.060428	4.060428	4.060428
After Layer 2	6.316109	4.619283	4.619283	4.619283
After Layer 3	2.617207	5.754940	4.024864	4.024864
After Layer 4	2.276143	2.226004	5.246429	3.593205
After Layer 5	2.406010	2.426735	2.638770	3.723164
After Layer 6	1.875006	1.847155	1.913199	4.511975
After Layer 7	2.223286	2.184484	2.108974	2.292799
After Layer 8	2.649871	2.838212	2.648263	2.219916
After Layer 9	2.901070	1.920530	2.646581	2.235068
After Layer 10	1.336907	1.301697	2.135550	1.592363

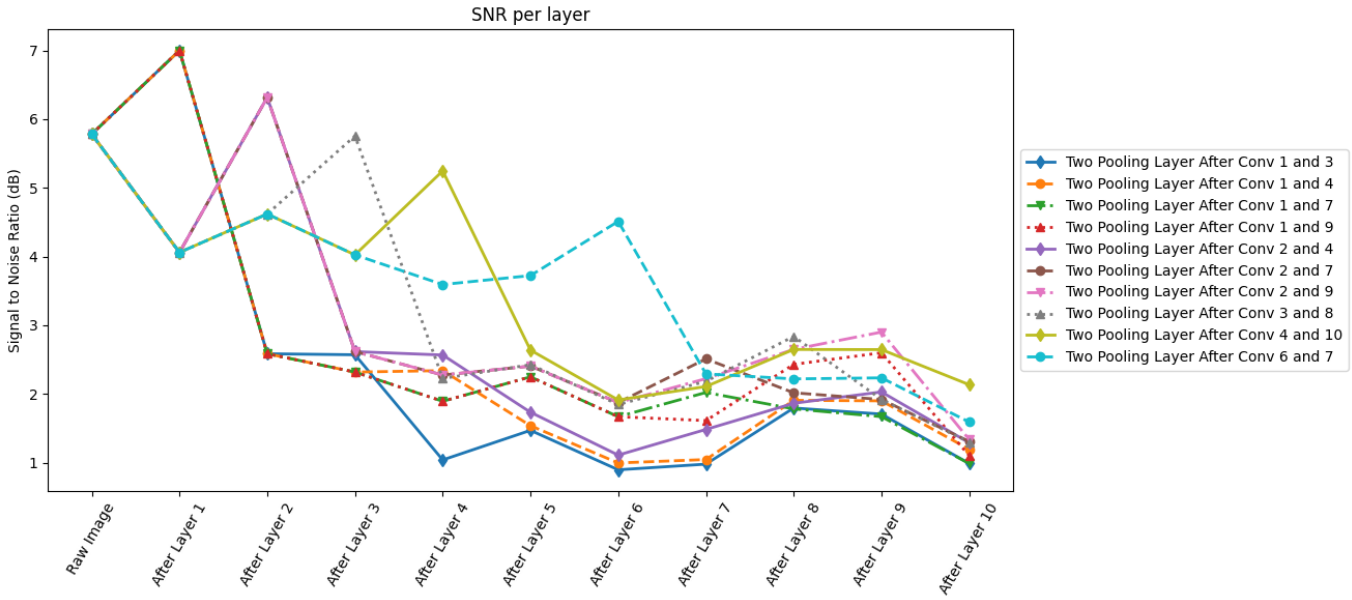

Figure 4.11: SNR Per Layer for Models with Two Pooling Layers with Pooling kernel size 4×4 (random sample of 10) - Gaussian Noise ($\mu = 0, \sigma = 0.2$).

The results for the two pooling layers with a kernel size of 6×6 are shown in Table 4.6 and Figure 4.12. The overall trend is consistent with that observed for the 4×4 kernel. Pooling continues to produce a noticeable increase in SNR immediately following each pooling operation, and the general trajectory across the network still reflects a gradual decline in SNR from layer to layer. However, a key distinction emerges in the magnitude of the final SNR values. Specifically, many models with a 6×6 kernel reach SNR values below 1, whereas for the 4×4 configuration, the minimum SNR remained above 1, and the 2×2 models exhibited even higher minima. This observation suggests that the increasingly aggressive spatial reduction introduced by larger pooling kernels can impair the network's ability to retain and reconstruct useful signal in deeper layers. Compared to the 4×4 configuration, the 6×6 models exhibit substantially lower final SNR values, with many falling below 1, indicating more severe loss of recoverable signal in deeper layers. In some configurations, the initial benefits of noise suppression from the 6×6 kernel are evident, as seen in models where SNR spikes occur in the early layers. However, these gains tend not to persist throughout the network. Compared to the 2×2 and 4×4 cases, where signal degradation was more gradual, the decline in the 6×6 models is steeper and more sustained, leading to poorer overall signal retention. These findings emphasize the diminishing returns of enlarging the pooling kernel beyond a certain point. While larger kernels can be effective in strongly noisy settings by eliminating local fluctuations, they may ultimately be too coarse for tasks requiring preservation of fine-grained spatial structure. Consequently, the choice of pooling kernel size must be guided not only by the architecture but also by the characteristics of the input data and the specific demands of the task. Further exploration in subsequent sections will consider whether the trend reverses under higher noise levels, where aggressive pooling might offer a comparative advantage in suppressing overwhelming noise.

Table 4.6: SNR Per Layer for Models with Two Pooling Layers.

	After Conv 1 & 3	**After Conv 1 & 4**	**After Conv 1 & 7**
Raw Image	5.781339	5.781339	5.781339
After Layer 1	7.221347	7.221347	7.221347
After Layer 2	2.098386	2.098386	2.098386
After Layer 3	1.583087	1.807096	1.807096
After Layer 4	0.081673	1.372733	1.428924
After Layer 5	0.674521	0.738169	1.900625
After Layer 6	0.014976	0.139001	1.255228
After Layer 7	0.144333	0.194136	1.172135
After Layer 8	1.079565	1.159792	1.040106
After Layer 9	1.016598	1.210860	0.987945
After Layer 10	0.285051	0.528893	0.328227

	After Conv 1 & 9	**After Conv 2 & 4**	**After Conv 2 & 7**
Raw Image	5.781339	5.781339	5.781339
After Layer 1	7.221347	4.060428	4.060428
After Layer 2	2.098386	6.390537	6.390537
After Layer 3	1.807096	2.137503	2.137503
After Layer 4	1.428924	1.572334	1.803872
After Layer 5	1.900625	0.859220	1.974694
After Layer 6	1.255228	0.294691	1.480704
After Layer 7	1.218656	0.643888	1.672507
After Layer 8	2.114000	1.070149	1.237081
After Layer 9	1.868705	1.227523	1.100424
After Layer 10	0.382700	0.494722	0.504697

	After Conv 2 & 9	**After Conv 3 & 8**	**After Conv 4 & 10**	**After Conv 6 & 7**
Raw Image	5.781339	5.781339	5.781339	5.781339
After Layer 1	4.060428	4.060428	4.060428	4.060428
After Layer 2	6.390537	4.619283	4.619283	4.619283
After Layer 3	2.137503	5.872701	4.024864	4.024864
After Layer 4	1.803872	1.875412	5.332750	3.593205
After Layer 5	1.974694	2.096683	2.251323	3.723164
After Layer 6	1.480704	1.530876	1.545000	4.577816
After Layer 7	1.822711	1.878692	1.685014	1.352832
After Layer 8	2.281067	2.127316	2.302340	1.349166
After Layer 9	2.118725	1.222708	2.291490	1.442033
After Layer 10	0.502759	0.593146	1.318414	0.836279

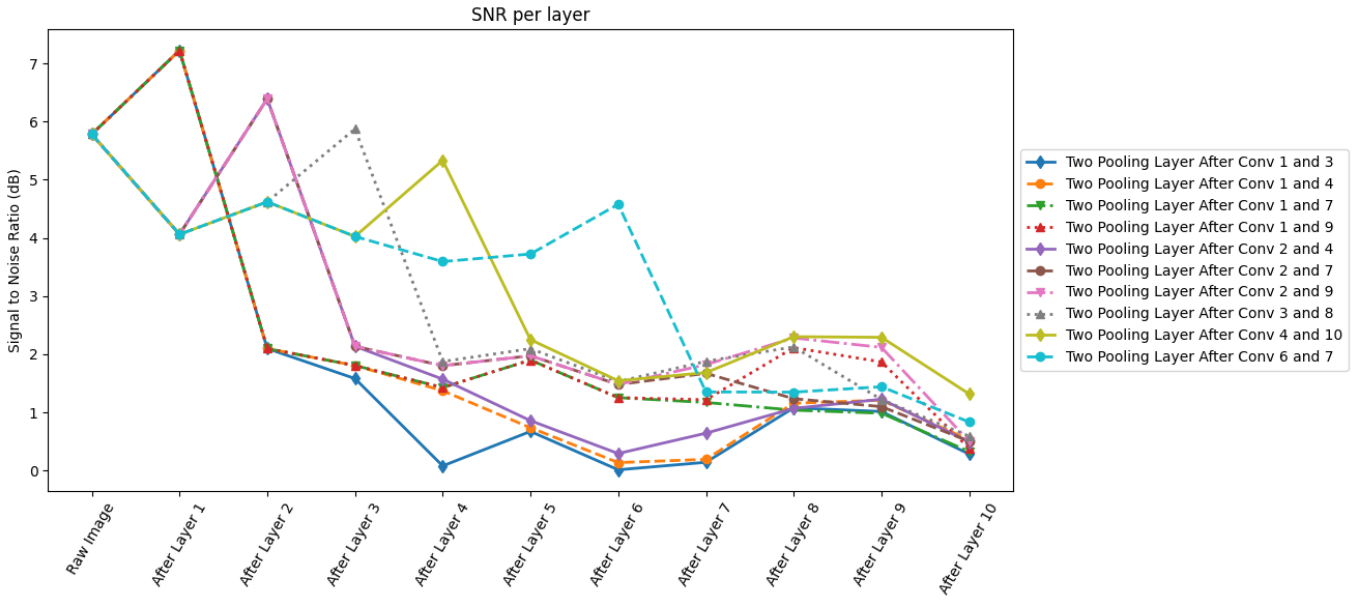

Figure 4.12: SNR Per Layer for Models with Two Pooling Layers with Pooling kernel size 6×6 (random sample of 10) - Gaussian Noise ($\mu = 0, \sigma = 0.2$).

4.4.2 SNR Calculation Across Layers for 10 Epochs

To better understand how noise propagates and accumulates through the network during training, SNR was computed across each convolutional layer over a span of ten training epochs. This approach enables a temporal view of how effectively the network manages to retain and enhance the signal in the presence of added Gaussian noise. Analysing SNR over multiple epochs is particularly important for identifying whether learning dynamics progressively improve or deteriorate the signal representation, and whether any patterns emerge that correlate with performance metrics such as accuracy. This insight is especially useful when evaluating baseline and ablated models, where architectural choices like pooling or batch normalisation may differentially affect signal preservation over time. Moving on to the calculation of SNR per layer over ten epochs, the results for the baseline model without pooling layers, incorporating Gaussian noise where $\mu = 0, \sigma = 0.2$, are presented in Figure 4.13. During the first epoch, the SNR remains relatively stable across the layers, although it follows a subtle downward trend. However, beginning from epoch 2, a marked and consistent decline in SNR is observed across the convolutional stack. The SNR decreases progressively with each epoch, eventually reaching negative values by epoch 10, where it concludes at approximately -6. A negative SNR implies that the noise dominates the signal, indicating that the learned representations are increasingly overwhelmed by irrelevant or noisy features. Furthermore, from epochs 2 through 10, the SNR becomes so diminished by the end of the network that it can no longer be reliably computed beyond the final layer, effectively "disappearing" after the last convolutional operation. This behaviour suggests not only an inability to recover signal in later layers but also a growing instability in the network's internal representations as training progresses. This pattern is mirrored in the associated accuracy

values, which drop substantially during the same period. While the baseline model maintains accuracy between 90% and 100% in the early-stages, this performance deteriorates sharply in the later epochs, underscoring a direct relationship between signal degradation and predictive reliability. The results suggest that without any pooling or regularising mechanisms, the baseline model struggles to preserve informative features in the presence of even modest noise. The accumulation of noise across layers over successive epochs leads to an erosion of the signal, which the model appears unable to counteract through learning alone. This reinforces the notion that architectural components such as pooling layers may be essential not only for abstraction and dimensionality reduction, but also for mitigating the adverse effects of noise across training time. Subsequent subsections will explore whether models incorporating pooling layers exhibit improved SNR trajectories over extended training durations.

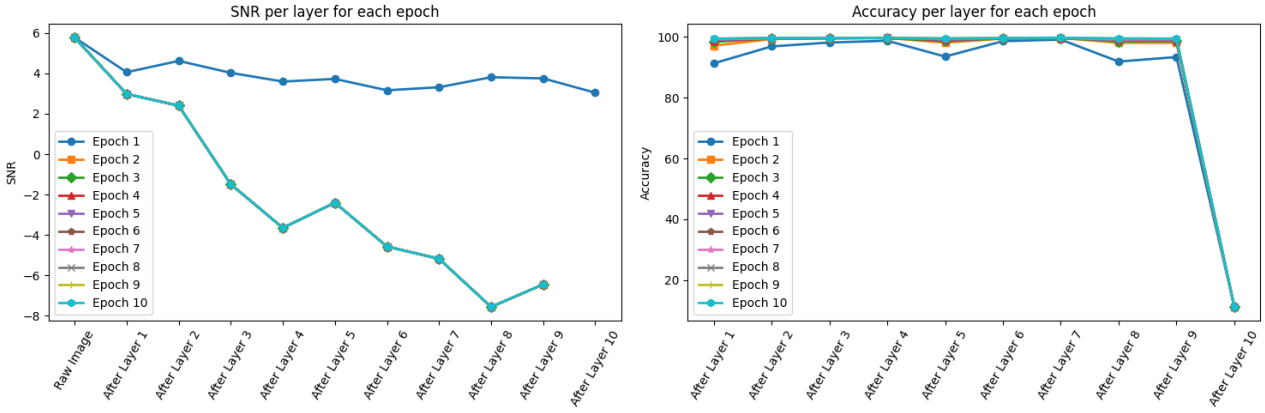

Figure 4.13: SNR Per Layer for Baseline Model with No Pooling Layers for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 0.2$).

The results when increasing the Gaussian noise to $\mu = 0, \sigma = 0.6$ and $\mu = 0, \sigma = 1.0$ are illustrated in Figures 4.14 and 4.15, respectively. Upon increasing the noise to $\sigma = 0.6$, we observe that while the SNR follows similar trends to noise with $\sigma = 0.2$, it does not reach as low negative values for epochs $2 - 10$. This observation indicates that the model is still able to retain a certain level of signal despite the increase in noise. However, when the noise is further increased to $\sigma = 1.0$, aside from a minor dip after layer 6, the SNR remains relatively stable at around -6 from layer 2 onwards for epochs 2 to 10, indicating that the model is struggling to maintain a SNR under these more extreme conditions. In both of these increased noise levels, the SNR remains noticeably higher during epoch 1, with the same disappearance after layer 10 in epochs 2 to 10. The sharp decline in accuracy at the tenth layer is still evident, suggesting that the model's ability to discriminate useful features becomes significantly impaired at this point. This behaviour is consistent with what is seen in previous noise settings, but the impact is more pronounced as the noise increases. This is likely due to the model's growing difficulty in learning relevant patterns as the noise becomes more overpowering. The increased noise also appears to exacerbate the issues related to the tenth convolutional layer, with accuracy degradation becoming more pronounced. For all levels of noise with this model, it could be suggested to reduce the number of convolutional layers to nine in order to improve performance, as accuracy

only seems to degrade significantly after the tenth layer. Furthermore, incorporating noise filtering techniques, such as denoising autoencoders, may help retain more of the signal before it reaches the convolutional layers, potentially improving the overall SNR. Additionally, the use of regularisation methods like dropout, batch normalisation, or L2 regularisation could prevent the model from overfitting to the noisy data and ensure better generalisation. A more thorough hyperparameter search could also reveal optimal settings under varying noise conditions, allowing the model to adapt more effectively. Lastly, it may be worthwhile to experiment with even higher levels of noise ($\sigma > 1.0$) to further assess how the model responds in extreme cases and refine strategies to enhance noise robustness. By applying these adjustments, it is likely that the model's performance under noisy conditions could be significantly improved.

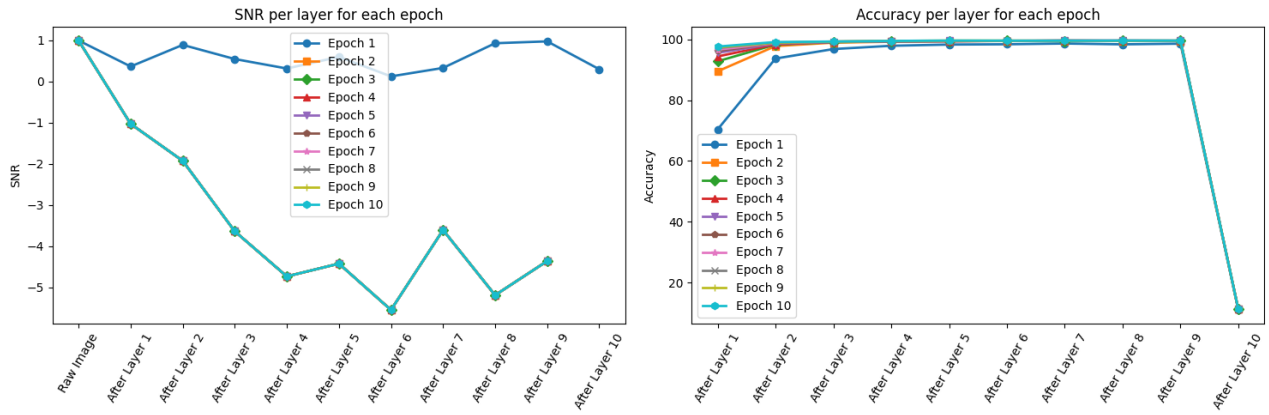

Figure 4.14: SNR Per Layer for Baseline Model with No Layers for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 0.6$).

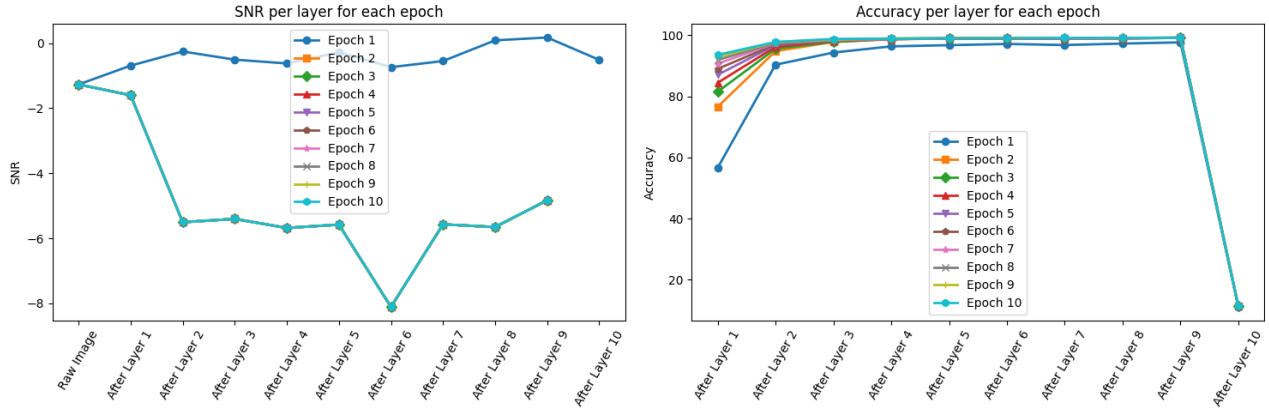

Figure 4.15: SNR Per Layer for Baseline Model with No Layers for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 1.0$).

The model selected to represent those with a single pooling layer was the one where the pooling layer was placed after the tenth and final convolutional layer. The results for Gaus-

sian noise with $\mu = 0, \sigma = 0.2$, $\mu = 0, \sigma = 0.6$, and $\mu = 0, \sigma = 1.0$ are shown in Figures 4.16, 4.17, and 4.18, respectively. Although the overall trend remains similar to that of the baseline models, there is some deviation in the SNR calculated for epochs 2 to 10 with noise levels of $\sigma = 0.2$ and $\sigma = 0.6$. With noise at $\sigma = 1.0$, the SNR calculated for epochs 2 to 10 appears to revert to being identical. This suggests that the model's ability to retain signal becomes significantly strained at higher noise levels, but at $\sigma = 0.2$ and $\sigma = 0.6$, it is able to maintain a somewhat stable SNR for a longer period during the early epochs. A notable observation is the earlier onset of the sharp drop in accuracy as the noise level increases. For noise with $\sigma = 0.2$, the drop occurs at layer 10, with $\sigma = 0.6$, it happens at layer 7, and with noise at $\sigma = 1.0$, the drop is observed as early as layer 4. This progression indicates that the model becomes increasingly sensitive to noise as it propagates through the network, and that larger noise levels disproportionately affect the deeper layers. The earlier drops in accuracy suggest that higher noise levels cause the model to struggle more with extracting meaningful features, which could result in degraded overall performance. Once again, the model takes progressively longer to learn the signal and stabilise in terms of accuracy as the noise increases. This pattern further confirms the critical impact of noise on model learning dynamics, and highlights the diminishing returns when trying to preserve signal integrity with higher noise levels. Given these observations, a recommendation would be to experiment with different noise regularisation techniques, such as noise augmentation or adaptive noise scaling, which could help the model generalise better under varying noise conditions. Finally, it may be worthwhile to explore reducing the number of convolutional layers or introducing early stopping mechanisms to prevent the model from overfitting to noisy data, which could improve the model's robustness and performance under high noise conditions.

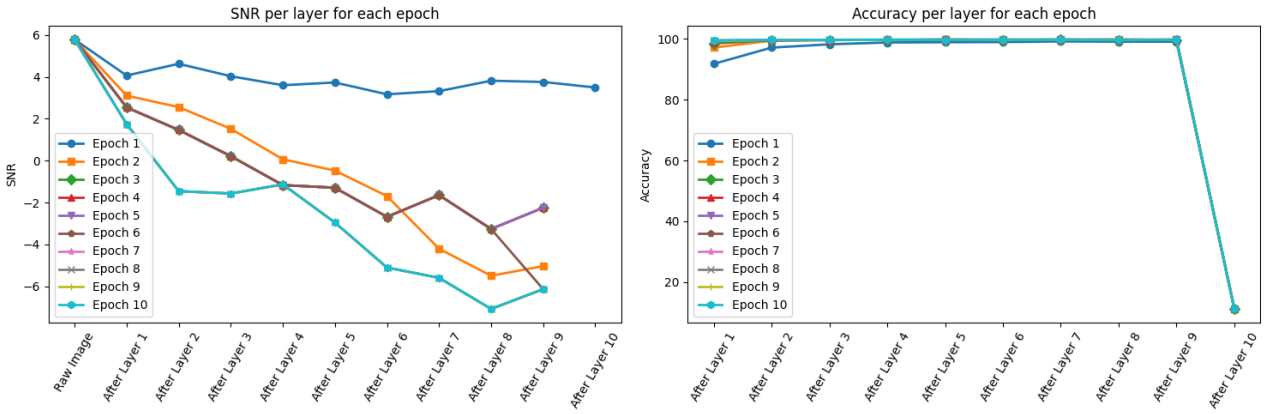

Figure 4.16: SNR Per Layer for Model with One Pooling Layer After Conv 10 for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 0.2$).

The model selected to represent those with two pooling layers was the one where pooling layers were placed after the first and third convolutional layers. The results for Gaussian noise with $\mu = 0, \sigma = 0.2$, $\mu = 0, \sigma = 0.6$, and $\mu = 0, \sigma = 1.0$ are presented in Figures 4.19, 4.20, and 4.21, respectively. It can be observed, particularly with noise levels of $\sigma = 0.6$ and $\sigma = 1.0$, that there is significant variation in the SNR calculation across all epochs.

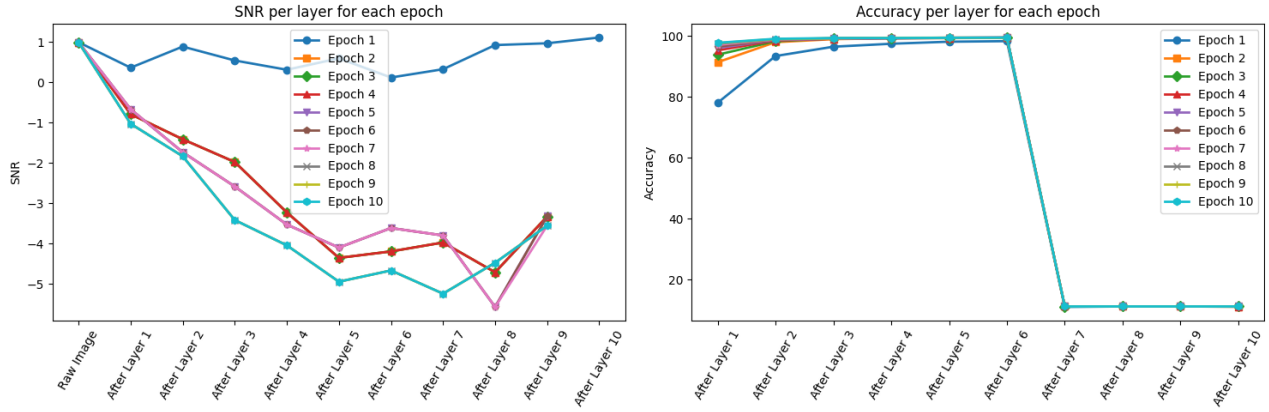

Figure 4.17: SNR Per Layer for Model with One Pooling Layer After Conv 10 for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 0.6$).

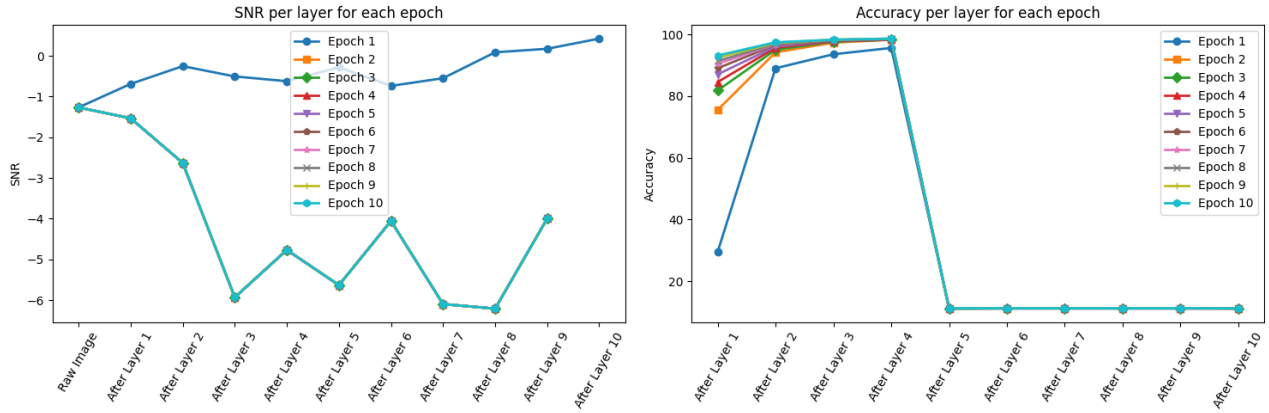

Figure 4.18: SNR Per Layer for Model with One Pooling Layer After Conv 10 for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 1.0$).

For noise with $\sigma = 0.6$, although the SNR drops sharply after layer 1, there is a slight recovery from layer 4 onwards, a trend that is similarly observed with noise at $\sigma = 1.0$. This implies that although early pooling layers may initially reduce the signal, they could facilitate faster signal recovery in subsequent layers, possibly due to the removal of irrelevant high frequency noise components early in the network. When the noise level is $\sigma = 1.0$, the model takes noticeably longer to interpret the signal and achieve high accuracy, though it improves progressively with each epoch. This gradual improvement under severe noise conditions implies that, with sufficient training time, the network is capable of compensating for noise introduced early in the pipeline. However, such compensation comes at the cost of increased training duration and potentially greater instability during the early-stages of training. Interestingly, with noise at $\sigma = 0.6$, accuracy decreases sharply after layer 8, whereas for noise levels of $\sigma = 0.2$ and $\sigma = 1.0$, this decline only occurs after layer 10. This indicates a potential sensitivity of the model to intermediate noise levels, where the signal might be partially degraded yet not completely overwhelmed, causing a sudden drop in performance. Based on these observations, it may be beneficial to incorporate an early stopping criterion around layer 8 in scenarios involving moderate noise. Doing so could help preserve accuracy by halting further convolutions before significant degradation occurs. Further recommendations include re-evaluating the position of pooling layers. While placing them early may aid in noise mitigation, it may also risk premature signal loss. A hybrid approach, such as adaptive pooling layer placement based on layer-wise SNR feedback, could optimise both noise reduction and signal preservation. Additionally, implementing skip connections or residual pathways might support signal recovery in deeper layers, particularly in the presence of higher noise. Finally, fine tuning the learning rate schedule or incorporating regularisation techniques, such as dropout or batch normalisation, could improve model stability and convergence in noisy conditions.

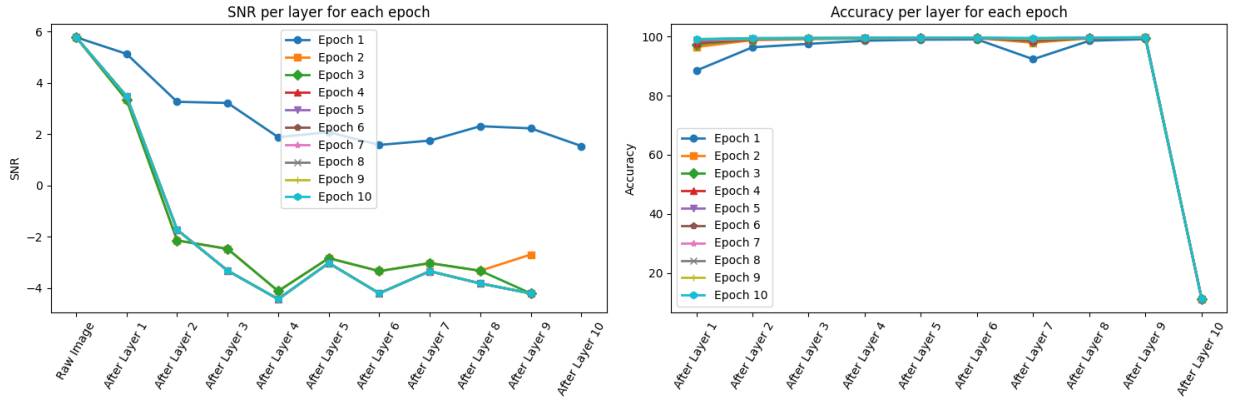

Figure 4.19: SNR Per Layer for Model with Two Pooling Layers After Conv 1 and Conv 3 for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 0.2$).

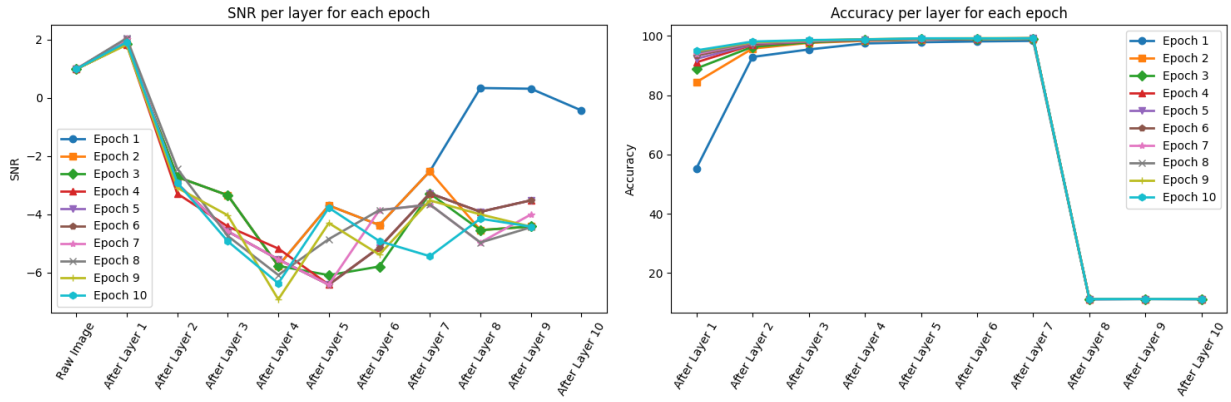

Figure 4.20: SNR Per Layer for Model with Two Pooling Layers After Conv 1 and Conv 3 for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 0.6$).

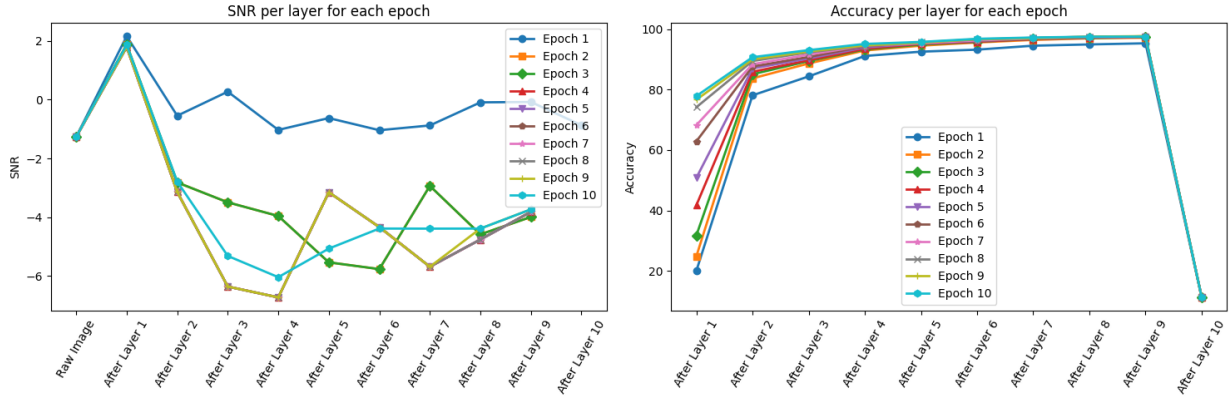

Figure 4.21: SNR Per Layer for Model with Two Pooling Layers After Conv 1 and Conv 3 for Ten Epochs - Gaussian Noise ($\mu = 0, \sigma = 1.0$).

Across all architectures, increasing noise delays convergence, shifts the onset of signal degradation earlier in the network, and amplifies instability in deeper layers. Robustness may be improved by reducing network depth, carefully staggering pooling layers, and incorporating regularisation, such as batch normalisation, dropout, and L2 weight decay, alongside early stopping under moderate noise conditions. In addition, skip connections or adaptive pooling strategies may support signal recovery in deeper layers, particularly under high-noise regimes. Finally, targeted hyperparameter tuning, including learning rate scheduling, and explicit denoising approaches, such as denoising autoencoders or noise-aware augmentation, could further improve performance as noise severity increases.

4.5 Conclusion

This initial investigation into the impact of SNR across convolutional layers provides important insights into how noise propagates through a model inspired by the AlexNet architecture.

The results reveal clear trends in how SNR evolves across depth and pooling configurations. In all cases, SNR tended to decrease as layer depth increased, although pooling operations mitigated this decline to varying degrees. Models employing smaller pooling kernels preserved higher SNR values in the earlier layers, while those with larger kernels demonstrated improved noise suppression in deeper layers. These findings suggest that pooling plays a critical role in balancing information retention and noise reduction throughout the network. Moreover, the results indicate that models exceeding nine convolutional layers experience considerable signal loss, resulting in diminished accuracy under noisy conditions. This outcome reinforces the observation that overly deep networks, when not supported by mechanisms such as residual connections or normalisation layers, can perform suboptimally in the presence of noise.

When trained over multiple epochs, all models displayed an overall decline in SNR across later epochs, yet those containing two pooling layers showed stronger resilience in maintaining signal integrity. The variations in SNR behaviour across different noise levels further emphasise the importance of pooling layer placement and kernel size. Models that pooled early and again mid-network tended to exhibit partial recovery of SNR, highlighting the benefits of distributing pooling operations to manage feature compression more effectively across layers.

This study illustrates the response of an AlexNet based CNN to Gaussian noise and examines how architectural components, such as pooling depth and kernel size, affect signal retention. These findings underpin the enhancement of model robustness through noise aware architecture design. In the subsequent chapter, this analysis is expanded to encompass a collection of more sophisticated CNN designs, including VGG, ResNet, DenseNet, Inception, Xception, and EfficientNet, which are assessed on a more intricate image dataset. By comparing SNR propagation across these deeper and more sophisticated models, the next phase of this research aims to uncover how architectural innovations influence noise resilience and overall classification performance in challenging visual environments. The MNIST based experiments in this chapter provide a baseline for anticipating how SNR and performance may behave in more complex imaging domains. In particular, the results show that SNR typically degrades with depth under noise, and that pooling placement and kernel size mediate a clear trade-off between noise suppression and information retention, with excessive depth leading to marked signal loss and reduced accuracy in the absence of additional stabilising mechanisms. In medical imaging, where noise is intrinsic and class boundaries are less visually distinct, it is therefore expected that SNR degradation may be more pronounced and that robustness will depend strongly on architectural mechanisms that regulate information flow. Chapter 5 builds on these expectations by evaluating whether the trends observed here persist on clinically realistic MRI data and by comparing SNR propagation and feature separability across modern CNN architectures.

Chapter 5

Extending SNR Analysis to Complex CNN Architectures and Datasets

5.1 Introduction

In this chapter, we perform a comparative analysis of SNR propagation of several complex CNN models such as VGG, ResNet, DenseNet, Inception, Xception, and EfficientNet using a dataset of MRI scans of brain tumours. Compared to the simpler MNIST dataset used in the previous chapter, this dataset presents a far more complex visual and structural challenge, characterised by heterogeneous textures, varying tumour morphologies, and realistic acquisition noise typical of clinical imaging. Analysing SNR propagation on such data is therefore crucial for understanding how deep networks handle noisy biomedical inputs and for assessing their robustness in clinical contexts where data quality cannot always be guaranteed. For each model, the SNR is computed at successive intermediate layers to quantify how effectively relevant information is preserved as it propagates through the network. The SNR is estimated by isolating signal and background regions in each test image using the tumour masks, then evaluating activation strength within these regions. This offers a direct assessment of the network's ability to preserve distinguishing tumour characteristics in relation to background noise. In parallel, we assess representational separability by training a logistic regression classifier (a linear probe) on pooled activations from each layer and measuring classification accuracy on the test set. This joint analysis of SNR and linear separability enables investigation into whether higher SNR corresponds to better feature discrimination and how this relationship evolves across different architectural designs. By extending the SNR analysis from the AlexNet based model in the previous chapter to these complex CNN architectures, this study aims to uncover how modern design innovations such as residual connections, dense feature reuse, and compound scaling affect signal preservation, noise suppression, and classification performance in complex medical imaging. The insights gained here provide a deeper understanding of how deep CNNs process noisy biomedical data and contribute to the broader goal of developing noise resilient architectures for reliable diagnostic image analysis.

5.2 Dataset

The dataset used in this chapter consists of MRI scans of brain tumours [56]. Although MRI data are typically stored as three-dimensional tensors, this dataset is provided as two-dimensional slices, enabling direct use for model training. The images are contrast enhanced, T1 weighted MRI scans (CE-MRI), comprising 3064 slices from 233 patients. The dataset includes 708 meningiomas, 1426 gliomas, and 930 pituitary tumours. For each slice, tumour regions have been manually segmented and verified by experienced radiologists. Figure 5.1 shows representative samples from each tumour class, with the top row showing the original MRI slices and the bottom row displaying the corresponding tumour masks. This dataset is particularly well suited to the work in this chapter, as the inclusion of tumour masks enables precise extraction of the signal, with the remaining image content treated as background noise. Figure 5.2 illustrates examples where a signal block (green, extracted within the mask indicating the tumour) and a matched background noise block (red) have been extracted. To obtain representative background noise samples, a dedicated procedure was implemented to ensure that the background block was spatially distinct from the tumour region while remaining in its immediate vicinity. First, the tumour region was identified from the binary mask, and its bounding box was determined. A background block of identical size was then sampled from a nearby location within a maximum offset of 200 pixels in both the x- and y-directions. To avoid spatial overlap and ensure that the background patch contained no tumour signal, two constraints were imposed:

- The candidate block was required to have no pixel overlap with the tumour bounding box.
- The binary mask within the selected region had to contain only zeros (i.e. non-signal regions).

If a candidate failed either condition, a new location was sampled at random within the defined offset range, and the process was repeated until a valid region was found or a maximum number of attempts was reached. This procedure introduces a controlled element of randomness to the background selection, preventing sampling bias while ensuring that the noise block is drawn from a spatially relevant region of the same MRI slice.

5.3 Classification and Results

To optimise model performance, a systematic hyperparameter tuning procedure was conducted across several deep learning architectures, including DenseNet121 [22], Xception [19], VGG19 [21], ResNet50 [3], ResNet50V2 [3], VGG16 [21], InceptionV3 [23], InceptionResNetV2 [24], and EfficientNetB0 [20]. Each model was implemented using transfer learning, with the base network initialised on ImageNet [12] weights and the final classification layer replaced with a task-specific output head. The pre-trained layers were frozen during the tuning phase to reduce computational cost and preserve previously learned low-level rep-

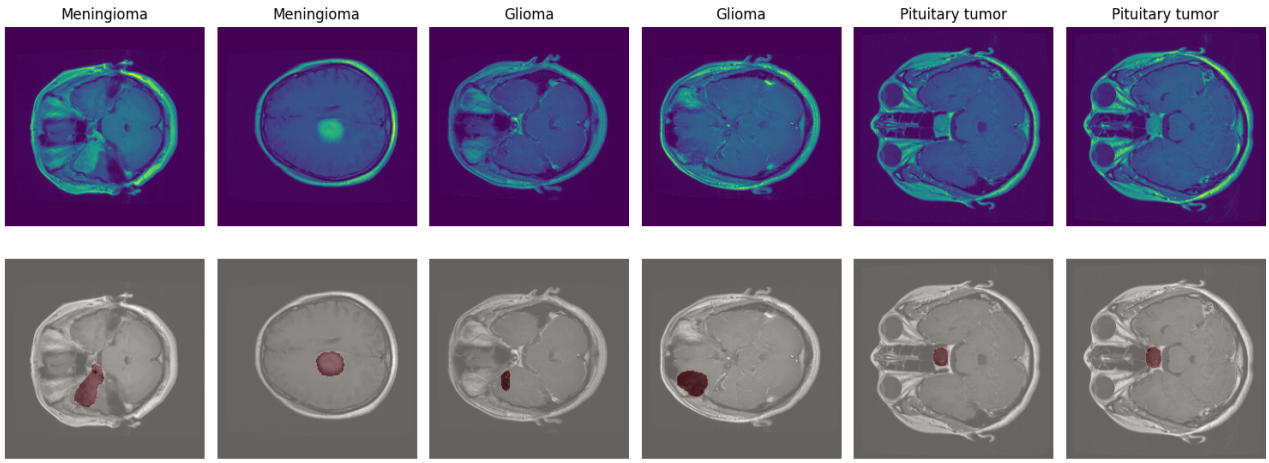

Figure 5.1: Representative samples from the brain tumour MRI dataset, showing contrast-enhanced T1-weighted slices (top row) and corresponding manually annotated tumour masks (bottom row).

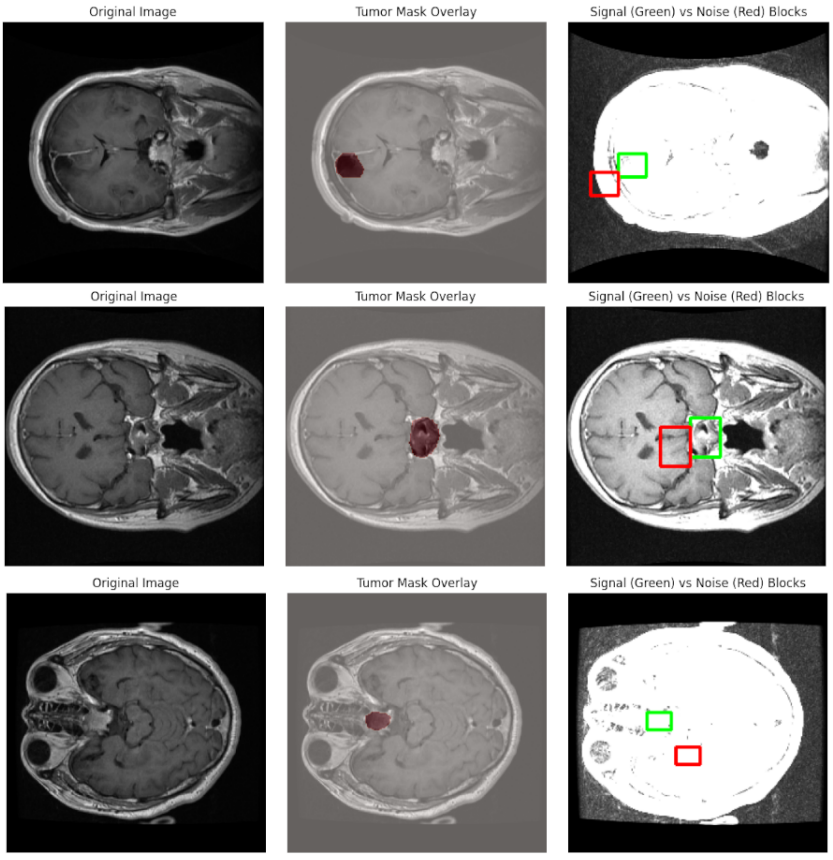

Figure 5.2: Illustration of signal and noise extraction from MRI slices, where tumour regions (green) define the signal and spatially matched non-tumour regions (red) define background noise for SNR computation.

resentations, thereby allowing the optimisation process to focus on the new classification layer.

5.3.1 Hyperparameter Search Strategy

Hyperparameter tuning was carried out using the RandomSearch algorithm from the Keras Tuner framework [57]. This approach was selected over exhaustive grid search due to its superior efficiency when exploring large, multi-dimensional parameter spaces [57]. Empirical evidence shows that random search can achieve comparable or better results with fewer evaluations, particularly when only a subset of hyperparameters strongly influences performance. The search space was designed to target key training dynamics, including the learning rate ($\{1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}\}$), batch size ($\{32, 64, 128\}$), optimiser type (SGD [58], Adam [59], RMSprop [60]), and momentum ($\{0.8, 0.9, 0.99\}$) where applicable. For each architecture, a maximum of 10 trials was conducted, with each trial trained for 10 epochs. The best performing configuration was selected based on the highest validation F1-score.

5.3.2 Data Splitting Protocol

The dataset was partitioned into independent training, validation, and test subsets to enable model development and evaluation. To prevent bias in model assessment, the held out test set was sampled randomly without stratification, ensuring that it represents an unbiased portion of the original dataset. The remaining data (90%) were then split into training and validation subsets using stratified sampling to preserve the original class distribution during model training. This approach maintains balanced representation across tumour types in the subsets used for optimisation, while keeping the test set fully independent of the stratification process. To ensure spatial correspondence between the input MRI images and their segmentation masks, the same index based splitting procedure was applied to the mask arrays. This guarantees that each image and its corresponding mask reside in the same subset and that there is no leakage of patient specific information between splits. All images were subsequently converted to three-channel RGB format to match the expected input dimensions of the pretrained CNN architectures employed during transfer learning. Finally, the resulting partitions were validated by inspecting class distributions and tensor shapes to confirm correct data alignment across images, masks, and labels, as well as consistent preservation of tumour category proportions in the training and validation subsets.

5.3.3 Optimisation Objective

The decision to optimise models using the F1-score rather than accuracy was motivated by the inherent class imbalance within the dataset and the need for balanced performance across tumour types. As discussed in Section 2.9, accuracy alone can be misleading in imbalanced classification settings, as it tends to be dominated by the majority class while under representing minority categories. In contrast, the F1-score, defined as the harmonic

mean of precision and recall, provides a more informative and robust measure of model performance by penalising extreme disparities between these two complementary metrics. By employing the F1-score as the validation objective during hyperparameter tuning, the optimisation process explicitly targeted models capable of achieving an improved trade-off between false positives and false negatives. This focus was particularly important for ensuring diagnostic reliability across all tumour classes, rather than favouring the most prevalent type. The best configurations selected according to this criterion were subsequently retrained to evaluate their convergence and generalisation behaviour.

5.3.4 Training Dynamics and Convergence Behaviour

The training and validation curves presented in this section correspond to the best performing models obtained from the hyperparameter optimisation process described in Subsection 5.3.3. The following analysis therefore characterises the learning behaviour of these optimised models during fine tuning, providing insight into convergence stability, generalisation, and potential signs of overfitting. To evaluate the learning stability and convergence characteristics of each architecture, training and validation accuracy and loss were monitored over ten epochs using the best performing hyperparameter configurations identified during tuning. Architectures such as DenseNet121 (Figure 5.3), InceptionV3 (Figure 5.4), ResNet50V2 (Figure 5.5), and InceptionResNetV2 (Figure 5.6) demonstrated smooth convergence with minimal divergence between training and validation curves, suggesting stable optimisation and effective feature transfer. In these networks, validation accuracy remained closely aligned with training accuracy, and both loss curves decreased steadily across epochs, indicating a low risk of overfitting under the chosen regularisation and learning rate settings. By contrast, models including EfficientNetB0 (Figure 5.7), VGG16 (Figure 5.8), and VGG19 (Figure 5.9) displayed more oscillatory loss patterns, particularly during the middle epochs. In these cases, validation loss fluctuated or increased slightly while training loss continued to decline, suggesting mild overfitting or local optimisation instability. This behaviour is most pronounced in the VGG architectures, which lack residual or dense connectivity and therefore rely more heavily on fine tuned learning rate control to stabilise gradient updates. The Xception (Figure 5.10) model exhibited moderately stable training behaviour, with validation loss remaining consistently higher than training loss but showing no severe divergence. This pattern indicates a degree of overfitting but overall steady convergence once early fluctuations subsided. In contrast, ResNet50 (Figure 5.11) showed an initially well aligned loss trend followed by a sharp late epoch increase in validation loss, also pointing to slight overfitting after convergence. The convergence behaviour across models indicates that the hyperparameter tuning phase produced predominantly well regularised configurations that can learn effectively, with only little instability noted in architectures devoid of internal normalisation or shortcut connections. Having confirmed stable convergence and limited overfitting across most architectures, the next step was to assess the generalisation capability of these optimised models on the independent test set. The following subsection presents the final evaluation metrics, providing an unbiased comparison of model performance under identical conditions.

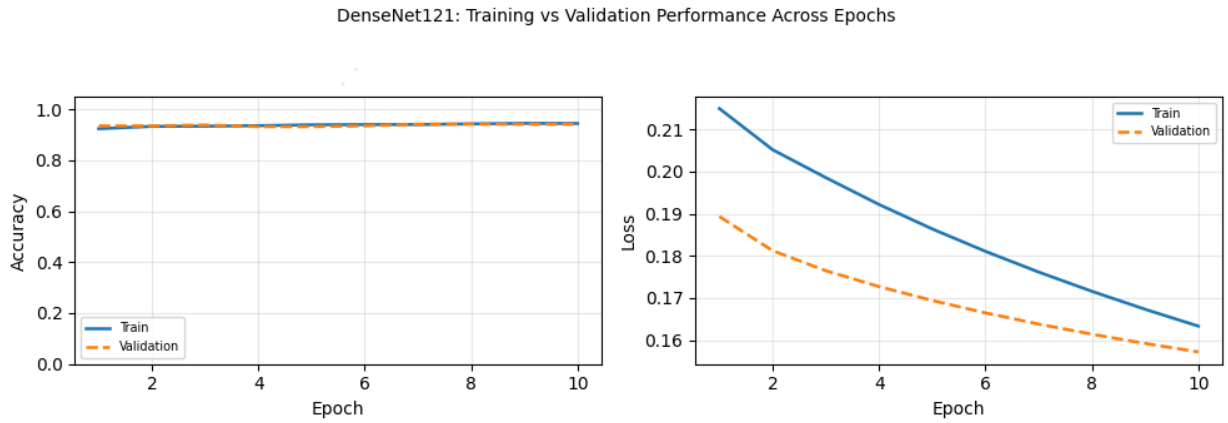

Figure 5.3: Training and validation accuracy and loss curves for DenseNet121 over ten epochs using the optimised hyperparameter configuration.

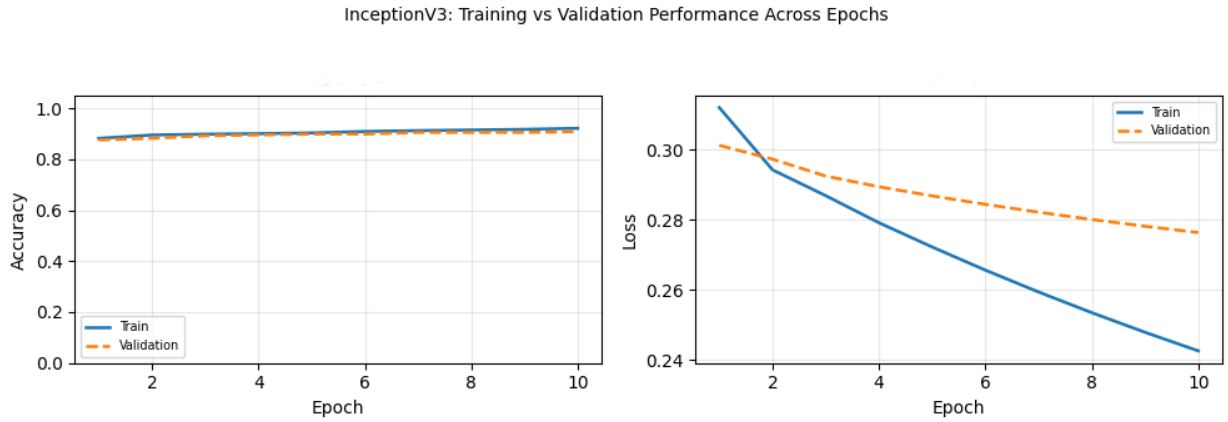

Figure 5.4: Training and validation accuracy and loss curves for InceptionV3 over ten epochs using the optimised hyperparameter configuration.

Figure 5.5: Training and validation accuracy and loss curves for ResNet50V2 over ten epochs using the optimised hyperparameter configuration.

Figure 5.6: Training and validation accuracy and loss curves for InceptionResNetV2 over ten epochs using the optimised hyperparameter configuration.

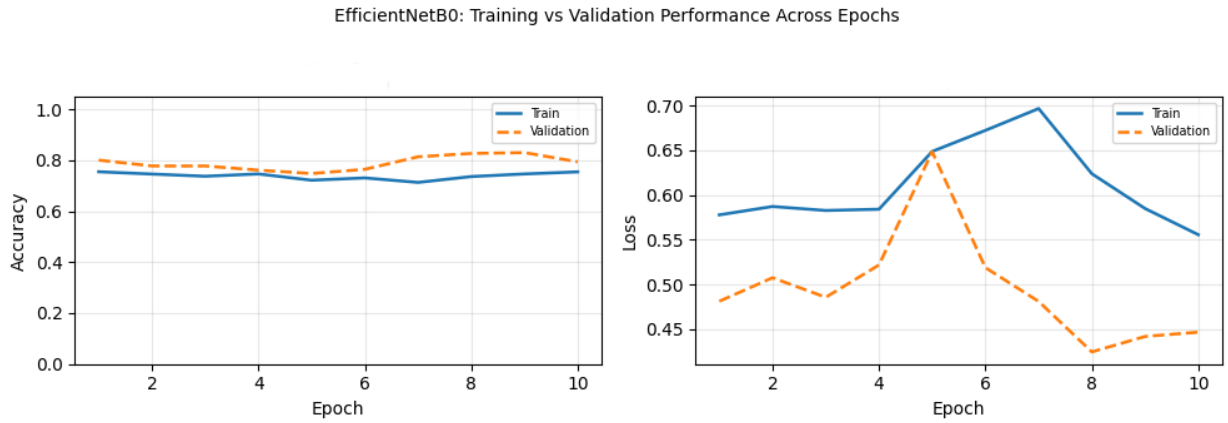

Figure 5.7: Training and validation accuracy and loss curves for EfficientNetB0 over ten epochs using the optimised hyperparameter configuration.

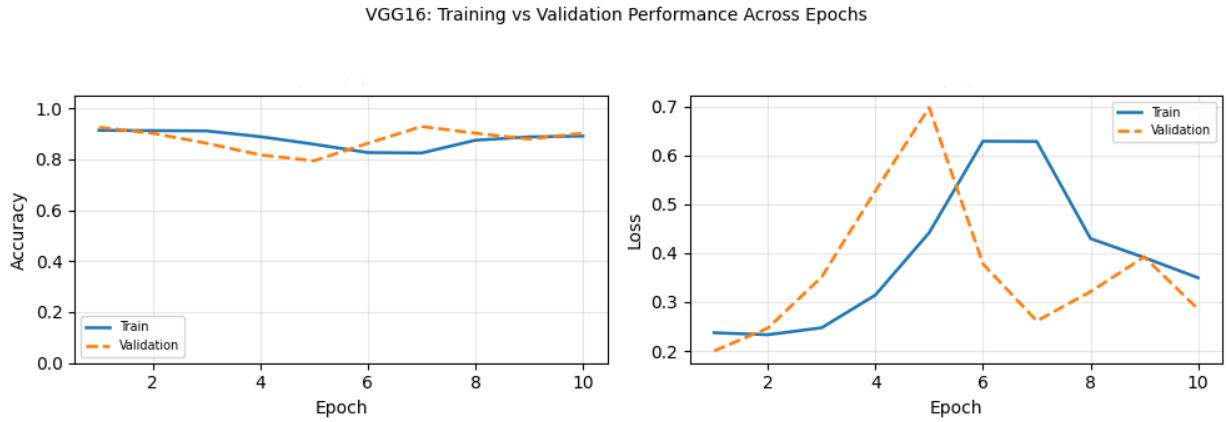

Figure 5.8: Training and validation accuracy and loss curves for VGG16 over ten epochs using the optimised hyperparameter configuration.

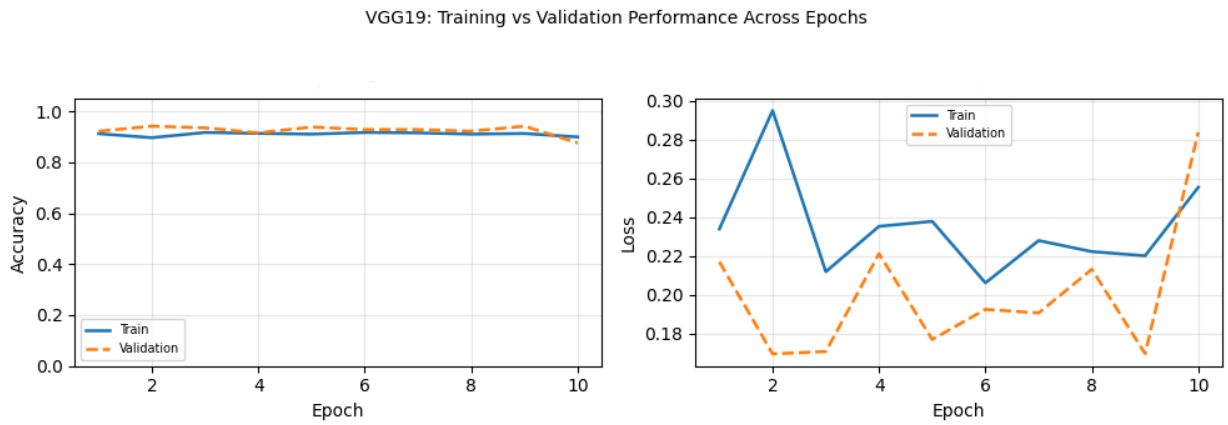

Figure 5.9: Training and validation accuracy and loss curves for VGG19 over ten epochs using the optimised hyperparameter configuration.

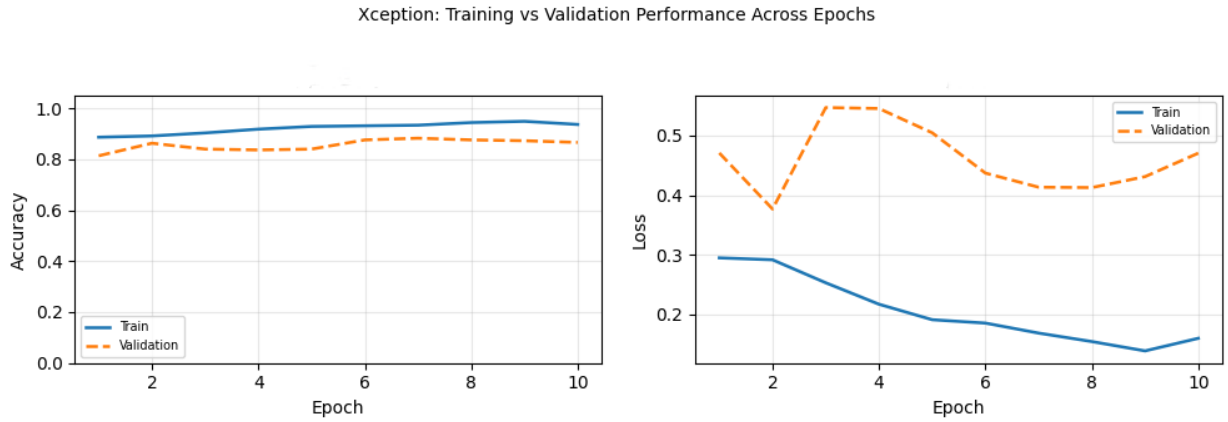

Figure 5.10: Training and validation accuracy and loss curves for Xception over ten epochs using the optimised hyperparameter configuration.

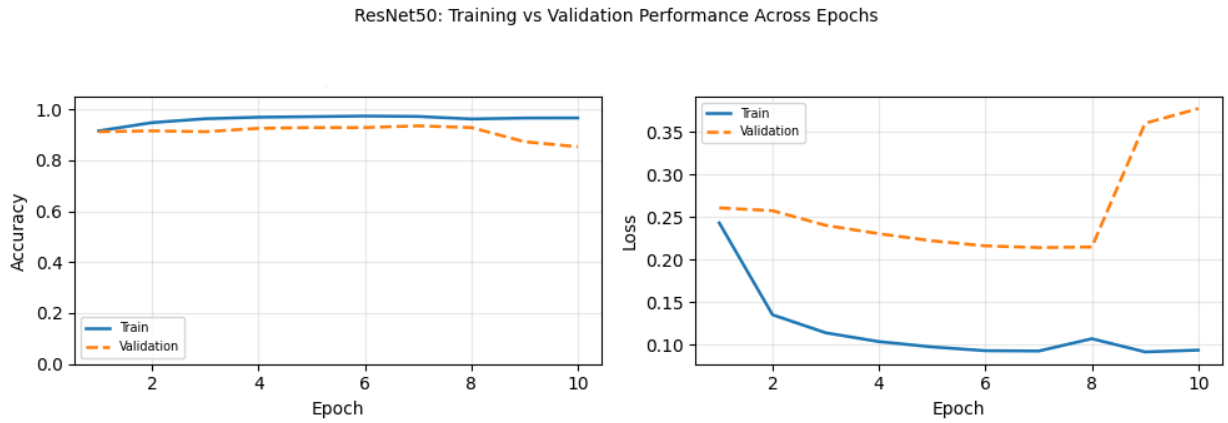

Figure 5.11: Training and validation accuracy and loss curves for ResNet50 over ten epochs using the optimised hyperparameter configuration.

5.3.5 Final Test Evaluation

Each best performing model selected according to its highest validation F1-score was evaluated on the independent test set to assess generalisation performance. Figure 5.12 reports the final classification metrics, namely accuracy, precision, recall, and F1-score, as defined in Section 2.9. These results provide an unbiased comparison of predictive performance across the optimised architectures. Among the evaluated models, ResNet50 achieved the highest overall performance, with a test accuracy of 0.928 and an F1-score of 0.921, indicating excellent generalisation and balanced discrimination across tumour classes. This outcome can be attributed to ResNet50's residual skip connections, which promote stable gradient flow and deeper representational capacity. DenseNet121 (F1 = 0.896) and VGG19 (F1 = 0.886) also performed competitively, the former benefiting from dense connectivity that enhances feature reuse and gradient propagation. The VGG models displayed slightly lower F1-scores, consistent with their more oscillatory training behaviour and lack of residual or dense pathways. In contrast, EfficientNetB0 and InceptionV3 attained lower F1-scores (0.745 and 0.791, respectively), suggesting reduced adaptability to the dataset scale or feature distribution. InceptionResNetV2 and ResNet50V2 demonstrated moderate but stable results (F1 $\approx 0.84-0.85$). Overall, architectures incorporating residual or densely connected blocks outperformed those without such mechanisms, confirming that shortcut based feature propagation and gradient stability contribute to improved class sensitivity and overall generalisation. These findings highlight that the F1-driven optimisation strategy successfully prioritised models with balanced precision-recall trade-offs, yielding robust predictive performance on unseen data. Residual and densely connected architectures, particularly ResNet50 and DenseNet121, proved most effective under the applied tuning and evaluation framework. While the preceding analysis quantified overall predictive performance, it did not address how information and noise are internally represented and transformed within each network. The following section therefore examines SNR propagation across layers to elucidate how architectural design influences the preservation and amplification of meaningful signal components.

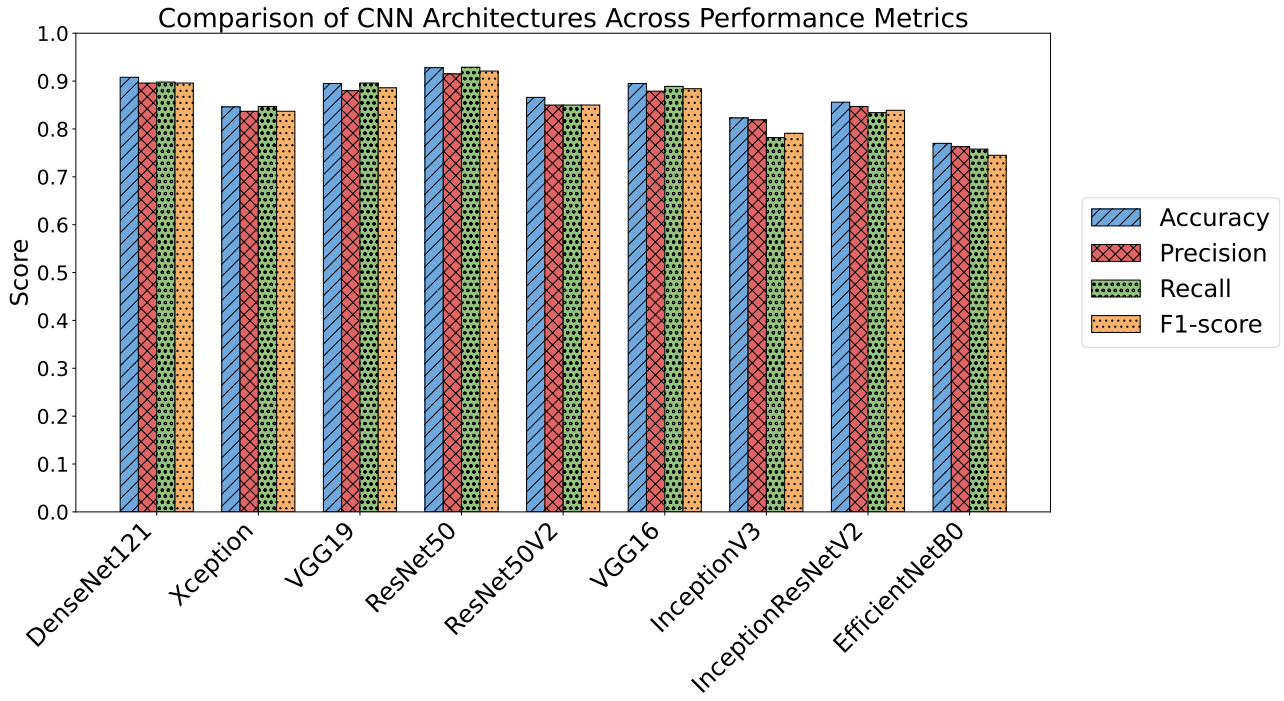

Figure 5.12: Comparison of final test set accuracy, precision, recall, and F1-score for optimised CNN architectures.

5.4 SNR Propagation

This chapter aims to examine the evolution of the SNR as information disseminates through intricate deep learning networks built on MRI brain tumour data. The objective was to delineate the transformation of signal and noise components across layers and to examine how these transformations affected the development of discriminative representations. The study commenced with the differentiation of signal and background noise components utilising the segmentation masks included in the dataset. Tumour regions are delineated to calculate average signal strength, whereas background blocks are obtained from non-tumor regions to signify noise. This facilitated a meticulous, incremental evaluation of how each network processed pertinent information in relation to extraneous background fluctuation. The transmission of SNR is subsequently monitored solely via the convolutional layers of each model, measuring the evolution of these components as they are filtered, increased, or diminished by the learned kernels. To enhance this approach and assess the representational quality of intermediate feature maps, lightweight linear probe classifiers (logistic regression models [61]) trained on activation vectors obtained from each convolutional layer are used. The probes are assigned the responsibility of predicting the accurate class labels of the test images utilising solely the information accessible at a specified depth. The resultant probe accuracies provide an estimation of the linear separability of features, so indicating the extent to which discriminative structure manifested as the network deepened [48]. Through a concurrent analysis of SNR dynamics and probe efficacy, we establish a connection between the signal's physical attributes (its strength in relation to noise) and the semantic quality of the

acquired representations. To further assess the networks' resilience to input degradation, Gaussian noise is introduced into the input data, and the subsequent SNR progression is recorded at each convolutional layer. This controlled perturbation enables us to evaluate the impact of initial noise circumstances on each network's capacity to retain or inhibit pertinent information. It also offers insight into whether specific architectures are intrinsically more resilient to input corruption. The analysis is conducted in two phases. Initially, we set a baseline by analysing how each model handles the intrinsic background noise in the original MRI scans, devoid of any extra perturbations. This serves as a benchmark for comprehending the evolution of SNR and representational quality under standard conditions. Subsequently, we expand the research by incorporating Gaussian noise of differing strengths to examine the effects of escalating corruption on both SNR and probe accuracy. The primary text concentrates on the most informative scenario under a significant noise condition ($\sigma = 1.0$), whereas outcomes for reduced noise levels are regarded as transitional stages between pristine and severely compromised inputs. This study systematically quantifies SNR flow under both natural and perturbed conditions, correlating these measurements with layer-wise classification performance, thereby elucidating the processing of medical image data by complex deep learning architectures, the impact of noise on internal representations, and the influence of architectural choices on robustness to signal degradation.

5.4.1 Impact of Background Noise on SNR and Accuracy

Before introducing artificial noise, it is useful to first examine how the networks perform with only the background noise inherently present in the raw images. This provides a baseline for how SNR and classification accuracy evolve as the signal propagates through the layers without additional perturbations. Analysing this scenario also reveals how each architecture naturally mitigates noise in the data before being subjected to more challenging conditions.

In Figure 5.13, VGG16 exhibits a generally stable SNR profile across the convolutional hierarchy, with moderate fluctuations but no severe degradation in deeper layers. Both training and test accuracies increase steadily through the initial and intermediate layers, reaching near saturation in the deeper convolutional blocks. The close alignment between training and test accuracy curves suggests limited overfitting, indicating that the learned representations generalise well to unseen data. The parallel rise in accuracy and SNR across early layers further implies that the network effectively amplifies signal dominant features while suppressing background variation. However, a slight divergence appears towards the final layers, where SNR declines marginally while accuracy remains consistently high. This decoupling may reflect that deeper layers capture increasingly semantic patterns that are less directly correlated with pixel level signal intensity. The correspondence between SNR and accuracy in VGG16, particularly in the earlier convolutional blocks, underscores the model's relatively linear feature hierarchy. Because VGG16 lacks residual or dense skip connections, its discriminative capacity remains closely linked to the preservation of raw signal clarity, offering a clear example of how simpler feed-forward architectures maintain a more direct relationship between SNR evolution and classification performance.

In Figure 5.14, VGG19 displays a pattern broadly consistent with that of VGG16, though

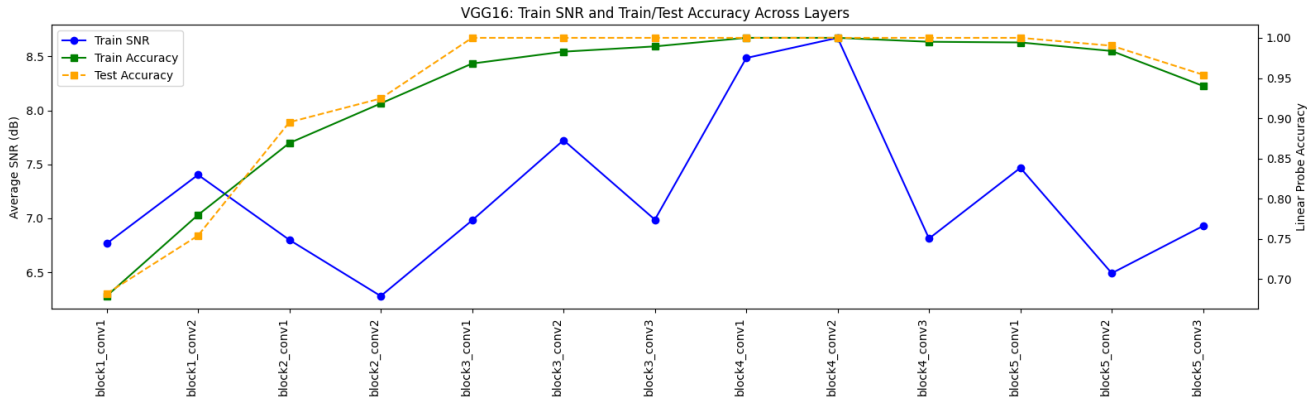

Figure 5.13: Layer-wise relationship between SNR and linear probe accuracy for VGG16 under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

with more pronounced fluctuations in SNR across layers and a smoother, more stable accuracy trajectory. The SNR rises modestly in the initial convolutional layers, dips through the second block, and then recovers to reach its peak around the mid-depth of the network before tapering off slightly toward the end. This transient degradation followed by recovery reflects how the deeper hierarchy of VGG19 temporarily disperses signal energy across a greater number of filters before consolidating it into more structured representations in later stages. Both training and test accuracies increase rapidly across the early layers and plateau at near perfect levels beyond the midpoint, with only minimal separation between the two curves. This close alignment indicates strong generalisation and little evidence of overfitting, suggesting that the network's additional convolutional depth allows it to extract robust, transferable features rather than merely memorising the training data. The partial decoupling between accuracy and SNR in the deeper layers, where accuracy remains high despite SNR oscillations, implies that representational quality in VGG19 is governed less by raw signal fidelity and more by abstracted, high-level feature integration. Compared to VGG16, VGG19 exhibits greater resilience to internal noise variation and a more stable convergence of accuracy across both training and test data. These trends highlight that increased depth enhances the network's ability to stabilise discriminative representations even when lower level SNR dynamics fluctuate, reinforcing the role of hierarchical feature refinement in mitigating the effects of input noise on downstream classification performance.

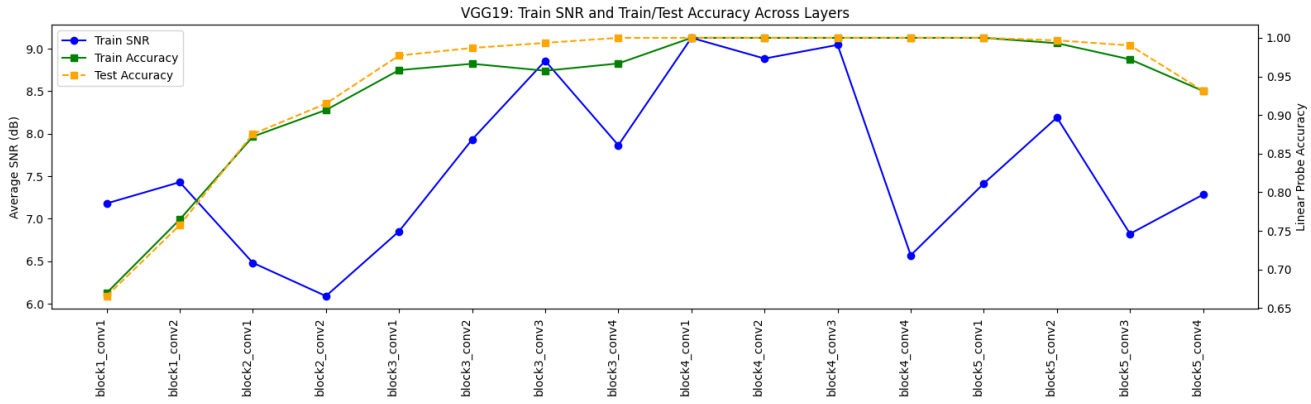

Figure 5.14: Layer-wise relationship between SNR and linear probe accuracy for VGG19 under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.15, ResNet50 exhibits a different pattern compared to the sequential VGG architectures. The SNR peaks sharply in the earliest convolutional layers before undergoing a gradual decline as depth increases, with pronounced oscillations corresponding to residual block transitions. In contrast, both training and test accuracies rise steadily through the middle layers, stabilising at relatively high values toward the deeper stages. The close alignment between training and test accuracy across most of the network indicates strong generalisation and minimal overfitting, despite the increasing representational complexity introduced by the residual connections. The inverse relationship between SNR and accuracy, in which discriminative performance improves despite a decline in raw signal clarity, illustrates ResNet50's ability to transform rather than simply maintain the input signal. Residual skip connections let the network change and improve feature representations across depth, making sure that important information for the job stays while noise and unnecessary details are slowly removed. This feature lets ResNet50 keep its high classification accuracy even when the SNR between layers goes down. This shows how strong and efficient its architecture is at learning hierarchical abstractions. The disconnection between SNR and accuracy demonstrates the efficacy of residual learning in maintaining discriminative capacity without requiring elevated signal intensity in deeper convolutional layers.

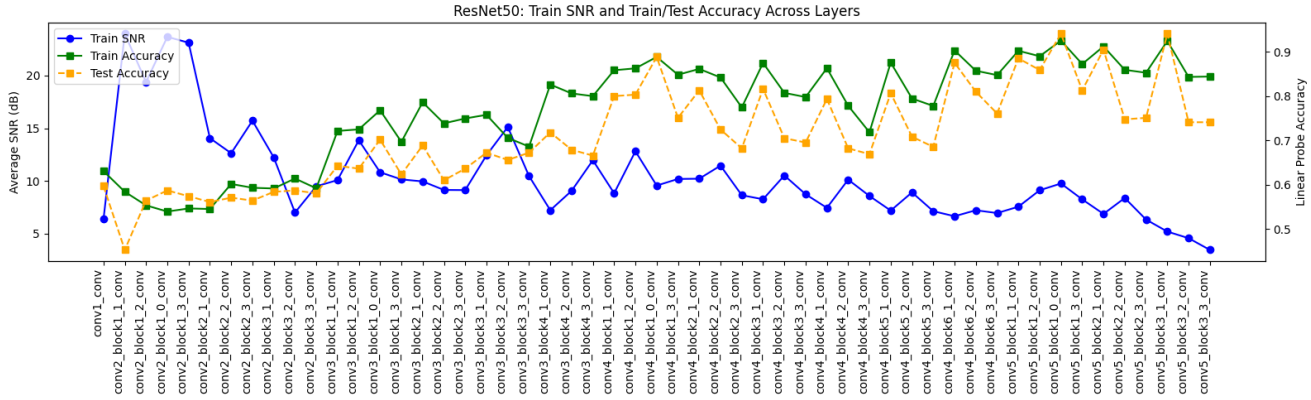

Figure 5.15: Layer-wise relationship between SNR and linear probe accuracy for ResNet50 under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.16, ResNet50V2 exhibits a highly dynamic interaction between SNR and accuracy, characteristic of its pre-activation residual architecture. Unlike the smoother progression observed in VGG networks, both SNR and accuracy fluctuate considerably across convolutional layers, reflecting the repeated normalisation and reweighting of activations inherent to the design. SNR peaks early in the network and then oscillates through successive residual blocks, suggesting that information is cyclically redistributed rather than monotonically attenuated. Both training and test accuracies follow a similar oscillatory pattern, with alternating rises and dips that roughly mirror the residual block boundaries. The close tracking between training and test accuracy indicates consistent generalisation, with little evidence of overfitting despite the architectural depth. Towards the deeper layers, accuracy converges to a high, stable level even as SNR continues to vary, demonstrating that representational quality is increasingly governed by abstract feature organisation rather than raw signal intensity. Compared to the original ResNet50, the V2 variant shows more local instability in both SNR and accuracy but ultimately achieves comparable or stronger discriminative performance. This behaviour highlights the stabilising influence of the pre-activation formulation which by normalising inputs before each convolution, ResNet50V2 preserves gradient flow and enables deeper layers to focus on task relevant transformations, sustaining high classification accuracy even when the underlying signal quality fluctuates.

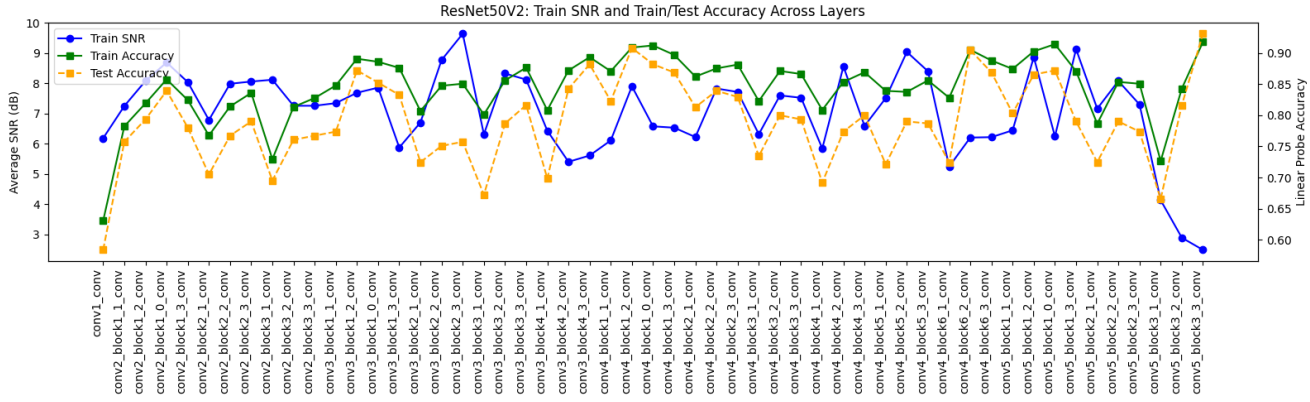

Figure 5.16: Layer-wise relationship between SNR and linear probe accuracy for ResNet50V2 under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.17, DenseNet121 exhibits highly variable SNR across its convolutional layers, with no consistent pattern of increase or decline at specific architectural transitions. Both training and test accuracies fluctuate substantially, reflecting the network's dense inter-layer connectivity in which each layer receives feature maps from all preceding ones. Despite this variability, the overall alignment between training and test curves indicates stable generalisation rather than overfitting, although local discrepancies, where training accuracy briefly exceeds test accuracy, suggest minor sensitivity to depth induced feature redundancy. The layers labelled *pool∗_conv* are particularly notable, showing distinct jumps in accuracy but only modest or inconsistent changes in SNR. Although named as pooling layers, these correspond in the Keras implementation of DenseNet to 1×1 convolutional projection transitions between dense blocks, used to compress feature maps and match dimensions before downsampling. The sharp rise in accuracy at the first *pool2_conv* layer likely reflects the initial major reduction in feature map size, where low-level noise is suppressed while structured signal content is preserved. Subsequent transition layers continue to improve accuracy without corresponding SNR increases, suggesting that the signal to noise balance stabilises after initial compression. DenseNet121 demonstrates that discriminative power in densely connected architectures emerges from feature reuse and aggregation rather than from consistent increases in raw signal clarity. The decoupling between SNR and accuracy, where performance peaks occur independently of local signal strength, illustrates how densely connected feature fusion enables the model to extract robust, distributed representations even in the presence of fluctuating layer-wise SNR.

In Figure 5.18, EfficientNetB0 exhibits a distinct SNR vs. accuracy relationship, reflecting its compound scaling strategy and the use of inverted residual blocks with squeeze and excitation mechanisms. The SNR is initially high in the stem and early convolutional layers but drops sharply as depth increases, eventually stabilising at much lower values. This steep decline indicates that the network performs substantial compression and refinement of information early in the processing hierarchy, discarding redundant or noisy signal components while retaining task relevant structure. Both training and test accuracies increase rapidly

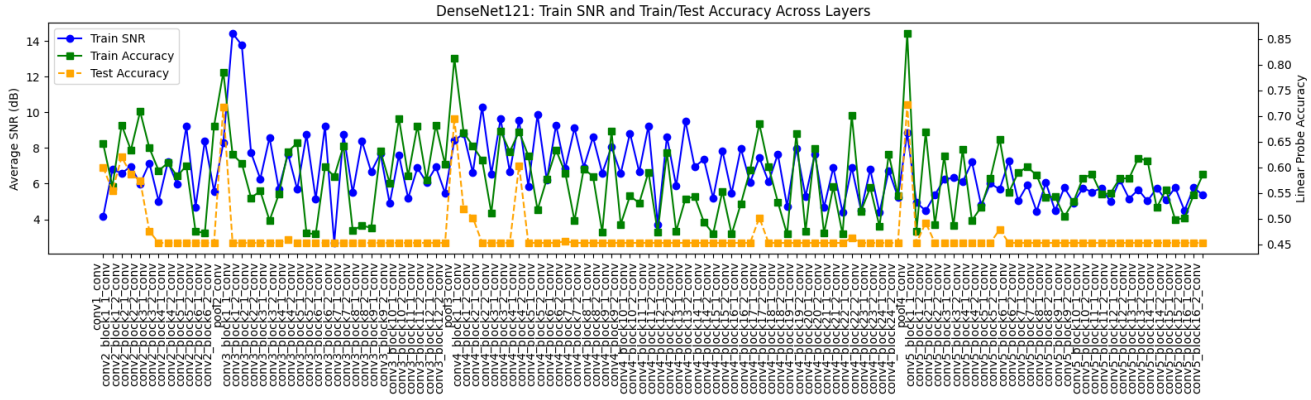

Figure 5.17: Layer-wise relationship between SNR and linear probe accuracy for DenseNet121 under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

through the early layers and then fluctuate around a high plateau in the deeper stages. The close alignment between the two accuracy curves suggests strong generalisation and minimal overfitting, despite the aggressive internal compression. The oscillatory behaviour visible in both metrics corresponds to the repeated expansion projection cycles within EfficientNet's inverted bottleneck blocks, where alternating dimensionality changes and channel recalibrations reorganise the feature space. The sustained high accuracy despite the strong SNR reduction illustrates that EfficientNetB0 achieves discriminative robustness through efficient feature reparameterisation rather than raw signal preservation. By learning to amplify salient activations via squeeze and excitation modules and suppress less informative components, the network maintains high representational quality even in low SNR conditions. These results highlight how EfficientNetB0's architectural efficiency enables strong classification performance through adaptive feature refinement rather than reliance on conventional signal strength.

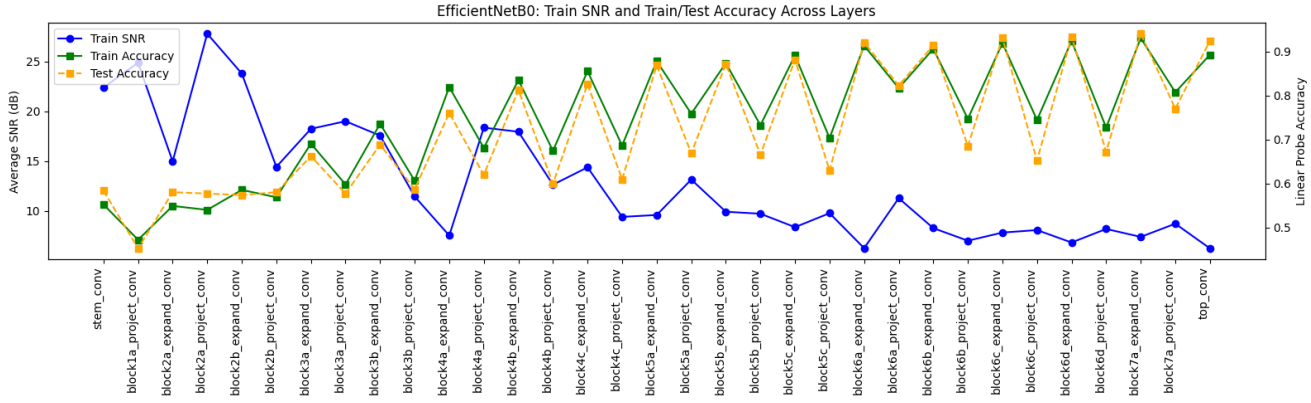

Figure 5.18: Layer-wise relationship between SNR and linear probe accuracy for EfficientNetB0 under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.19, Xception exhibits a distinct divergence between SNR and accuracy that reflects the architectural impact of depthwise separable convolutions. The SNR increases across the initial layers, peaking at the second convolution, before gradually declining towards deeper stages. This pattern suggests that the network initially amplifies salient spatial signal components but progressively shifts towards forming more abstract, high-level representations, even at the expense of raw signal clarity. Both training and test accuracies follow a consistent upward trajectory, with only a minor dip in the mid-network region, after which performance rapidly recovers and approaches near perfect levels. The close correspondence between training and test curves indicates strong generalisation, suggesting that Xception maintains stable representational learning without evident overfitting. The observed decoupling between SNR and accuracy highlights the efficiency of depthwise separable convolutions in independently filtering spatial and channel-wise information. By doing so, the network can sustain discriminative capacity even as traditional measures of signal strength decline. Xception demonstrates that architectural efficiency, achieved through separable convolutions and reduced parameter coupling, enables the extraction of highly informative features from progressively abstract representations. This reinforces the idea that modern architectures can maintain robust classification performance through structured feature decomposition rather than reliance on raw signal magnitude.

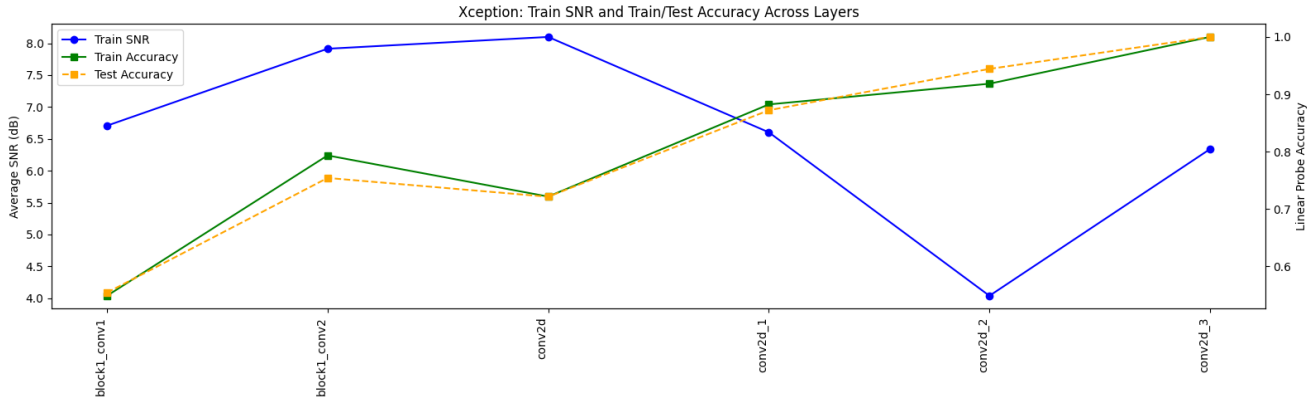

Figure 5.19: Layer-wise relationship between SNR and linear probe accuracy for Xception under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.20, InceptionV3 presents a highly irregular relationship between SNR and accuracy, reflecting the architectural complexity introduced by its multi-branch convolutional modules. The SNR fluctuates substantially across layers, exhibiting recurring peaks and troughs rather than a smooth progression, which indicates that different branches within the inception modules alternately emphasise and suppress various aspects of the input signal. Both training and test accuracies follow a similarly oscillatory pattern, with frequent local increases that often, but not always, coincide with SNR peaks. Despite these fluctuations, the overall alignment between the training and test curves suggests consistent generalisation and limited overfitting. Accuracy remains strong across much of the network, even in regions where SNR dips, underscoring the capacity of InceptionV3 to maintain task relevant feature

quality despite high variability in signal clarity. This partial decoupling between SNR and accuracy highlights that discriminative performance in InceptionV3 arises primarily from the integration of complementary multi-scale features rather than from uniform signal strength. These results illustrate how the network's modular design, combining parallel convolutions of different kernel sizes, enables robust feature aggregation. By capturing spatial patterns at multiple receptive field scales, InceptionV3 sustains high classification accuracy even when conventional measures of signal quality fluctuate widely across layers.

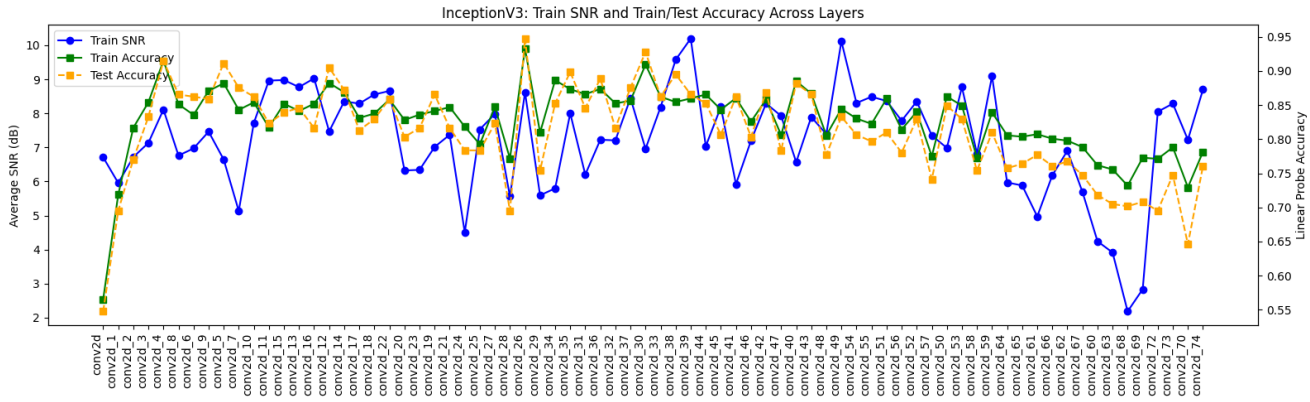

Figure 5.20: Layer-wise relationship between SNR and linear probe accuracy for InceptionV3 under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.21, InceptionResNetV2 displays one of the most intricate SNR vs. accuracy dynamics among the architectures analysed, reflecting its hybrid design that combines inception style multi-branch processing with residual connections. Both SNR and accuracy fluctuate substantially across layers, with frequent peaks and troughs and no clear global trend. SNR remains relatively high overall but exhibits sharp local oscillations, indicating continual adjustments in how the network balances noise suppression and feature amplification. Training and test accuracies follow a similarly variable pattern, with repeated rises and dips across residual blocks. The two curves track one another closely, suggesting that despite the volatility, generalisation remains stable and that the architecture avoids significant overfitting. Notably, local maxima in accuracy often correspond to, but do not always coincide with, peaks in SNR. This highlights that discriminative performance is influenced by complex internal feature interactions rather than direct dependence on raw signal clarity. This interplay reflects the dual influence of inception modules, which extract multi-scale information with differing signal characteristics, and residual pathways, which stabilise feature propagation and refine representations. Overall, InceptionResNetV2 demonstrates how combining multi-branch feature extraction with residual learning enables the network to maintain strong classification performance even under highly variable SNR conditions, underscoring the adaptability of hybrid deep architectures in balancing representational diversity with discriminative robustness.

These results demonstrate that the relationship between SNR and linear probe accuracy is strongly influenced by network architecture. Simpler feed-forward models such as VGG16

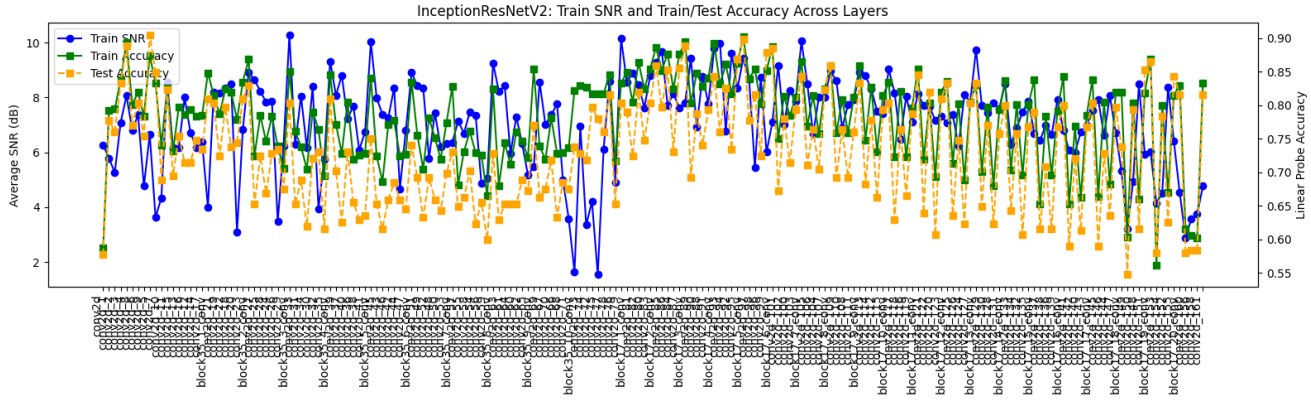

Figure 5.21: Layer-wise relationship between SNR and linear probe accuracy for InceptionResNetV2 under background noise only, illustrating how signal clarity and discriminative performance evolve across depth.

and VGG19 exhibit a clear correspondence between signal clarity and discriminative performance, particularly in the early and intermediate layers. In contrast, residual architectures such as ResNet50 and ResNet50V2 achieve progressively higher accuracy with depth despite declining or fluctuating SNR, while DenseNet121 shows accuracy gains primarily at major architectural transition points. More advanced models, including EfficientNetB0, Xception, and the Inception variants, highlight that high classification accuracy can be maintained even when conventional measures of signal quality are relatively low. This resilience arises from architectural mechanisms such as feature reweighting, multi-branch processing, and representational abstraction, which prioritise the extraction of task relevant information over the preservation of raw signal strength. Importantly, the noise examined here stems from the natural background of the MRI images rather than from artificially introduced perturbations. Collectively, these findings illustrate that discriminative capacity in modern CNNs depends not solely on signal preservation but on the network's ability to transform, refine, and integrate features across layers. Architectural innovations, including residual connections, dense connectivity, depthwise separability, and inception modules, enable networks to suppress irrelevant information and amplify salient structure, thereby sustaining high performance even in inherently noisy conditions. In the following Subsection, this analysis is extended to examine how the networks respond when Gaussian noise is artificially added to the inputs, providing deeper insight into their robustness under controlled perturbations.

5.4.2 Impact of Gaussian Noise on SNR and Accuracy

After establishing how the networks behave under natural background noise, the next step is to investigate their robustness when additional perturbations are introduced. To do this, Gaussian noise is injected into the inputs at $\sigma = 1.0$. The focus of this subsection is on the $\sigma = 1.0$ case because it reveals the most pronounced effects, while lower levels of σ can be regarded as intermediate stages between the clean and heavily corrupted conditions.

In Figure 5.22, VGG16 exhibits a pronounced decline in SNR across its convolutional layers

when Gaussian noise with $\sigma = 1.0$ is added to the input images. Although raw signal clarity deteriorates substantially, linear probe accuracy continues to rise steadily with depth and remains near perfect in the final layers. This behaviour parallels that observed under clean conditions, but the divergence between SNR and accuracy becomes more pronounced under noise, indicating that discriminative features are maintained even as the physical signal degrades. The results suggest that VGG16's straight forward feed-forward structure can progressively suppress input noise and recover class relevant information, with deeper layers effectively compensating for early-stage corruption and preserving task performance despite significant degradation in input quality.

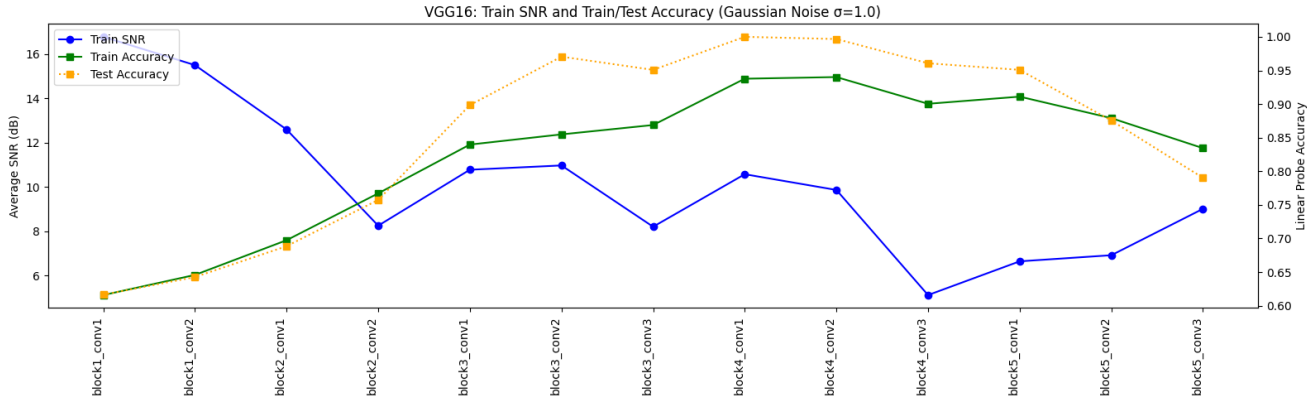

Figure 5.22: Layer-wise relationship between SNR and linear probe accuracy for VGG16 with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.23, VGG19 demonstrates strong robustness to additive Gaussian noise, maintaining a performance profile closely resembling that observed under clean conditions. SNR decreases markedly across the network, particularly in the mid to late convolutional blocks, reflecting the impact of severe input corruption. Despite this decline in raw signal clarity, linear probe accuracy rises steadily with depth and approaches near perfect levels in the final layers. The widening separation between SNR and accuracy compared to the background noise scenario indicates that deeper layers effectively compensate for early-stage degradation by filtering and reconstructing task relevant structure. This behaviour highlights the advantage of VGG19's increased depth where the additional convolutional stages allow for more extensive abstraction and noise suppression, enabling the network to sustain strong discriminative performance even under heavy perturbation. Compared to VGG16, VGG19 exhibits marginally stronger resilience, suggesting that deeper sequential hierarchies confer an incremental yet meaningful improvement in robustness to input corruption.

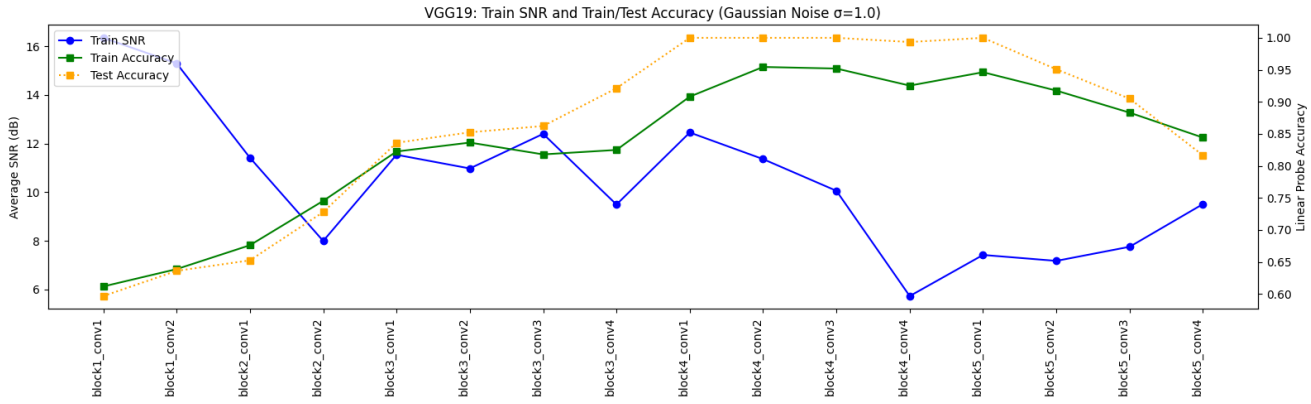

Figure 5.23: Layer-wise relationship between SNR and linear probe accuracy for VGG19 with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.24, ResNet50 displays reduced stability under strong Gaussian noise. SNR values remain relatively high in the early layers but decline steadily with increasing depth, reflecting progressive signal degradation. Linear probe accuracy, initially suppressed, gradually recovers in deeper layers but exhibits more erratic fluctuations than under clean conditions. This behaviour indicates that residual connections continue to support the emergence of discriminative features. However, under severe perturbation, the skip pathways also propagate noise alongside useful activations, thereby reducing the consistency of feature refinement. As a result, the architecture's robustness is only partial while deeper layers can still reconstruct task relevant structure, their efficiency in isolating informative representations diminishes compared to simpler feed-forward networks such as VGG16.

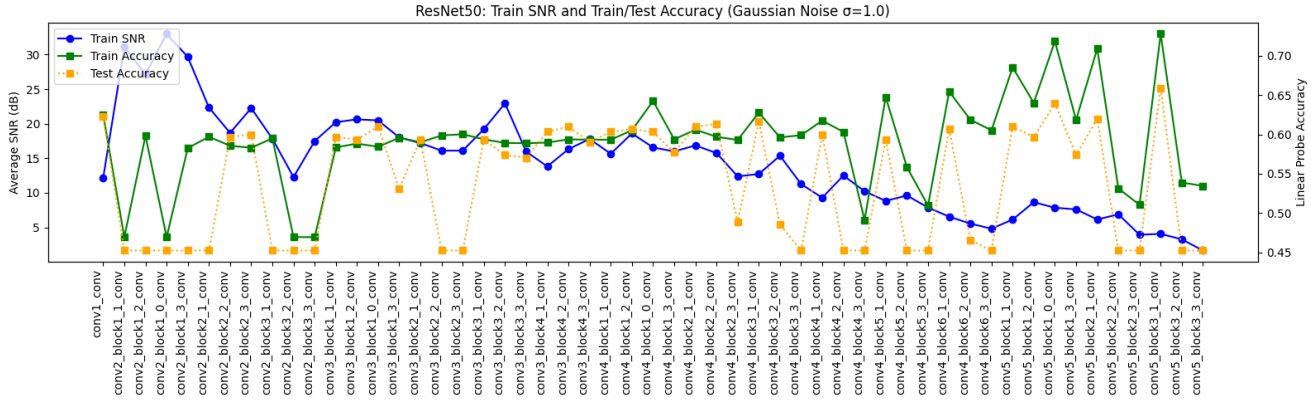

Figure 5.24: Layer-wise relationship between SNR and linear probe accuracy for ResNet50 with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.25, ResNet50V2 exhibits pronounced sensitivity to additive Gaussian noise, with both SNR and linear probe accuracy showing strong oscillations across layers. SNR remains relatively high in the early-stages but fluctuates substantially thereafter, suggesting that

noise is repeatedly reweighted and redistributed as it propagates through the network. Linear probe accuracy is consistently lower than under clean conditions and features frequent peaks and troughs, with only partial recovery in the deeper layers. This behaviour reflects the dual effect of the pre-activation residual structure that while normalisation and skip connections facilitate the retention of useful features, they also enable corrupted information to persist through the network, limiting its ability to stabilise internal representations under heavy perturbation. Consequently, ResNet50V2 retains some capacity to extract discriminative features but does so less efficiently and less consistently than under background noise alone. Compared to the original ResNet50, the V2 variant demonstrates slightly stronger late-stage recovery of accuracy, implying that pre-activation supports more stable gradient propagation under noise stress though it does not fully mitigate the instability introduced by additive corruption.

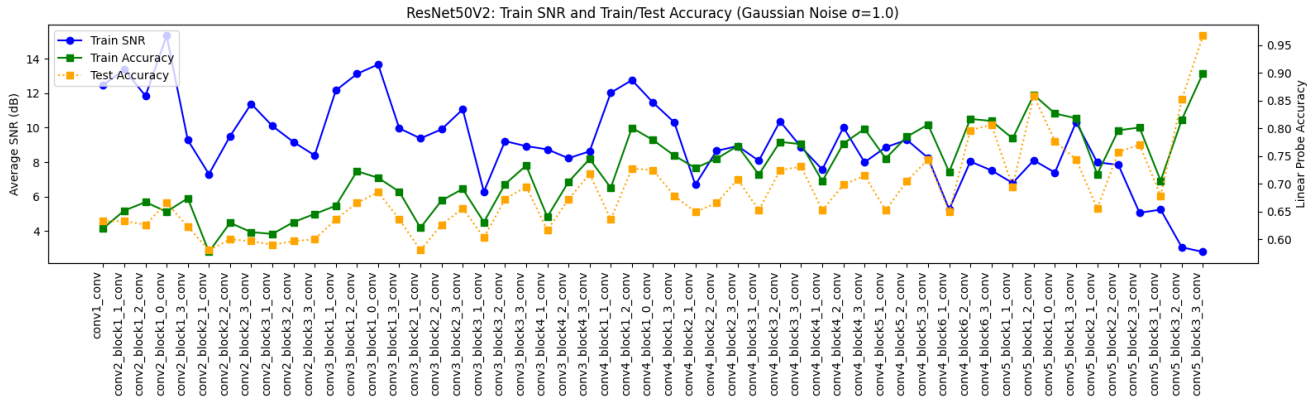

Figure 5.25: Layer-wise relationship between SNR and linear probe accuracy for ResNet50V2 with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.26, DenseNet121 appears to be the most negatively impacted by Gaussian noise. Linear probe accuracy remains consistently low across layers, with only modest spikes at the projection transitions, mirroring the pattern observed under clean conditions. However, unlike VGG16 or ResNet50, these local increases are insufficient to counteract the pervasive effect of noise on the network's internal representations. The dense connectivity that ordinarily facilitates efficient feature reuse instead amplifies the propagation of noise, entangling corrupted activations across layers and reducing the network's capacity to isolate useful signal components. As a result, both SNR and accuracy plateau at modest levels, revealing that DenseNet121's inter-layer coupling, while powerful for information flow in noise free settings, compromises its ability to suppress or recover from severe input perturbations.

In Figure 5.27, EfficientNetB0 exhibits a pronounced loss of robustness when Gaussian noise is introduced, with linear probe accuracy substantially lower than under clean conditions. SNR remains relatively high in the early layers and continues to fluctuate dynamically throughout the network, suggesting that the architecture still engages in substantial signal transformation despite heavy input corruption. However, accuracy oscillates sharply and remains consistently low, showing little recovery even in the later layers. This instability re-

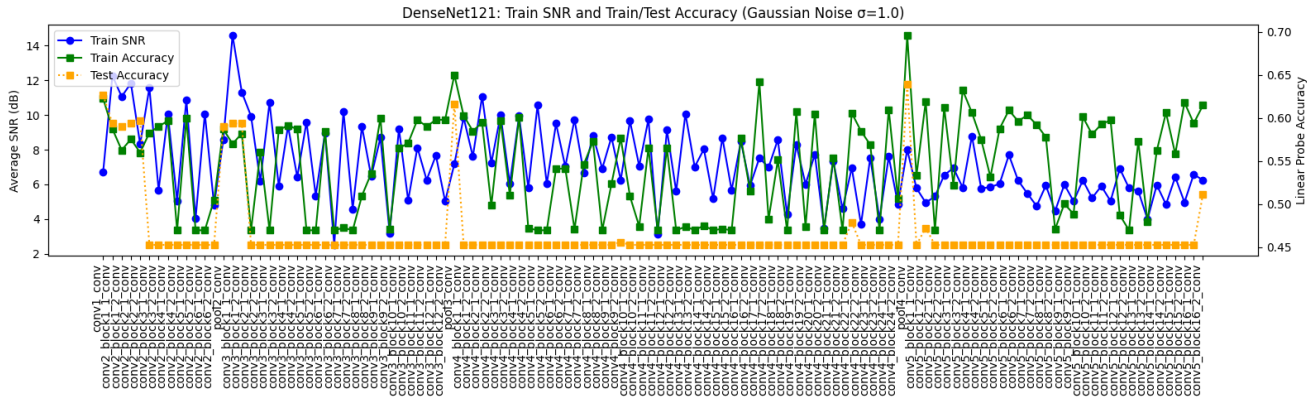

Figure 5.26: Layer-wise relationship between SNR and linear probe accuracy for DenseNet121 with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

flects the sensitivity of EfficientNet's compound scaling and inverted residual design to noisy inputs suggesting that while expansion projection cycles efficiently compress and reorganise features under normal conditions, they also tend to propagate and amplify noise when the input is heavily degraded. The squeeze and excitation mechanism may further exacerbate this effect by selectively reweighting noisy activations, thereby reducing the network's capacity to emphasise meaningful patterns. Consequently, EfficientNetB0 demonstrates limited noise tolerance, with its highly optimised but delicate transformation pipeline proving less resilient under severe perturbations.

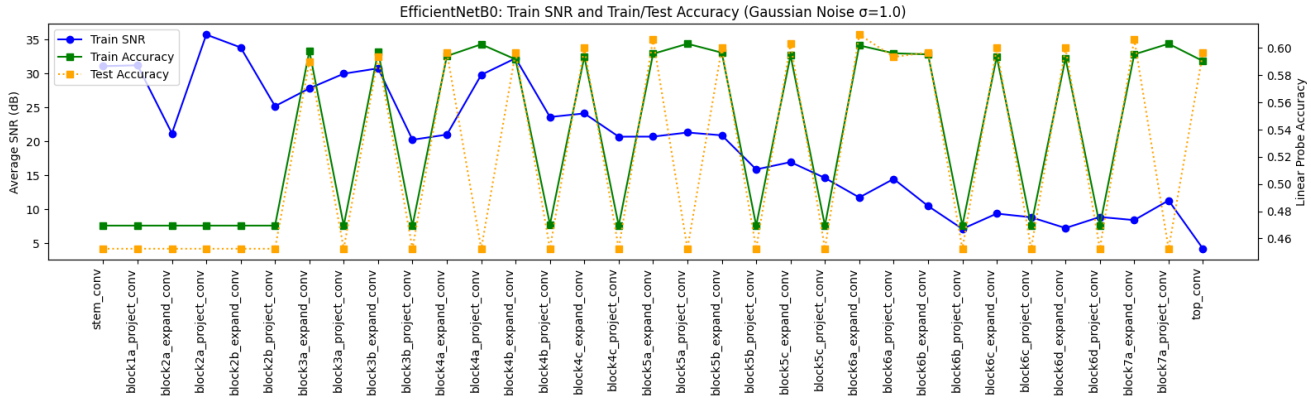

Figure 5.27: Layer-wise relationship between SNR and linear probe accuracy for EfficientNetB0 with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.28, Xception exhibits a pronounced divergence between SNR and accuracy under Gaussian noise, revealing both resilience and sensitivity within its depthwise separable convolutional design. SNR increases across the early layers, peaking around the third convolution, before dropping sharply in the middle of the network and stabilising at lower

levels. Despite this decline in raw signal clarity, linear probe accuracy continues to improve with depth, reaching near perfect levels in the final layers. This behaviour indicates that Xception retains its capacity to disentangle spatial and channel-wise information even under severe input corruption, enabling deeper layers to recover and refine task relevant features. However, the sharp mid-layer SNR decline suggests that depthwise operations remain susceptible to input perturbations, temporarily reducing the efficiency of signal to noise separation. Overall, Xception demonstrates strong robustness to additive noise, showing that its feature decomposition strategy continues to support effective discrimination even when intermediate representations are degraded.

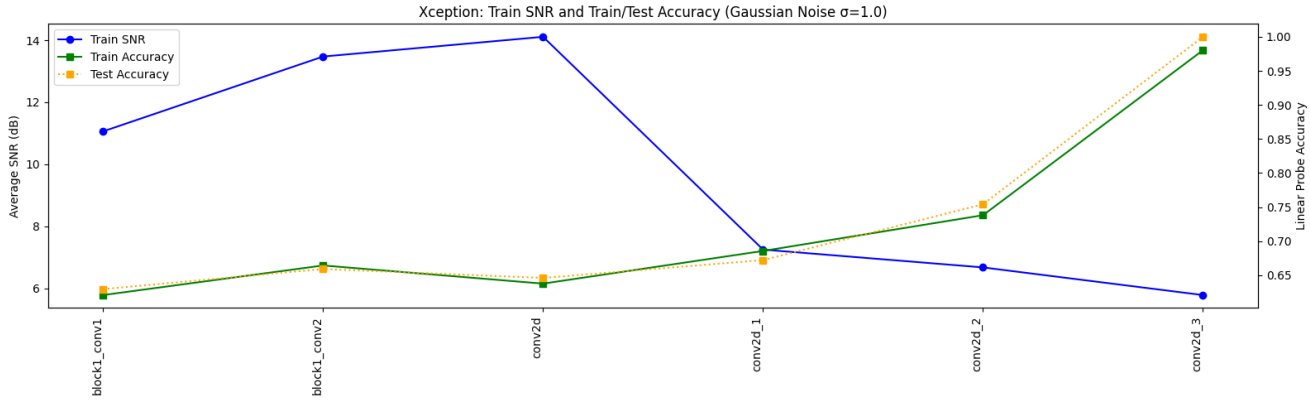

Figure 5.28: Layer-wise relationship between SNR and linear probe accuracy for Xception with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

In Figure 5.29, InceptionV3 exhibits significant deterioration in both SNR and linear probe accuracy in the presence of Gaussian noise, with both measures demonstrating notable variability between layers. The SNR fluctuates significantly, exhibiting recurrent peaks and troughs that indicate regular adjustments in the network's management of signal integrity and alteration in the presence of contamination. Accuracy exhibits a similarly erratic pattern, consistently remaining significantly lower than the levels recorded under background noise alone and showing only slight improvement with depth. This instability underscores the susceptibility of InceptionV3's multi-branch feature extraction mechanism to input perturbations, although the architecture proficiently captures complementary spatial features under optimal conditions, significant noise impairs the integration of these feature streams, diminishing its capacity to isolate task relevant information. Nevertheless, a degree of local congruence between SNR peaks and accuracy spikes remains, suggesting that specific branches may still extract valuable structure despite the degradation of the global signal. InceptionV3 exhibits moderate robustness, with its multi-scale processing advantage reduced when noise disrupts the spatial coherence of the input.

In Figure 5.30, InceptionResNetV2 exhibits a highly unstable relationship between SNR and linear probe accuracy under heavy Gaussian noise, with both metrics fluctuating markedly across layers. SNR oscillates throughout the network, indicating that noise is repeatedly amplified and attenuated as it propagates through the hybrid inception residual architec-

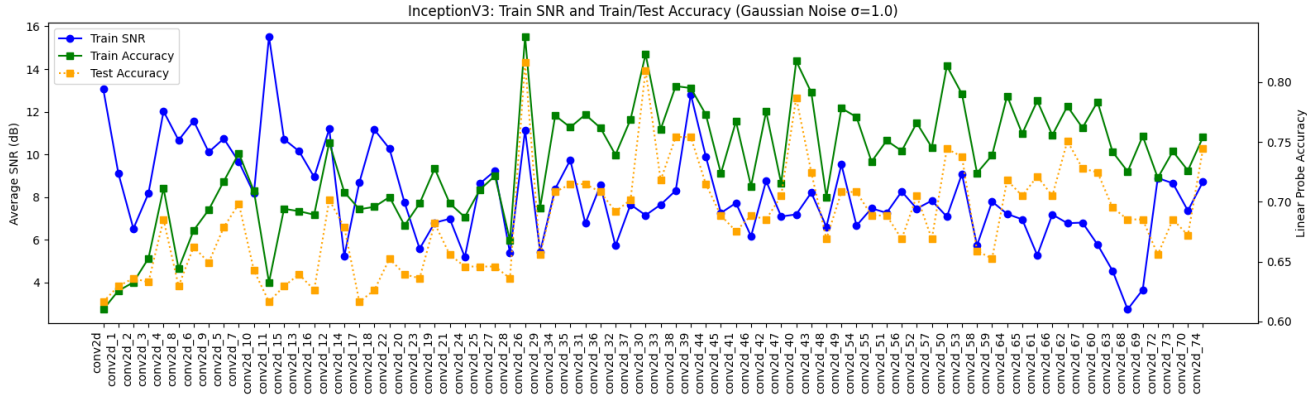

Figure 5.29: Layer-wise relationship between SNR and linear probe accuracy for InceptionV3 with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

ture. Accuracy follows a similarly erratic trajectory and remains well below the levels observed under background noise, with only partial recovery in deeper layers. The frequent decoupling between SNR peaks and accuracy spikes suggests that the model struggles to consistently translate clearer representations into discriminative performance. This instability likely stems from the competing effects of inception modules and residual pathways that suggest that while the former extract multi-scale features, the latter propagate corrupted activations, compounding the challenge of noise suppression. Despite these limitations, occasional local accuracy gains indicate that the network can still recover some useful features from noisy representations. Overall, however, InceptionResNetV2 demonstrates limited robustness, with its architectural complexity offering little resilience advantage under severe input corruption compared to simpler convolutional networks.

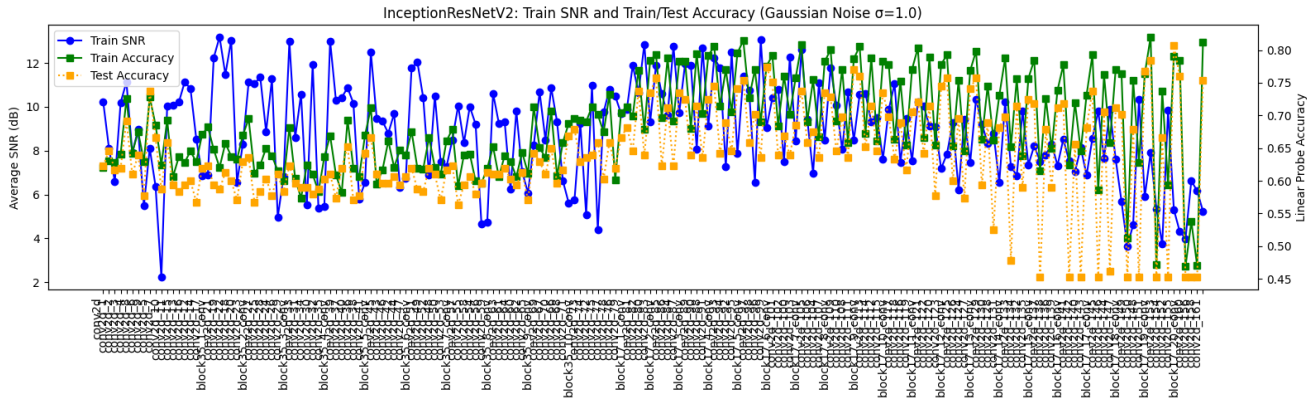

Figure 5.30: Layer-wise relationship between SNR and linear probe accuracy for InceptionResNetV2 with background and additive Gaussian noise ($\sigma = 1.0$), illustrating how signal clarity and discriminative performance evolve across depth.

Across architectures, a consistent pattern emerges. Gaussian noise injected at the input stage

exerts a lasting influence on early representations, with its effects either progressively attenuated or continually propagated depending on the network design. Architectures characterised by strong sequential filtering stages, such as VGG16 and VGG19, show clear evidence of progressive noise suppression, as deeper layers recover discriminative features despite early corruption. In contrast, networks that rely heavily on skip connections or feature reuse, such as ResNet50 and DenseNet121, tend to redistribute or retain noise across layers. While residual links support information flow under clean conditions, they also propagate corrupted activations, leading to reduced stability under strong perturbations. Multi-branch and attention based architectures, such as InceptionV3, InceptionResNetV2, and EfficientNetB0, display mixed behaviour, with certain pathways effectively filtering noise while others amplify it. This divergence underscores that robustness is governed less by depth or parameter count than by architectural mechanisms regulating information flow. Among these, Xception stands out as an exception because its depthwise separable convolutions continue to extract meaningful features even under heavy corruption, whereas DenseNet121 proves least robust, with dense connectivity spreading noise throughout the network. Collectively, these findings demonstrate that the addition of Gaussian noise fundamentally reshapes the relationship between SNR and accuracy across architectures, revealing that resilience depends on how effectively networks manage the trade-off between feature reuse and noise suppression.

5.5 Conclusion

This chapter extended the analysis of SNR propagation to several complex CNN architectures, establishing how architectural design influences the transformation of signal and noise through depth. By jointly examining SNR and linear probe accuracy under both natural background noise and artificially induced Gaussian perturbations, we characterised the relationship between signal clarity and discriminative performance. The results revealed that each network exhibits a distinctive SNR vs. accuracy trajectory shaped by its internal information flow mechanisms. Sequential models such as VGG16 and VGG19 maintained a relatively direct correspondence between SNR and accuracy, indicating that their simpler feed-forward structures progressively suppress noise while preserving discriminative features. In contrast, architectures employing shortcut or feature reuse mechanisms, such as ResNet50, ResNet50V2, and DenseNet121, achieved strong classification accuracy even as SNR declined, reflecting their ability to transform rather than merely preserve the input signal.

Under controlled Gaussian perturbations, these architectural differences became more pronounced. Residual connections, while beneficial for gradient flow under clean conditions, tended to propagate corrupted activations, leading to reduced stability as input noise increased. DenseNet121 proved most vulnerable, with its dense connectivity amplifying noise and constraining its ability to isolate task relevant representations. Conversely, Xception demonstrated exceptional resilience, maintaining high accuracy despite substantial SNR degradation, owing to its depthwise separable convolutions that enable efficient disentangling of spatial and channel-wise features. More complex hybrid and compound architectures, including EfficientNetB0 and InceptionResNetV2, showed inconsistent robustness which highlights that architectural sophistication or parameter count alone does not guarantee stabil-

ity under perturbation. These findings collectively indicate that robustness is governed by how effectively architectural mechanisms control feature aggregation and noise propagation, rather than by model scale or depth.

Taken together, the results demonstrate that SNR is a valuable diagnostic measure of representational dynamics but cannot be viewed as a standalone predictor of discriminative performance. High accuracy can persist even when SNR declines, provided the architecture supports effective transformation, filtering, and recombination of features. Robust performance therefore depends on the network's ability to suppress irrelevant variation while amplifying task-specific structure. Understanding these mechanisms not only clarifies why certain CNNs exhibit greater tolerance to input degradation but also provides a foundation for designing architectures tailored to noisy, real-world domains such as biomedical imaging.

Chapter 6

Conclusion

6.1 Summary of the Problem and Research Aim

This thesis set out to investigate how CNNs process signal and noise as information propagates through their layers, and how these dynamics influence the discriminative quality of intermediate feature representations. While CNNs have become the backbone of modern computer vision and medical imaging systems, their internal behaviour, particularly their sensitivity to noise and robustness to degraded inputs, remains poorly understood. This limitation is especially critical in clinical applications, where input data are frequently noisy, heterogeneous, or corrupted by acquisition artefacts, and where model reliability is essential. To address this gap, the research developed a framework for analysing CNN behaviour using SNR and accuracy. By isolating signal and background components in MRI brain tumour data, the study quantified how the relative strength of the signal evolves across network layers and how this relates to the linear separability of learned representations. The analysis was conducted across several state-of-the-art CNN architectures, both under natural background noise and controlled Gaussian perturbations, providing a comprehensive view of how architectural choices shape robustness and feature extraction dynamics.

6.2 Summary of Key Results

The analyses conducted throughout this thesis reveal several important insights into how CNNs process signal and noise and how architectural design fundamentally shapes these dynamics. One of the most striking findings is that the relationship between SNR and classification accuracy is highly architecture dependent. Simpler feed-forward models such as VGG16 and VGG19 displayed a relatively direct correspondence between signal clarity and discriminative performance, with deeper layers able to recover useful features even when early-stage representations were degraded. In contrast, architectures incorporating skip connections or dense feature aggregation, such as ResNet and DenseNet, achieved strong accuracy despite declining or fluctuating SNR, indicating that discriminative power depends less

on raw signal preservation and more on how features are transformed and refined through the network. This finding underscores the importance of viewing feature processing as a dynamic transformation rather than a linear function of input quality.

A second key observation is that noise robustness is not solely determined by network depth. Although deeper architectures extracted more abstract and semantically meaningful representations, depth alone did not guarantee resilience to noise. Residual networks leveraged skip connections to preserve gradient flow and retain discriminative features, but these same mechanisms also allowed noise to propagate, reducing stability under strong perturbations. DenseNet121 further illustrated this limitation, with its dense connectivity leading to the amplification and entanglement of noise, which limited the network's capacity to suppress irrelevant variation. This observation indicates that while depth facilitates representational richness, the specific pathways of information flow play a critical role in determining robustness.

The results also show that architectural complexity does not necessarily equate to higher noise tolerance. Models such as EfficientNetB0 and InceptionResNetV2, which incorporate advanced scaling strategies or hybridised feature extraction modules, did not consistently outperform simpler networks when exposed to high-levels of corruption. In some cases, their intricate feature pipelines even heightened sensitivity to input noise, indicating that highly optimised designs may come at the cost of robustness. Conversely, Xception stood out for its ability to recover strong discriminative performance despite significant intermediate degradation. This resilience suggests that efficient feature decomposition through depthwise separable convolutions can offer an important mechanism for mitigating noise even when raw signal quality is compromised.

Finally, the study demonstrates that SNR and linear separability provide complementary yet distinct perspectives on network behaviour. High SNR does not always coincide with strong classification performance, and conversely, low SNR layers can still support highly discriminative features. This divergence indicates that representational utility depends not only on the physical quality of the signal but also on the organisation of the learned feature space. The addition of Gaussian noise made it even clearer that different architectures adapt representations in different ways. Although VGG models and Xception performed adequately despite corruption, DenseNet and EfficientNet struggled to recover valuable features. These results show that robustness is not only about the size, depth, or capacity of a model. It is also about how the architecture, information flow, and feature translation algorithms work together.

6.3 Limitations of the Study

While the findings provide valuable insight into the SNR dynamics in CNNs, several limitations must be acknowledged. The analysis was based on a single medical imaging dataset. Extending the experiments to other domains, such as real-world scenery images or multimodal data, would help assess the generality of the observed patterns. The exclusive use of linear probes provides a restricted view of representational quality. While they quantify

linear separability, they do not capture more complex, non-linear structure in the feature space. Incorporating broader representational analysis would yield deeper insights. Finally, the experiments focused on additive Gaussian noise. Real-world corruption such as motion blur, sensor artefacts, and systematic acquisition distortions may interact differently with network dynamics. Additionally, computational limits prevented exploration of larger models or extended hyperparameter tuning.

6.4 Future Work

This thesis elucidates the interplay between SNR, feature representation, and classification performance in CNNs, and it also motivates several directions for future work. A key direction is to expand the study to a broader variety of noise types that better reflect real imaging conditions. These include structured noise patterns, scanner specific artefacts, motion induced distortions, or adversarially generated perturbations. Evaluating these forms of corruption would help determine the conditions under which different architectures maintain or lose robustness. Another promising area involves architectural and training modifications explicitly informed by the observed SNR dynamics. In particular, selective regulation of skip connections in residual and densely connected architectures could be explored, for example through learnable gating mechanisms that suppress direct information pathways in layers where SNR degradation is detected. Similarly, channel-wise attenuation or weighting of low-SNR feature maps prior to dense aggregation may help limit noise entanglement while preserving the benefits of feature reuse. The robustness exhibited by depthwise separable architectures further suggests that targeted use of separable convolutions in noise-sensitive layers could improve resilience without increasing model complexity.

In addition, SNR analysis could be incorporated directly into training objectives. Auxiliary SNR-based regularisation terms may be introduced to discourage excessive noise amplification between successive layers, encouraging stable or controlled SNR progression during optimisation. Alternatively, layer-wise SNR estimates could be used to adapt regularisation strength or loss weighting dynamically during training, aligning optimisation pressure with internal representational quality rather than relying solely on output-level performance. Studying the temporal evolution of SNR throughout training, rather than only at convergence, may further reveal how networks learn to prioritise salient features over noise, informing optimisation strategies that promote early development of robust representations. Finally, extending the analysis beyond medical imaging to real-world scenery images, remote sensing data, or industrial sensor measurements would help determine whether the structural relationships between SNR and accuracy identified here are universal or domain-specific. Combining these investigations with interpretability tools such as attribution maps or representational similarity metrics may provide deeper insight into how architectures manage noise and how these mechanisms can be improved.

6.5 Concluding Remarks

This thesis demonstrates that understanding CNN behaviour requires going beyond conventional performance metrics to examine how signal and noise evolve internally. By analysing SNR dynamics and linear separability across several architectures and noise conditions, this research provides new perspectives on how CNNs extract, transform, and preserve meaningful information. These findings not only deepen our theoretical understanding of robustness in deep networks but also lay a foundation for designing future architectures better suited to real-world, noisy environments.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS 2012)*, pages 1097–1105, 2012.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[5] Geert et al. Litjens. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88, 2017.

[6] Andre Esteva et al. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24–29, 2019.

[7] Alexander S. Lundervold and Arvid Lundervold. An overview of deep learning in medical imaging focusing on mri. *Zeitschrift für Medizinische Physik*, 29(2):102–127, 2019.

[8] James Reed et al. Evaluation of noise suppression methods in medical imaging. *Journal of Digital Imaging*, 29(3):341–354, 2016.

[9] Simon Haykin. *Neural Networks and Learning Machines*. Pearson, 3rd edition, 2013.

[10] John G Proakis and Dimitris G Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice Hall, 4th edition, 2006.

[11] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased toward texture; increasing shape bias improves accuracy and robustness. In *Proceedings of the International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA, 2019.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE, 2009.

[13] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*, pages 125–136, 2019.

[14] Bolei Zhou, Yu Sun, David Bau, and Antonio Torralba. Learning deep features for discriminative localization under noise. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3215–3229, 2021.

[15] Zhiqiang Wang et al. Noise in medical imaging: Effects, models, and mitigation. *Medical Physics*, 47(6):e85–e98, 2020.

[16] Christian Szegedy et al. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[17] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[19] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1251–1258, 2017.

[20] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 6105–6114, 2019.

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.

[22] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[24] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*, pages 4278–4284, 2017.

[25] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 04(12):310–316, May 2020.

[26] Chigozie E. Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. In *Proceedings of the 2nd International Conference on Computational Sciences and Technology (INCCST)*, pages 124–133, Jamshoro, Pakistan, 2021.

[27] Eric R. Kandel, James H. Schwartz, Thomas M. Jessell, Steven A. Siegelbaum, and A. James Hudspeth. *Principles of Neural Science*. McGraw-Hill Education, 5th edition, 2013. Comprehensive reference on the structure and function of neurons and neural systems.

[28] Oluwatobi Ariyo and Farhana Akter. Anatomy and physiology of the neuron. *Neuroscience for Neurosurgeons*, page 72, 2023.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[31] Bin Ding, Huimin Qian, and Jun Zhou. Activation functions and their characteristics in deep neural networks. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 1836–1841, 2018.

[32] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *Schedae Informaticae*, 25:49–59, 2016.

[33] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[34] Murat H Sazlı. A brief review of feed-forward neural networks. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, 50(01), 2006.

[35] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of convolutional neural network (cnn). *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[36] Jianxin Wu. Introduction to convolutional neural networks. Technical report, National Key Lab for Novel Software Technology, Nanjing University, China, 2017. Technical Report, not peer-reviewed.

[37] Pierre Baldi and Peter J. Sadowski. Understanding dropout. In C. J. C. Burges, Léon Bottou, Max Welling, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26 (NeurIPS 2013)*, pages 2814–2822. Curran Associates, Inc., 2013.

[38] Will Nash, Tom Drummond, and Nick Birbilis. A review of deep learning in the study of materials degradation. *npj Materials Degradation*, 2, 12 2018.

[39] Yufeng Zheng, Clifford Yang, and Aleksey Merkulov. Breast cancer screening using convolutional neural network and follow-up digital mammography. page 4, 05 2018.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. volume 9908, pages 630–645, 10 2016.

[41] Noha Radwan. *Leveraging Sparse and Dense Features for Reliable State Estimation in Urban Environments*. PhD thesis, 06 2019.

[42] Ana Nogueira, Hugo Oliveira, José Machado, and Joao Tavares. Transformers for urban sound classification—a comprehensive performance evaluation. *Sensors*, 22:8874, 11 2022.

[43] Cheng Peng, Yikun Liu, Xinpan Yuan, and Qing Chen. Research of image recognition method based on enhanced inception-resnet-v2. *Multimedia Tools and Applications*, 81:1–21, 05 2022.

[44] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[45] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *International Conference on Learning Representations (ICLR) – Workshop Track*, 2016.

[46] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR)*, 2017.

[47] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3519–3529, 2019.

[48] Yuanzhi Liang, Linchao Zhu, Xiaohan Wang, and Yi Yang. A simple episodic linear probe improves visual recognition in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9559–9569, June 2022.

[49] Arun Saseendran, Lovish Setia, Viren Chhabria, Debrup Chakraborty, and Aneek Barman Roy. *Impact of Noise in Dataset on Machine Learning Algorithms*. February 2019.

[50] Rashida Hasan and Cheehung Chu. Noise in datasets: What are the impacts on classification performance?:. In *Proceedings of the 11th International Conference on Pattern Recognition Applications and Methods*, page 163–170. SCITEPRESS - Science and Technology Publications, 2022.

[51] R Devon Hjelm, Alexey Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Philip Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[52] Odd Nakken. Fisheries sonar. *Fisheries Research*, 3:81–82, January 1985.

[53] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist`, 2010. Accessed: 2024-10-30.

[54] Patrick J Grother and KK Hanaoka. Nist special database 19. *Handprinted forms and characters database, National Institute of Standards and Technology*, 10:69, 1995.

[55] Hyun-Ju Lee. Introduction to convolutional neural network using keras: An understanding from a statistician. *Communications for Statistical Applications and Methods*, 26(6):591–605, 2019.

[56] Jun Cheng. Brain tumor dataset. April 2017. Accessed: 30 October 2024.

[57] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[58] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[59] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.

[60] Dongpo Xu, Shengdong Zhang, Huisheng Zhang, and Danilo P. Mandic. Convergence of the rmsprop deep learning method with penalty for nonconvex optimization. *Neural Networks*, 139:17–23, 2021.

[61] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. John Wiley & Sons, 2013.

Appendix A: Ethics Declaration

RHODES UNIVERSITY
Where leaders learn

RESEARCH ETHICS DECLARATION

To be included in the Appendices of research papers/dissertations/theses submitted for postgraduate examination where research did not involve interaction with human participants, or the use of animal subjects, and therefore did not require research ethics approval.

Candidates whose research did require ethics clearance must include their ethics approval letter in the Appendix of their examination submission.

Name of Candidate: Georgina Bianca Fiorentinos

Name of Supervisor: Prof. Marcellin Atemkeng

Degree: Masters of Science (MSc) in Mathematics

Title of research: Signal to Noise Dynamics and Feature Robustness in Convolutional Neural Networks

DECLARATION

I declare that my research did not require ethical clearance because (tick all that apply):

I did not collect data from human participants or animal subjects	✓
I used previously collected data that had already received ethics clearance.	
I analysed documents / open-access digital texts that are freely available in the public domain.	✓
I did a literature review/analysis of theoretical or secondary material only.	
I used human datasets of non-sensitive information that are either anonymous (identifiers were never collected) or have been deidentified (identifiers have been completely removed).	✓
I used commercially produced human biological material (e.g. established human cell lines).	
I observed people in public spaces and natural environments where they had no reasonable expectation of privacy and I did not interact with them or intervene in any way.	
I used non-living animal materials (eg bones of already deceased organisms or fossils) while complying with any custody and/or jurisdiction requirements.	
I did a content analysis of public media (newspapers, advertisements, and social media posts).	
I did a simulation study with no real-world consequences and does not involve disturbing or distressing content.	
I observed flora, fauna, and ecosystems without interfering with or disturbing their natural state while complying with any jurisdiction requirements.	
Other (Please provide details):	

Signature of Candidate: Fiorentinos

Date: 02/12/2025

Signature of Supervisor:

RHODES UNIVERSITY
Where leaders learn

Date: 02/12/2025