Beyond Correlation: How the Explainability of Artificial Neural Networks Misaligns with Human Cognition

Kaito Soga
e249378@edu.sbl.ch
M4d

Supervisor: Lesther Zulauf
lesther.zulauf@sbl.ch

Maturaarbeit 2025/26
Gymnasium Münchenstein

October 21, 2025

Abstract

In recent years, explainable artificial intelligence (XAI) has gained increasing importance and attention in research [15, 41, 52]. Its goal is to provide humans with a better understanding of the internal decision-making processes of black-box models [25, 42], i.e., machine learning (ML) systems, typically artificial neural networks (ANNs), whose complexity makes them difficult to intuitively understand [22, 42]. However, while the field of ML has advanced rapidly in recent decades, XAI has seen comparatively few fundamental breakthroughs [6, 52].

In this paper, I investigate the root causes of the challenges and limitations faced in explainability and identify the intrinsic properties of ANNs that constrain explanations at both the mathematical and conceptual level. To this end, I implement a set of experiments designed as case studies to demonstrate and visualise the implications of my theoretical findings. I focus on feedforward networks (FFNs) applied to optical character recognition (OCR) as a representative and interpretable task. Because FFNs are embedded in many modern architectures [20, 57], the limitations observed herein remain relevant in contemporary AI systems [25, 41].

Drawing on related insights from cognitive science [3, 30], I highlight key mismatches between FFNs and human reasoning [19, 30], most notably, correlation in place of causation and arbitrary composition in place of structured concepts. Whereas human reasoning is characterised by causal inference and compositional structure [19, 30], FFNs rely on distributed, correlation-based representations [18, 22]. I conclude that this misalignment with human cognition creates a fundamental limitation to the explainability of modern AI.

Contents

Preface

Motivation

Yann LeCun, a pioneer of machine learning, noted that “before we reach Human-Level AI (HLAI), we will have to reach Cat-Level & Dog-Level AI” [61]. His statement reflects the current limitations of modern AI and the need for novel approaches to more human-aligned AI. I think this is both important and highly interesting. Explainability, defined as the extent to which humans can understand an AI’s decision-making, faces several consequences arising from this gap between machine intelligence and human cognition. Bridging this gap and thereby improving explainability would greatly increase trust and safety in artificial intelligence. I believe that identifying the architectural limitations in explainability can help in understanding the core limitations that need to be resolved in future work.

Acknowledgements

I would like to sincerely thank Mr Zulauf for his guidance throughout this project, which helped me structure the focus of my research. I am also grateful to Prof. Dr. Hermann for providing useful references from a parallel field and to Prof. Dr. El-Assady for her presentation and the short but interesting conversation we had, which offered valuable perspectives for my work.

Statement on AI Assistance

I used ChatGPT (OpenAI) for language correction (grammar/vocabulary) and for the literature search. The tool did not generate analyses, results, or figures. I checked all AI suggestions independently and only cited verified sources. Responsibility for all content remains with the author [45].

Declaration of Authorship

I confirm that I have carried out this work independently. Contributions from others are clearly marked as such. All tools and sources used are listed in accordance with the rules of academic work.

Place, Date Kaito Soga

1 Introduction

1.1 What Is Explainability in AI?

Explainability in AI refers to the extent to which humans can understand the internal processes and decision-making logic of an artificially intelligent system [41]. The so-called *black-box* models are models with internal processes that are inherently difficult to understand and, therefore, cannot be explained intuitively [25]. Explaining a model, especially artificial neural networks, means understanding the features of the input to which it pays attention and its internal transformations that lead from input to a particular output [41]. An explanation essentially tries to express internal model reasoning, i.e., the model's internal operations transforming inputs through hidden representations to arrive at predictions, in human-comprehensible terms, to obtain a sense of reason for a model's predictions [15].

Interpretability is often used interchangeably but should be used more narrowly as understanding individual components (e.g., weights, neurons), while explainability encompasses the overall process and justification of model decisions [39, 42]. It is the difference between recognising *that* A has caused B (interpretation) and understanding *why* A has caused B (explanation), with A and B being the input feature and prediction.

1.2 Why Is Explainability Important?

Understanding model decisions is particularly important in fields that require transparent reasoning and objectivity from an AI model, such as medical diagnosis, autonomous vehicles, legal decisions, etc., all of which have implemented AI to date [41, 52]. The frequently mentioned term *transparency* refers to the ease with which a model can be interpreted and understood [9, 25]. That being said, the accessibility of a model's parameters alone does not make it fully transparent [42, 39]. Understandable reasoning and justification are what make a model transparent [41, 9]. A more transparent model thus facilitates explaining model decisions, which, in turn, increases trust in and control over an AI's decisions [12, 25]. In other words, explainability is not only a technical goal but a necessity for human alignment: if AI systems are to make decisions that affect humans, they must do so in a way that humans can understand and evaluate [19, 30].

1.3 Human Cognition

Assuming that challenges in model explainability originate from a fundamental difference between human and machine intelligence, it seems appropriate to refer to insights from cognitive science and neuroscience that establish how human reasoning relies on conceptual and compositional abstraction [3, 19, 30, 54]. Physical experience and general learning skills are linked to these processes [30, 34]. Humans tend to explain their decisions by decomposing them into understandable sub-concepts (concepts that compose higher-level concepts) and causal rules (reliable rules defining causal relationships) [19, 30].

There is no analogue of the human sensory experience in artificial neural networks [34]. This difference makes explainability a technical challenge strongly connected to cognitive aspects [30]. What humans expect from an explanation (causal, compositional, concept-based reasoning) differs fundamentally from what neural networks provide (distributed, correlation-based representations) [18, 19, 39, 22]. Thus, identifying these differences and establishing *how* they constrain explainability may give rise to more human-aligned architectures and the incentive to bridge the gap between human and machine intelligence [52, 58].

1.4 Motivation and Case Study

To investigate these limitations in a controlled environment, I used a feedforward neural network (FFN), specifically a multilayer perceptron (MLP), i.e., a fully connected FFN with a sequence of affine transformations and nonlinear activations, trained on an optical character recognition (OCR) task for empirical observations [22]. The FFN maps an input vector $x \in \mathbb{R}^n$ to an output $y \in \mathbb{R}^m$. Despite their simplicity, FFNs remain fundamental components of modern architectures: convolutional neural networks typically end in fully connected layers, and transformer blocks are built around feedforward sublayers [22, 20]. As a consequence, more complex models often inherit the limitations seen in FFNs [40, 57]. Precisely *how* the FFN is implemented in modern architectures is described in Section 6.3. I chose OCR on handwritten digits to serve as a demonstration task [42] because it provides a natural domain for comparing human and machine reasoning on a visually interpretable task, which reinforces the purpose of this case study in supporting more generally applicable theoretical assumptions about the limitations faced in explainability, even in modern architectures [12, 41]. The case studies thus serve not merely to visualise OCR, but as a tool to demonstrate the theoretical predictions and assumptions made about architectural limitations [25, 41].

1.5 Research Questions and Hypotheses

Building on the above motivation, the following four **research questions** will be addressed:

- **(Q1)** What problems does the field of explainability concern itself with?
- **(Q2)** Where do the limitations that contribute to these problems originate?
- **(Q3)** Why do current neural architectures work as well as they do, despite these limitations?
- **(Q4)** How could resolving these limitations be valuable?

The central focus lies on question Q2: identifying the architectural constraints of feedforward networks that make their decision-making difficult to explain in human terms. Given that explainability revolves around the human understanding of a model decision, the framework for investigating this question will be based on comparison to human cognition, assuming that any challenge faced in explaining neural network decision-making must arise from a difference in machine and human reasoning [12, 19, 30]. The purpose of questions (Q1), (Q3), and (Q4) is to contextualise this research, discuss the implications of the results, and provide an outlook on possible future research.

To answer these questions, I will first establish theoretical hypotheses and predictions about the intrinsic properties of FFNs that challenge explainability. Especially hypotheses Q2 and Q3 will then be tested or demonstrated through controlled experiments on the OCR task. The purpose of these experiments is not performance but demonstration: they serve as case studies to make visible the consequences of the theoretical properties, which extend beyond FFNs to modern architectures.

Hypotheses:

- **(H1)** Existing methods follow a unifying objective: to relate internal decision-making to human-interpretable and justifiable causes and to explain machine learning processes in terms aligning with human reasoning.
- **(H2)** Limitations arise from the intrinsic architectural properties of FFNs: arbitrary data representations, correlation-based processing, lack of compositionality, and high-dimensional nonlinearity.
- **(H3)** Neural networks achieve strong performance by approximating patterns that are the consequences of underlying rules, even without inferring the rules themselves if given sufficient training data.
- **(H4)** Overcoming these limitations could improve trust (the extent to which a model prediction is expected to be based on acceptable reasons) and generalisation (the maintaining of high accuracy beyond seen training data) by aligning model reasoning more closely with human cognition.

The properties of FFNs demonstrated in this research are not limited to simple models but can be inherited by modern architectures and, as later discussed, display central limitations recognisable in existing architectures.

2 Related Work and Background

2.1 Mathematical and Architectural Properties of Feedforward Networks

A feedforward network (FFN) [22, 51], often a multilayer perceptron (MLP) [5, 49], the simplest case of an FFN, is technically a large, nonlinear function that passes information forward in one direction. Formally, an FFN is defined as a function f_θ

$$f_\theta : \mathbb{R}^n \to \mathbb{R}^m,$$

where n denotes the input dimension, m the output dimension, and θ the set of trainable parameters. The network is composed of L layers, each of which consists of an *affine transformation* followed by a nonlinear *activation function* [5, 22].

Let $x \in \mathbb{R}^n$ be the input. The computation of output h of layer l is given by

$$h^{(l)} = \sigma\left(W^{(l)} h^{(l-1)} + b^{(l)}\right),$$

where $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{d_l}$ denote the weight matrix and bias vector of layer l. The purpose of the bias is to shift the output of the activation function, for instance, when the input is 0 and the weight loses its influence, the output can be prevented from remaining at 0 [5, 22]. $\sigma(\cdot)$ is a nonlinear activation function (e.g., hyperbolic tangent (tanh) [22, 27] or Rectified Linear Unit (ReLU) [22, 44]). Its purpose is to normalise data as it flows through the network and to allow the model to learn more complex and nonlinear patterns. This is called the forward pass of the network, used for inference, i.e., computing the predictions of the network. With the input $x = h^{(0)}$, the output of the network is then

$$f_\theta(x) = h^{(L)}.$$

Locally, a layer of size d_l has d_l neurons. Each neuron takes d_{l-1} inputs and computes the activated sum of the products of the inputs and their respective weights. This is a so-called perceptron [49, 51]. For instance, given inputs x_1, x_2, x_3 and weights w_1, w_2, w_3, it computes:

$$output = \sigma(x_1 \times w_1 + x_2 \times w_2 + x_3 \times w_3),$$

which can be visualised for multiple layers as follows:

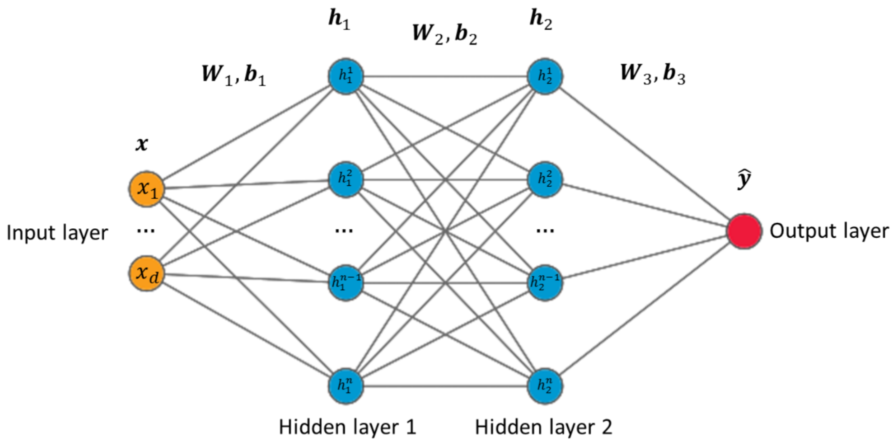


Figure 1: A visualisation of a perceptron with two hidden layers (reproduced from [62], CC BY 4.0).

This feedforward network, and more specifically, the multilayer perceptron, assumes a fully connected architecture, i.e., every neuron in layer $l - 1$ is connected through weights to every neuron in layer l. The architecture's nonlinearity and its dense connectivity between all layers are what allow FFNs to approximate a variety of patterns in data and make it serve as a universal function approximator [5, 22].

It is important to note that without a nonlinear activation function σ, the network could only approximate linear functions. The simplest example may be the approximation of logic gates. Projecting AND and OR gates to the coordinates x and y, where $x, y \in \{0, 1\}$ means the AND operator separates coordinate (1, 1) from (1, 0), (0, 1), and (0, 0). The OR operator separates (0, 0) from (0, 1), (1, 0), and (1, 1). In both cases, a linear function is sufficient to separate the classes. However, the XOR logic cannot be approximated by a linear function and requires the nonlinearity introduced by the activation function.

However, this architecture does not come without limitations [25, 39]: given their design, FFNs' internal representations of data are distributed across many neurons, the transformations from layer to layer are highly nonlinear, and the resulting decision boundaries, i.e., the manifolds in the n-dimensional input space defining where class m_a changes to class m_b [22, 38], are high-dimensional and beyond intuitive or easily interpretable. These properties make FFNs powerful, but also earn them the title of *black-boxes*, models whose internal decision-making processes are difficult to understand.

Furthermore, the number of parameters grows rapidly with the size of the network. That is, for an architecture with input dimension d_{in}, hidden layers of sizes $d_1, d_2, \ldots, d_{L-1}$, and output dimension d_{out}, the total number of (trainable) parameters is

$$\sum_{l=1}^{L} \left(d_{l-1} \cdot d_l + d_l\right),$$

where $d_0 = d_{\text{in}}$ and $d_L = d_{\text{out}}$. Even for smaller networks, this number becomes fairly large, representing a strongly *overparameterised* network, i.e., it holds many almost identically optimal weight configurations.

Such networks produce high-dimensional decision boundaries that are hard to understand and directly visualise. There are a number of *dimensionality reduction* techniques, such as Principal Component Analysis (PCA) or Uniform Manifold Approximation and Projection (UMAP) [7, 22], that can project these boundaries into two or three dimensions. The true geometry of the decision boundaries, however, is distorted by such projections and would be beyond human comprehension.

Figure 2 plots the highly nonlinear and unintuitive decision regions of a ten-class FFN when reduced to an interpretable space using UMAP. It visualises the digits from the MNIST dataset.

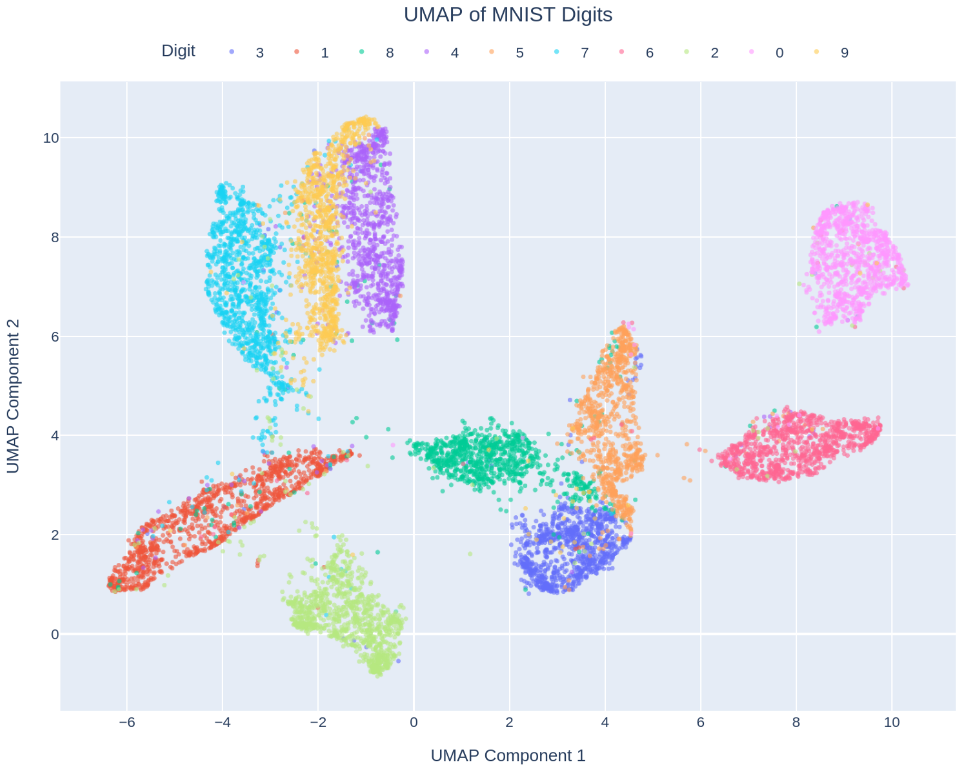


Figure 2: 10 handwritten digit classes reduced to 2 dimensions in UMAP

2.2 Weight Initialisation

Before a feedforward network can be trained, its parameters θ must be assigned initial values. This process is known as *weight initialisation* and is critical because it defines the starting point of optimisation and influences the stability of training [21, 26].

If weights are initialised too small, gradients can vanish, whereas weights initialised too large may cause gradients to explode. A naive approach, such as generating each weight randomly from a simple uniform distribution or setting all weights to zero, would cause unstable training. Therefore, weights are usually initialised with a probability distribution chosen specifically to balance the variance V between layers.

Given the pre-activations $z^{(l)}$ in a hidden layer l:

$$z^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)},$$

the variance of $z^{(l)}$ should remain approximately constant across all L layers to prevent vanishing or exploding activations. One method often used in combination with the *tanh* activation function is *Xavier initialisation* (also called Glorot initialisation) [21], which sets

$$W_{ij}^{(l)} \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{d_{l-1}+d_l}}, \frac{\sqrt{6}}{\sqrt{d_{l-1}+d_l}}\right),$$

where d_{l-1} and d_l are the input and output dimensions of layers $l-1$ and l, and $W_{ij}^{(l)}$ is the weight between neuron i in layer l and neuron j in layer $l-1$. As a result, the gradients in the backpropagation and the activations in the forward pass have a variance of roughly 1.

For rectified linear units (ReLU), the above distribution would not work well, and one should rather use *He initialisation* [26]:

$$W_{ij}^{(l)} \sim \mathcal{N}\left(0, \frac{2}{d_{l-1}}\right).$$

ReLU benefits from higher variance because half of its outputs are expected to be zero. The term $\frac{2}{d_{l-1}}$, the variance V in the normal distribution $\mathcal{N}$, accounts for this by using a higher variance, thereby preserving better signal flow when passing forward through ReLU activations.

Biases $b^{(l)}$ are usually initialised to 0 or small constants > 0, since they are not multiplied but added to the pre-activation sums, meaning they will not remain at 0. The basic concept is that initialisation should allow gradient flow, prevent symmetry, and enable each neuron learn unique functions.

In summary, weight initialisation provides the starting distribution of weight values that are to be optimised during training. With well-chosen initialisation techniques and stable training, *effective* optimisation can be achieved in modern neural networks.

2.3 Training and Evaluation

After the weight initialisation, the weights of the network are iteratively updated to minimise the error calculated through a loss function. This process, called *training*, makes use of the **loss function**, the **gradient computation**, and the **optimisation algorithm**.

Loss Functions. A loss function measures the difference or *error* between the prediction $\hat{y} = f_\theta(x)$ a network has made and the *true label* y. The most common loss function $\mathcal{L}$ used for regression is the mean squared error (MSE) [5, 22]:

$$\mathcal{L}_{\text{MSE}}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2.$$

For binary classification, the standard loss function is the binary cross-entropy (with logits) [5, 22]:

$$\mathcal{L}_{\text{BCE}}(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N}\Big(y_i \cdot \log(\sigma(\hat{y}_i)) + (1-y_i)\cdot \log(1-\sigma(\hat{y}_i))\Big),$$

where σ is the logistic sigmoid. For multi-class classification, the binary cross-entropy generalises to cross-entropy:

$$\mathcal{L}_{\text{CE}}(\hat{y}, y) = -\sum_{i=1}^{m} y_i \log \hat{y}_i,$$

where y is a one-hot vector encoding the label, and $\hat{y}$ is the softmax probability vector holding the confidence for each class [5, 22]. Confidence can be described as the entropy of this probability distribution, where low entropy implies that the model is confident about a certain classification, and high entropy can be seen as analogous to the model guessing.

Gradient Descent and Backpropagation. Training uses a process known as backpropagation, which uses the chain rule to calculate the gradients of the loss with respect to all parameters. This is particularly useful because the gradient is, mathematically speaking, pointing in the direction of the steepest ascent. Since the objective is to find the minimum of the loss function, the update rule in simple gradient descent for parameter θ is therefore

$$\theta \leftarrow \theta - \eta \, \nabla_\theta \mathcal{L},$$

where $\eta > 0$ is the learning rate [22].

The choice of hyperparameter η is critical. If it is chosen too large, training may diverge, meaning the updates made to a parameter are so large that they *overshoot* the minimum, i.e., even though the gradient is pointing in the right direction, gradient descent may fail to achieve an ideal amount of change in that direction. Conversely, if η is chosen too small, convergence can be too slow, meaning the minimum can be found very precisely, but reaching the minimum takes too many weight updates and time.

Optimisation Algorithms. To improve vanilla stochastic gradient descent (SGD), modern methods like momentum are usually applied [50]. An optimiser's job in training is to modify the model parameters θ to more effectively minimise a loss function $\mathcal{L}(\theta)$. While plain stochastic gradient descent (SGD) simply updates parameters in the direction of the negative gradient,

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L},$$

this approach often converges slowly and results in oscillating updates. Many improved optimisation algorithms have been developed to make learning more efficient and stable.

Adam Optimiser. The Adam Optimiser (Adaptive Moment Estimation) [32] builds on the idea of an existing optimiser called momentum. In gradient descent, the momentum smooths gradient updates by keeping track of an exponentially weighted average of past gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad \theta \leftarrow \theta - \eta m_t,$$

where $g_t = \nabla_\theta \mathcal{L}(\theta_t)$ is the gradient at step t, and $\beta_1 \in [0, 1)$ is the *momentum coefficient* acting as a ratio between gradients at step t and $t-1$. It becomes *exponentially* weighted because it is essentially a recursive function with base case m_0. The moving average acts like a *first-order term*, making updates continue in consistent descent directions.

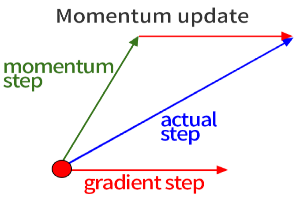


Figure 3: A vector-based visualisation of an optimised momentum step (reproduced from [63], CC BY 3.0).

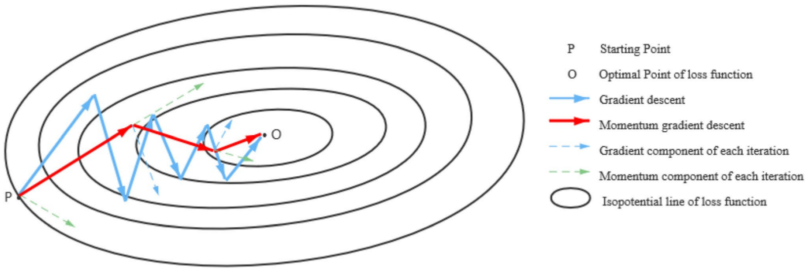

P
O
P Starting Point
O Optimal Point of loss function
Gradient descent
Momentum gradient descent
Gradient component of each iteration
Momentum component of each iteration
Isopotential line of loss function

Figure 4: SGD compared to momentum visualised on the surface of the loss function (reproduced from [64], CC BY 4.0).

RMSProp [50] extends the idea of an exponentially weighted average with *squared* gradients,

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2,$$

which can be viewed as a *second-order term* that adapts the learning rate to the variance of each parameter. This results in *parameter-specific* updates, where large squared gradients reduce the step size, while small squared gradients increase it.

Adam combines both of these ideas. It combines the mean of past gradients (first moment) and the variance of gradients (second moment) [32]. However, with m_0 and v_0 initialised at 0, the gradient at time step t will initially be too small to approximate the mean correctly and will instead be biassed toward 0 because there have not yet been enough gradients to increase the mean. To correct for this bias, Adam computes bias corrections:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

where the denominator $1-\beta_1^t$ or $1-\beta_2^t$ is the factor in m_t or v_t that would cause a small mean when t is initially still small. The update rule for the parameters is then

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},$$

where η is the learning rate and ϵ is a small constant to prevent division by zero.

Adam can be understood as using two types of momentum: one that represents the magnitude (variance) of previous gradients and another that accumulates past gradients. The most widely used algorithm for training artificial neural networks at the moment is the result of this combination, which is both stable and adaptive.

Evaluation. To measure generalisation after training, the model is tested using testing data that is strictly separated from the training data.

Generalisation can conceptually be described as the model's ability to successfully predict unseen data labels outside of the training data. Its ability to do so roughly implies that it has found a *general* function that approximates the original rule responsible for the patterns in the dataset. Essentially, evaluation is a measure of how little the model has merely memorised the training data.

In summary, training updates randomly initialised parameters into an effective function mapping inputs to outputs, and evaluation is the final measure of success, distinguishing a good approximation from *overfitting.* An overfitted network has approximated a function specific to the training data [5, 22] rather than one that is generalisable to unseen data.

2.4 The Role of FFNs in Modern Architectures

While fairly simple, the FFN explained in further detail in Section 2.1 is incorporated in many modern architectures, and limitations found in it can thus become inherited traits in modern AI [20, 22]. In particular, two prominent examples of architectures based on the FFN are considered to demonstrate the importance of an FFN in modern AI, namely CNNs and transformers.

Transformers. A transformer embeds input sequences, usually words, into dense vectors to process them. The relationship between each word (token) and the others in its sequence is then found by computing self-attention [59]. It must be noted that this paper deals with a different notion of *attention* of neurons in later sections, which is to be distinguished from attention in the context of transformers. This self-attention mechanism effectively correlates the data. After attention, each sequence is passed through a feedforward network that transforms its dimensions, adding nonlinearity and a more complex data representation but it does not directly model relations between tokens [20, 59]. Thus, the FFN's role in a transformer is not to learn data patterns but to improve the representation of the data already related by attention [59].

Convolutional Neural Networks (CNNs). CNNs use kernels, which are weight matrices smaller than the input that move over the input image and detect local patterns like edges, curves, or textures, creating feature maps [22, 35]. Convolutional layers combine these *feature maps* created by each kernel, which highlight the locations in the input where the learnt patterns occur, to create more abstract representations of the image [22]. Finally, these feature maps are flattened into a one-dimensional vector and passed through an FFN, which is responsible for mapping it to a prediction over output neurons [35]. Essentially, the CNN relies on the FFN to interpret the extracted features into an output decision.

2.5 Baseline Methods in Explainability

There are several approaches to explaining the decision-making of machine learning models, the most prominent of which are outlined below to establish an overview of the currently available methods. At the highest level, there is a distinction between *post-hoc methods*, which aim to explain model decisions after training, and *intrinsically interpretable models*, the architectures of which are designed to be transparent from the start. This distinction is important for my research, as in this paper only post-hoc methods are explored. This is both due to the fact that intrinsically interpretable models tend to differ a lot in architecture to FFNs, which my research relies on and focuses on, and due to the fact that intrinsically interpretable models do not generally perform as well as standard neural networks [41, 42]. In particular, the latter is what makes the explainability of modern AI more urgent than having a custom architecture that is intrinsically interpretable but less accurate: the industry strongly favours the most performant models. Thus, to achieve safe and trustworthy AI, the focus should lie on making current architectures that perform well in crucial tasks more explainable rather than building less performant and interpretable models. Therefore, this paper focuses on *why* post hoc explanations face fundamental limitations related to neural architectures themselves [6, 52].

Historically, the first post-hoc techniques, such as *LIME* and *SHAP* [42], became popular for their model-agnostic nature, i.e., they did not need to know the model architecture, nor were they limited to certain architectures.

LIME (*Local Interpretable Model-agnostic Explanations*) builds a simple surrogate model (i.e., a simpler and thus more interpretable approximation) around a single prediction. The idea is that even if the global behaviour of a neural network is highly complex, it can be approximated locally around a single input by an interpretable model. Specifically, LIME creates slightly modified versions of the input (perturbations) and tests how the predictions change. From these local variations, it estimates which features were most influential for the specific prediction [48].

SHAP (*SHapley Additive exPlanations*) is based on Shapley values from game theory, where each feature can be considered a *player* that contributes to the model's prediction. The overall prediction is then fairly distributed among all input features, according to how much each one contributes on average when considered in different combinations with others. SHAP assigns each feature an importance score, which is well-designed theoretically but computationally expensive to compute in practice.

While LIME and SHAP approximate the influence of features on predictions, there are techniques that analyse gradients or activations inside the network directly. The simplest implementation of this is a *saliency map* [29] that shows how sensitive the model is to regions of the input by computing the gradient of the output with respect to the input [18]:

$$S(x) = \left| \frac{\partial f(x)}{\partial x} \right| .$$

The resulting heatmap visualises how strongly each pixel affects the prediction. This is simple to compute, but tends to be unstable and noisy because each pixel is tested as a separate feature [18, 41].

LRP *(Layer-wise Relevance Propagation)* [4] instead uses the prediction score as relevance and propagates it backward through the network. For example, for neuron activation a_j in layer l contributing to neuron k in layer $l+1$, the relevance is propagated from layer $l+1$ to l as

$$R_j^{(l)} = \sum_k \frac{a_j w_{jk}}{\sum_{j'} a_{j'} w_{j'k} + \epsilon} R_k^{(l+1)}.$$

This means that the relevance of an entire layer is conserved, and the contribution of one neuron relative to the rest of the layer is computed. Propagating backward to the input results in more structured attribution maps that make it clearer how each input dimension contributed to the output [42].

Activation maximisation [17] takes the opposite direction. Instead of testing *how* inputs contribute to an output, it tests *what* input maximises the activation of a given neuron. Formally, for neuron i in layer l with pre-activation $z_i^{(l)}(x)$, the maximised input x^* is computed assuming

$$\mathcal{L}(x^*) = -z_i^{(l)}(x^*),$$

which, in terms of gradient descent, means

$$x^{(t+1)} = x^{(t)} - \eta \nabla_x \mathcal{L}(x^{(t)}) = x^{(t)} + \eta \nabla_x z_i^{(l)}(x^{(t)}), \quad \text{with } x^* = x^{(T)},$$

where backpropagation is essentially reversed, and the input is being optimised through gradient descent with respect to the negative activation of a neuron, that is, the loss. The input is initialised as noise and the resulting x^* visualises which patterns are most strongly associated by that neuron.

Lastly, there is the method of *counterfactual explanations* [42, 60]. Counterfactuals (CFs) ask what minimal change in a specific input would change the model prediction. For an input x with prediction y, a counterfactual $\tilde{x}$ is found such that

$$f(\tilde{x}) = y' \neq y \quad \text{while minimising } d(x, \tilde{x}),$$

where d is a distance metric for similarity. Counterfactuals are useful for their human intuitiveness ("Had feature A been different, the outcome would have changed") but finding perfect solutions is often computationally infeasible [42, 52].

In conclusion, these baseline methods display the variety of existing attempts at post-hoc explainability. However, they all share a reliance on correlational associations between input and output [6, 39]. Despite human understanding striving for causal explanations and reasoning, these methods do not address the deeper architectural limitations that may cause neural networks to resist such human-like explanations [19]. This motivates the theoretical investigation in Section 3.2.

2.6 Goals of Explainability

From the variety of existing methods, it becomes clear that model explanations have a unifying intention: to enable a human observer to better understand the input-output relations created by a machine learning model [41]. A more precise way to frame this goal is as a *translation problem*, i.e., converting the internal representations of machine learning models into reliable forms (e.g., concepts, symbols) [39, 42]. While humans use shared languages, common assumptions, and low-level concepts (i.e., concepts composing a higher-level concept) to express their internal reasoning, machines instead process high-dimensional representations and statistical associations that are far from intuitive [22]. Effective explainability must therefore bridge this gap, expressing the model's internal processes in human-interpretable terms without altering or falsifying its meaning [12, 19, 30].

More generally, and in the context of my research, I define a model explanation as follows: given the model's internal parameters, an explanation must make the input-output relations of the model visible and allow one to understand *why* a certain relation has been established, so that one can evaluate whether a model decision resulting from this relation can be justified [39, 42].

Although the existing explainability methods do not directly achieve this by providing a clear reason behind a model's prediction, they aim to facilitate the process of understanding and inferring why a model made a certain decision [6, 52].

This framing also links to the central research question Q2 of this paper. If explainability methods aim to translate model reasoning into human terms but repeatedly fall short, then the challenges and limitations may not stem from the methodology but from the nature of the reasoning to be translated [19, 30]. In other words, the difficulty of explainability could be intrinsic to the conflicts between neural networks and human cognition, rather than to poor methodology [52]. Therefore, I propose that explainability should be concerned more with architectural limitations than with the methods applied in an attempt to improve explanations, given that there may be inherent limitations to such explanations [25, 30, 41].

2.7 Existing Limitations in Explainability

Explainability assumes that the explanation is for a human interpreter, meaning the ultimate challenge faced in this field will be to make complex machine intelligence reasonable and suitable for human cognition [19, 30]. Although existing explainability methods, which were investigated in Section 2.5, have demonstrated useful insights, there are several well-known limitations that they fail to fully account for [15, 52]. First, even smaller modern networks consist of millions of parameters, making their high-dimensional decision boundaries difficult to visualise or interpret directly [25, 42]. Second, nonlinear transformations across layers make input–output relations unintuitive, i.e., they may be mathematically traceable but become cognitively inaccessible [19, 30]. Third, internal representations are not always meaningfully structured. A model's representation of information may appear arbitrary, despite resulting in high accuracy [18, 39]. These limitations further reinforce the idea that the problem of explainability is not merely methodological but also rooted in the underlying architecture of neural networks.

3 Theoretical Foundations

3.1 Human-Interpretable Features: A Working Definition

In this paper, the term *human-interpretable* is used to describe something as non-arbitrary, i.e., interpretable to humans. Specifically, in the case of the OCR task considered, the term is used in the context of visual features. A visual feature is considered *human-interpretable* if it fulfils the following:

(1) **Connectivity:** The pixels composing the feature must form a single connected component in the 2D input space.

(2) **Localisation:** The pixels are in a spatially limited and recognisable region, not creating random noise across the input.

Formally, let $x \in [0,1]^{28\times 28}$ be the input image, and let $C \subseteq \{(i,j) \mid 1 \leq i,j \leq 28\}$ be a set of pixel coordinates of a concept C. Then C is human-interpretable if:

(1) $\forall (p,q) \in C,$ there exists a path of adjacent pixels in C connecting p and q,

(2) The spatial extent of C is small and concentrated, i.e., $\text{diam}(C) \ll 28,$

where $\text{diam}(C)$ denotes the diameter of C in pixel space. This definition does not assume a human-interpretable feature to carry meaning through concepts known a priori.

3.2 Why Explaining Decisions Is Difficult

This subsection extends Section 2.7 where limitations to the explainability of neural networks were discussed rather conceptually. Especially research question Q2, being *where* the limitations to *explainability*, as defined in Section 2.6, originate, is discussed in this section. Given the mathematical foundation of neural networks, I intend to investigate and hypothesise *how* the architectural constraints of artificial neural networks, in particular the multilayer perceptron, contribute to the limitations and problems faced in explainability. I will frame the challenges in explainability as a direct consequence of the fundamental design of an FFN by arguing, supported by theoretical arguments, that the problems in explainability do not merely come from poor methodology but from limitations intrinsic to an FFN's architecture. The consequences of these proposed theoretical limitations will be further supported by the experiments in Section 5 and discussed in Section 6.

Parameters and Composition. Given the layer-wise forward pass of neurons that become the activated sum of weights and inputs, certain assumptions can be made about the nature of *information preservation*. Since information is compressed as a sum and passed forward, information about the summands is lost. Considering the pre-activation sum of weights A, B and inputs x, y, the sum z of a single neuron before activation is calculated as

$$z = A \times x + B \times y$$

It becomes evident that for a particular value $\tilde{z}$ of z and fixed values $\tilde{x}, \tilde{y}$ of x, y there exist many combinations of weights A and B resulting in the same $\tilde{z}$. Formally

$$S_z = \{(A,B) \in \mathcal{D} \mid \tilde{z} = A\tilde{x} + B\tilde{y}\}.$$

where S_z denotes the set of all possible A, B, and $\mathcal{D}$ denotes the space constrained by $(\tilde{z}, \tilde{x}, \tilde{y})$. The number of possible solutions for (A, B) is then $|S_z|$. In a practical implementation, the number of possible values for A and B would be finite. Assuming *float32* values [46], the range of values would be of size 2^{32}. Given d parameters, S_z is more precisely a hyperplane in $d-1$ dimensions. With parameters A, B and fixed inputs $\tilde{x}, \tilde{y}$, the possible combinations of values in the two dimensions A, B are constrained to a one-dimensional line where $\tilde{z}$ maintains a constant (and optimal) value. This can be extended when more parameters are added, where, for instance, parameters A, B, C create a hyperplane in three dimensions, that is, a plane of one dimension less than the space in which it exists because it is constrained to the optimal value of $\tilde{z}$, given fixed inputs $\tilde{x}, \tilde{y}$.

This implies that even in a smaller neural network, such as the toy FFN considered in this paper, there exists an enormous number of optimal solutions [18] due to the large number of degrees of freedom. This combinatorial variety of all values in the optimal hyperplane is not to be confused with the phenomenon of *overparameterisation*. An overparameterised network is one that has more parameters θ than needed to reach an optimal solution [1, 22]. This does not generally hurt the accuracy of a network, as more degrees of freedom do not restrict the process of finding the optimal parameter values. In fact, it can accelerate the process of gradient descent because, geometrically speaking, there are now more paths on the d-dimensional surface to reach a global minimum. Although having more parameters than necessary is a different phenomenon from having multiple optimal weight configurations, which implies multiple global minima, there is an important relation to be drawn between the two. Increasing the number d of parameters means *exponentially* increasing the combinatorial variety of parameters that result in an optimal weight configuration. Formally, with the constrained space $\mathcal{D}$ for optimal values of $\tilde{z}$, the parameter space for d weights is $|\mathcal{D}|^{d-1}$.

Implications for Explainability. These properties of feedforward networks have several direct implications for their explainability. The internal representations have only one constraint as to how features are processed and combined in the forward pass: minimising the loss $\mathcal{L}$. This implies that the activations and weights cannot be expected to have any meaningful structure since there is no loss that optimises for it [18]. The updates on randomly initialised weights are only computed based on the gradient of the loss with respect to each weight. Thus, any configuration of weights where the loss is minimal will be treated by the network equally as *optimal*: the composition does not matter [18]. Since every model instance has different initial weight configurations, the trained set of weights that minimise the loss in one model instance will not be similar in composition to another model instance.
It should be noted that, in practice, the minimal loss can slightly vary, as there will likely be multiple similarly low minima in the loss function.

Considering that explainability, as opposed to interpretability, aims not only to demonstrate *how* the weights of a model contribute to the output but also to explain *why* a particular set of parameters can make an optimal prediction, the arbitrary nature of parameters is a difficult aspect of explaining neural networks. Although one could restrict an explanation to the activations of the network, the arbitrary composition of those activations has a major impact on truly *justifying* the decision-making of the network, that is, there is no particular cause provided for the weight's composition apart from "being optimal". Essentially, the network finds an optimal weight configuration using an efficient search algorithm (gradient descent + optimisation) but cannot provide a *reason* for the found weights' optimality.

With respect to research question **Q2**, two major consequences can be predicted (P1, P2) given the derived theoretical properties:

- **(P1)**: The FFN's internal representation of data is not composed of meaningful, lower-level structures that could be interpreted as or associated with known concepts. The only consistency across different weight initialisations is the minimisation of loss, for which arbitrary weights are sufficient, i.e., they do not need to have meaningful structure across model instances.
- **(P2)**: The FFN lacks causal inference, relying purely on correlation optimised by gradient descent, which hurts explainability by not providing sufficient *cause* for the justification of model predictions.

The lack of meaningful composition will make it impossible to directly interpret *why* a certain weight configuration results in accurate predictions [25, 42], because it is not based on existing, lower-level concepts that can be described in human terms. Instead, they are arbitrary and only approximate what causal reasoning could infer, but they lack a *rule-based* cause behind their weight configuration. Therefore, there is no true reasoning *beyond correlation* [19, 30, 31] when training the network, which raises the question of what the difference may be between a machine's pure approximation of a rule and causal reasoning.

Reasoning, Rules and Patterns. Interestingly, the answer to what defines the act of reasoning, in the context of image classification, lies in the equivocal noun "reason". That is, the term reason means both the *justification* of a thing and its *cause.* The *justification* of an AI's predictions is the verification of its soundness. *Cause*, on the other hand, could, in the example of handwritten OCR, be interpreted as referring to the cause of a pattern in the image data, that is, a rule.

Therefore, I frame the generation of a pattern, such as a handwritten digit, as the consequence of a set of rules $\mathcal{R}$, which is to say that these rules define the pattern class (e.g. "if these rules apply, it is a 7"). This process of *causation* is what human cognition uses to infer the class of a pattern through *reasoning* [19, 30]: by knowing the rules that define a class, a human can infer whether a handwritten digit is of a certain class or not. This process, called *causal inference* [14, 19], is not only a useful way to make predictions with high accuracy and certainty but also a great way to *explain* a prediction on causal grounds.

Conversely, a neural network only approximates a function with the patterns that have been generated via the rules, but it never truly infers or knows the rules themselves. Therefore, the difference between human reasoning and the approximation of a pattern can be described as the difference between *correlation* and *causality* [22, 39, 19]. Formally, a neural network only correlates inputs with outputs, thereby approximating the pattern:

$$y \approx f_\theta(x)$$

whereas human cognition would infer a class given a set of rules $\mathcal{R}$:

$$y = g(x, \mathcal{R})$$

where g denotes a function for causal inference. It is important to note that such a function g would be highly complex and currently only exists in limited forms, far from reaching human-level cognition [2, 8, 13].

3.3 Summary: From Theoretical Constraints to Practical Challenges

In summary, the theoretical properties of FFNs create substantial challenges for explainability [52]. On a practical level, these mathematical constraints become clear as well-known limitations in justifying a model's decision-making [25, 52]. Therefore, many of the practical difficulties faced in explanations can be directly traced back to the underlying architecture of feedforward networks and their difference to human cognition [30]. This motivates the experimental investigations following Section 4.4, where empirical tests and visualisations demonstrate how these theoretical limitations appear in practice.

4 Methodology and Data

In this section, the methodological setup used in the experiments is outlined. First, the dataset and task, the feedforward network architecture, and its training procedure, which will form the basis of all following experiments, are described. The detailed evaluation and analysis specific to each experiment follow thereafter.

4.1 Task and Dataset

To showcase the theoretical predictions that have been made about the architectural properties of FFNs and the conclusions drawn regarding explainability, the experiments are based on a simple and visually interpretable task. Optical character recognition (OCR) tested the model's ability to classify image data. Specifically, the experiments used the Modified National Institute of Standards and Technology database (MNIST) dataset [36] of handwritten digits. This dataset consists of 2D greyscale images of 28x28 pixels, meaning pixel values are one-dimensional and range from 0 to 1. Each image depicts a handwritten digit and has a label $y \in (0, 9)$. Given that each data point in the dataset holds information in coordinates (x, y) and their greyscale value, the input data X can be formally described as:

$$X \in \mathbb{R}^{28\times 28\times 1\times n}$$

where n denotes the length of the dataset, that is, the number of images. In the considered MNIST dataset, there are 60000 training images and 10000 testing images [36]. An input $x^{(i)}$, flattened into a vector of size $28 \times 28 = 784$, is described as:

$$x^{(i)} \in \mathbb{R}^{784}, \quad i = 1, \ldots, n,$$

Typically, handwritten digits are centred in the image with comparatively little variation in the style of strokes. Therefore, the variation in the data is focused on and limited to handwriting. This makes the MNIST dataset suitable for simple internal model representations but complex enough to test and visualise the theoretically proposed limitations to explainability from Section 3.2.

4.2 Model Architecture

To showcase the theoretical arguments regarding the limitations of explaining FFNs, the experiments are based on a simple multilayer perceptron. Formally, a layer of the network computes:

$$h^{(l)} = \tanh\Big(W^{(l)} h^{(l-1)} + b^{(l)}\Big), \quad h^{(0)} = x,$$

where $W^{(l)}$ and $b^{(l)}$ denote the weights and biases of layer l, respectively, and tanh [27, 22] was used as the activation function. Given the input $x \in \mathbb{R}^{784}$ from the MNIST dataset, the FFN requires 784 input neurons and, given the integer labels from 0 to 9, 10 output neurons. To ensure that the network is not too complex but can achieve acceptable accuracy in moderate time, the hidden layers l_i were chosen as $l_1 = 32$ and $l_2 = 16$. The resulting layer sizes were:

$$784 \rightarrow 32 \rightarrow 16 \rightarrow 10$$

Considering that the network's weights and activations were inspected, it was wiser to choose an activation function that does not omit negative values and limit the range of interpretable values, as would be the case with ReLU. Therefore, *tanh* was an ideal activation function since it normalises inputs into a range from −1 to 1, formally:

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad z = x_i \times w_i$$

To achieve stable gradients across layers, Xavier (Glorot) initialisation was chosen [21]. The models to be tested in these experiments achieved a mean training accuracy of 97.95% and a training loss $\mathcal{L}$ of 0.069. The mean accuracy on the 10000 test images resulted in 96.59%, repeatedly achieving relatively high and stable accuracy.

4.3 Training Procedure

Every model instance used in the experiments was trained with identical hyperparameters (e.g., learning rate, batch-size) to avoid results being altered by different training processes. A *model instance* is one instance of the probabilistic weight initialisation, which varies across each model instance. Architecture and training remained the same. For stable gradient descent, the Adam optimiser was chosen in combination with a learning rate η of 0.001, formally:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},$$

The training iteratively updated weights for 10 epochs with a batch size of 64. This means that the network processed 64 data points (samples) in one forward and backward pass. Every epoch, the model processed all of the batches in the training data. With the model processing $\frac{N}{64}$ batches per epoch, the remainder was added as an additional, smaller batch [46]. With 10 classes of the digits 0 to 9, the model used cross-entropy for the computation of loss $\mathcal{L}$:

$$\mathcal{L}_{\mathrm{CE}}(\hat{y}, y) = -\sum_{i=1}^{m} y_i \log \hat{y}_i$$

4.4 Experimental Setups for Evaluation and Analysis

The above described FFN and the following experiments were implemented mainly in PyTorch [46], which is an ML library in Python providing highly optimised functions to efficiently train models and compute tensors. For numerical operations and when arrays were of advantage, NumPy [24], a native Python library, was used. Matplotlib served as the primary library for visualisations of data such as tensors and matrices [28].

Since the conducted experiments were intended to serve as demonstrations of the theoretical predictions about explainability made in this paper and the implications of the architecture of FFNs, the main focus of the experiments was to create visual data for interpretation and intuition, and empirical data to illustrate the intrinsic properties of FFNs, whereby they should reinforce theoretical predictions. Explicitly, the FFN's weights, biases, and activation patterns were directly inspected and visualised, customised inputs were passed through the model, and statistical data was conducted from the comparison of internal representations of data across multiple model instances.

The following sections describe the specific environments designed for each experiment, and their results will be discussed in Section 5 with respect to the predictions P1 and P2 of the research question Q2. The rationales for each experiment will also be established in Section 5.

4.5 Neuron Attention

To test the entire perceptive field of each neuron in the FFN, the input images were set as a completely white image with every pixel at a value of 1. This ensured that there was no specificity toward the input image, but rather demonstrated the entire weight distribution for a neuron, avoiding pixels of value 0, which would have falsified the visualisation by zeroing out the pre-activations $z = 0 \times w_i$. Nevertheless, an example of a real input was also considered as a practical example. With the FFN of the above specified architecture, the focus lay on a single digit, namely '8', as a representative case. The considered network has two hidden layers, which allowed the visualisation of both hidden and output layers.

For each neuron in the first hidden layer, the activation was simply the weighted and bias-shifted sum of the input pixels, passed through the activation function tanh. In the forward pass this was merely one value, the activation sum, but the intention of this experiment is to visualise the true composition and meaning of this sum. To achieve this in the following hidden layers, the summands $w_{ij} \times x_i$ of the pre-activation z were preserved and visualised as a matrix of the weighted input pixels' intensities. Since the colour scale has changed from greyscale $c_g \in [0, 1]$ to $c_g \times w_i$, the values could have both positive and negative signs, and the largest absolute value z_{max} became the upper and lower bounds. For the rendered plots, the scale was normalised into the range [-1, 1]. This provided a direct view of how each neuron represents information across the 28 × 28 input space. Formally, for neuron j in the first hidden layer, an activation is given by

$$h_j^{(1)} = \tanh\left(\sum_{i=1}^{784} w_{ji}^{(1)} x_i + b_j^{(1)}\right),$$

where x_i are the pixel values of the flattened input image and $w_{ji}^{(1)}$ the corresponding weights. Visualising each of the sum's components $w_{ij} \times x_i$ results in 58 *attention maps* for the 58 neuron activations.

For the second hidden layer and all following layers, each neuron's attention map was calculated by recombining the activations of the previous layer. To preserve the components of these pre-activation sums, the neurons represented the weighted attention maps from the previous layer. This means that for neuron k in layer $l + 1$ every attention map from layer l was weighted by w_{jk} and summed *per component* of the attention map. These values were then normalised into a range of [-1, 1] for better visualisation. Formally, the attention map M_k was computed as:

$$M_k = \sigma(\sum_j w_{jk} \times M_j)$$

where matrix M_j was the attention map of neuron j in layer l and the scalar w_{jk} the single weight from neuron j to neuron k in layer $l + 1$.

4.6 Model Instance Comparison

Structure in Attention Maps. To test the presence or absence of structure more empirically, the cosine similarities between the attention maps of two different model instances given the same inputs were computed. To account for the fact that the position of a neuron with a certain function within a layer is not fixed, the program implemented in this experiment searched for the highest possible cosine similarity between any two attention maps from model instances f_1 and f_2. Formally, the set of attention map pairs was found such that the overall similarity was maximised. This was done using the Hungarian algorithm [33] given a matrix of similarities between every possible pair of attention maps. With the matrices of two neurons' attention maps M_{n1} and M_{n2} and their respective one-dimensional flattened vectors u and v, the cosine similarity was calculated as:

$$\text{cosine\_similarity}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \, \|\mathbf{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \, \sqrt{\sum_{i=1}^{n} v_i^2}}$$

The idea behind this approach was the following: if a stable, interpretable structure were present, it should be found consistently across the 58 neuron pairs of two different model instances trained on the same task. Conversely, if no such consistency, i.e., structure, were to exist, then attention patterns would vary arbitrarily across models, and the similarity scores computed should be rather low. This analysis was repeated for each hidden layer and the output layer, computing the mean, median, maximum, and minimum similarity across all attention maps.

Structure in Activation Patterns. To test the structure of internal representations at a higher level, the cosine similarity across the same model instances was also applied to the activation sums themselves, rather than to their components. For each model instance, the raw activation vectors of both hidden layers, given the same input data, were extracted. The output layer was excluded because the activations of output neurons were expected to be specifically optimised to be similar. Thus, the focus was laid on the neurons that were not necessarily similar, leaving 48 neuron activation pairs for comparison. For the neuron attention maps, in contrast, the output layer had been included because it had only been trained to be similar in sum, but not in its composition. Unlike the fully white input image of 1's tested for the composition of activations, the input data for testing activations themselves consists directly of 1000 sample images of random labels from the testing data, because fully white inputs would not result in meaningful activation patterns. Across the 1000 images, the mean cosine similarity and its standard deviation were calculated for two cases:

Case 1: The raw (unsorted) activation vectors were compared, which meant that the positional differences of neurons contributed to the similarity score.

Case 2: The activation vectors of each layer were sorted before comparison, meaning that each neuron activation was sorted according to its raw value. This should bring about higher similarity scores in the case of consistency in the activation values, but not in their positions in the layer.

This allowed for the distinction between activation vectors that are similar both numerically and in their positions, and activation patterns that were numerically similar but lacked consistent positions.

4.7 Testing Rule Approximation

For this experiment, I assumed, given the theoretical evidence, that there exist perturbed inputs $\hat{x}^{(i)}$ such that they follow the definition of class y_i but are predicted by the model as class $y_h \neq y_i$. To effectively test and build such inputs, I developed a simple program that allowed me to edit existing inputs. Specifically, I used the Python library PyGame to edit any existing input image from the MNIST dataset. With it, I could mask or add pixels on the image, such that it became a different class. I intentionally kept the changes minimal, intending to demonstrate whether the model is able to follow rules rather than just common patterns in the MNIST dataset. If the model had generalised to and learnt the rules responsible for the patterns in the training data, then it would be able to correctly predict the perturbed inputs, given that these inputs still follow the rules that define a digit class y_i.

To make the modifications effective and straightforward to create, I chose an input image with class '1'. Passing this image through my PyGame program for perturbations, I created four different variations of it, aligning with the definitions of the digits '8', '3', '9', and '0' as clearly as possible. Then, I computed the pre-activations of the output neurons that the FFN has generated for each perturbed input $\hat{x}^{(i)}$, the plots of which will be shown in the results. The results found in these 4 examples were tested with further examples of perturbations to ensure the behaviour was not incidental.

By testing the structure and consistency of the internal data representation of the FFN and its prediction process, these experiments served to demonstrate the architectural constraints in explainability predicted in the theoretical discussion of Section 3.2. In particular, they aimed to show the lack of compositionality and causality, which inherently differ from what is known about human cognition [14, 19], thereby making certain limitations in explaining the network's decision-making inevitable.

5 Results

5.1 Neuron Attention

Rationale. To test precisely *how* neural networks process information internally and to highlight incompatibility with human cognition, the explicit attention of each layer's neurons were first visualised. The goal of this experiment was to create an intuitive and visual understanding of how information is processed and represented locally by a neuron in the FFN. Beyond visual intuition, its results serve to highlight the structural ambiguity of how neuron activations represent information, as discussed theoretically in Section 3.2.

Resulting Attention Maps. The resulting visualisations were mainly intended to provide an overview and visual intuition, rather than direct empirical evidence. The following observations are therefore descriptive, while the sections following hereafter performs more empirical analyses.

The overall 58 neurons and their corresponding *weighted attention* maps were visualised and are plotted below. A frequent yet inconsistent trend became visible: in some regions, the attention values changed gradually and formed continuous gradients rather than abruptly changing from positive to negative signs. This led to regions that appear mostly positive or mostly negative. It should be kept in mind that in hidden layers, the sign does not directly imply *how* a value contributes to the final prediction, as it can be inverted by the weights of the following layers. At the same time, there were many noisy areas where the sign (+, -) distribution seemed random and did not follow smooth patterns.

Furthermore, it became clear that no human-interpretable features with *recognisable meaning* (i.e., known concepts) were visible. Individual neurons could not be associated with a specific function, e.g., representing some known structure (loop, arc, etc.), because their attention maps lack specificity to one feature alone. Features with recognisable meaning in human terms, e.g., 'loop', 'line', 'arc', did not occur. It appeared that neurons only encode what becomes relevant in combination with other activations but remain uninterpretable when considered alone. This lack of meaningful composition became visible.

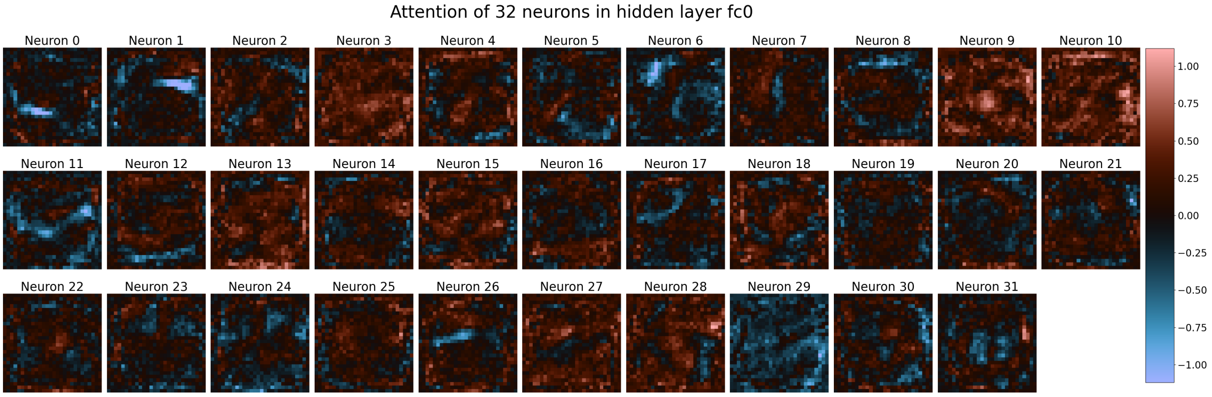


Figure 5: The first fully connected layer "fc0" with 32 neurons is shown.

Figure 5 is the visualisation of each of the 32 neurons in the first layer. The attention maps illustrate how inputs are combined and transformed. Due to values with both positive and negative signs, a diverging colour scale was helpful: the colour encodes the *sign* (red = positive, blue = negative), and the brightness encodes the *magnitude* of the activation.

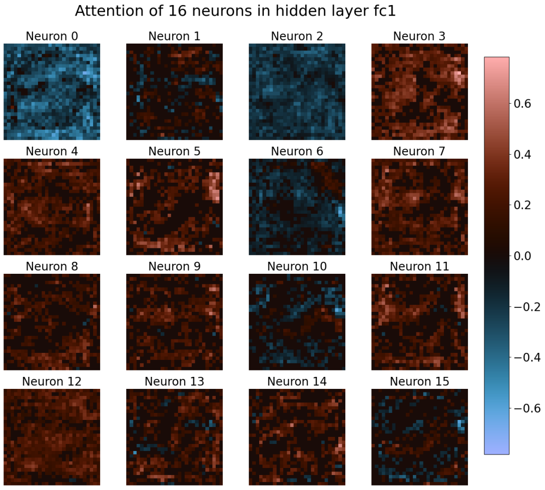


Figure 6: The second hidden layer "fc1" and its 16 neurons are shown.

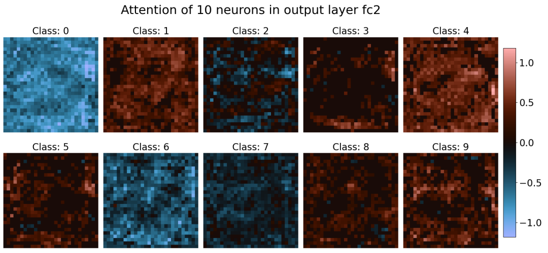


Figure 7: The output layer "fc2" and its 10 output neurons are shown.

What becomes visible in Figure 6 showing the second fully connected layer "fc1", is the comparatively low amount of *clustered regions* that mostly show one sign, which was much more common in the first hidden layer "fc0". There also tend to be more attention maps with one dominating colour, whereas layer "fc0" had more variety in each map. This lower variety in the second hidden layer will likely result in higher activation sums because positive and negative components do not neutralise each other as often. Although signs have become more uniform in each map, magnitude still creates many clustered regions of mostly high or low magnitude, that is, regions with bigger or smaller weights.

In the output layer "fc2" shown in Figure 7 the observations from the previous layers become even more evident. The colours, i.e., signs, are much more definite, meaning the attention maps are mostly composed of only one colour. In the case of two colours appearing in the same region, they appear much more scattered and mixed, creating noise rather than being separate or focused on specific regions, similarly to the previous layer "fc1". What differs in the output layer is the lack of clustered regions of specific magnitudes, i.e., the magnitude (bright / dark) now seems more scattered rather than focused on local regions (mostly bright or dark).

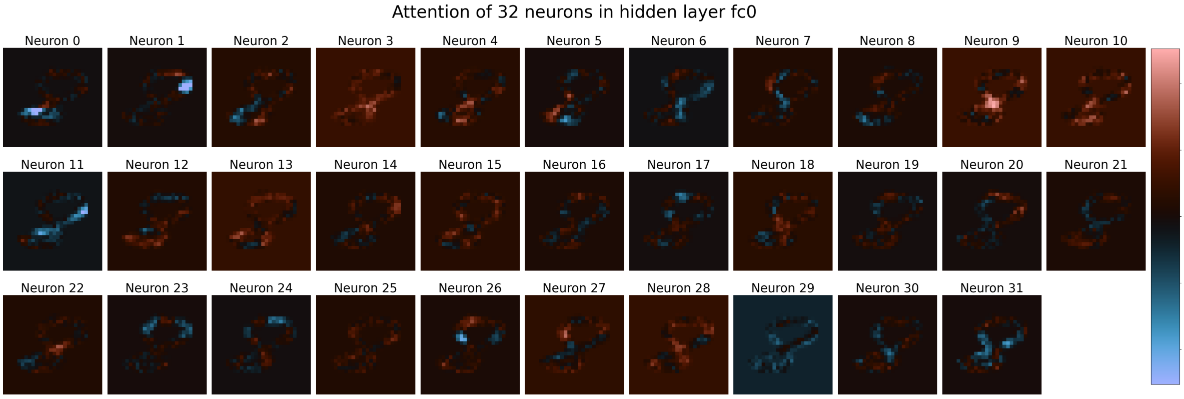


Figure 8: The first hidden layer applied to a real input of digit '8'.

To visualise not only the attention of each neuron on the entire input region, Figure 8 plots the first 32 neurons with a real input from the MNIST data instead of a fully white (pixel values 1) input, labelled '8'. This visualises the attention of each neuron being mostly limited to the digit itself, although the background is also slightly activated in most attention maps due to bias correction. This means that while the input pixel may have a value of 0, the activation becomes $0 \times w_i + b_i$. In each attention map, it becomes visible how different parts of the digit are weighted, however, the ambiguous nature of each neuron's attention remains: what a single neuron's activation encodes appears to have no *independent* meaning.

5.2 Model Instance Variation

Rationale. While the previous section has shown the lack of visually interpretable meaning in a neuron's attention map, the question remains whether there is consistency in the patterns of attention maps across model instances. If there is, explainability could benefit from this consistent pattern and assign meaning or certain functions to it. If this consistency, i.e., similar attention maps across model instances, is lacking, it reinforces the idea that arbitrary weight composition hurts explainability, as predicted in theory (P1).

Resulting Similarities of Attention Maps. The values produced by cosine similarity can range from -1 to 1, where 1 means perfect (linear) correlation, -1 means perfect negative correlation, and 0 means a complete lack of correlation. Considering that the attention maps were matched using the Hungarian algorithm [33], such that the overall similarity is maximised, the values were expected to be above 0. The following results show that similarities between the model instances increased with increasing layer depth. They started off at considerably low mean correlation values (34%). This can be attributed to the fact that, in shallower parts of the network, the incentive or need for structural function is limited because the activations can still be recombined and their structure transformed significantly in deeper layers. The second hidden layer showed very similar values, implying a lack of internal convergence to any particular structure or composition of weights. While the minimum values stayed consistently low across all three layers, the output layer clearly showed an increase in *mean* correlation of up to 45.6% and a *maximum* cosine similarity of 78.1%. This was expected, given that both models optimised for the same loss function

$\mathcal{L}$, which is why the attention maps in the *output* layers, while remaining considerably varied, have shown an increase in structural similarity compared to hidden layers.

	Mean	Median	Maximum	Minimum
Hidden Layer "fc0"	0.340	0.341	0.564	0.078
Hidden Layer "fc1"	0.342	0.322	0.566	0.116
Output Layer "fc2"	0.456	0.495	0.781	0.075

Table 1: Cross-instance similarity of neuron attention maps per layer.

Resulting Similarities of Activations. The above values represent weight-level attention patterns and reveal strong variability across model instances. Following that, I tested for internal consistency on a higher level, namely in the activations themselves, i.e., the sum that the attention maps from the previous comparisons had represented. Specifically, I tested for numerical and positional structure across the same model instances, with the motivation of finding the presence or absence of structure that could potentially be useful for explainability. The table below summarises the findings. *Raw*, i.e., unsorted cosine similarity values are close to zero for both hidden layers, evidently representing the lack of structure in the positions of neurons. In contrast, the sorted similarity is close to 1, showing a remarkably strong correlation in the *values* that the neuron activations demonstrate. Thus, given the same inputs and model architecture, and independently initialised weights, the activations appear to lack consistent compositions of weights or positions in a layer, whereas the high correlation in sorted values implies consistency in the presence of certain activation values, regardless of their position or composition.

	Raw Cosine Similarity	Sorted Cosine Similarity
Hidden Layer "fc0"	0.066 ± 0.156	0.967 ± 0.031
Hidden Layer "fc1"	0.121 ± 0.201	0.931 ± 0.059

Table 2: Cross-instance similarity of neuron activations (mean ± standard deviation).

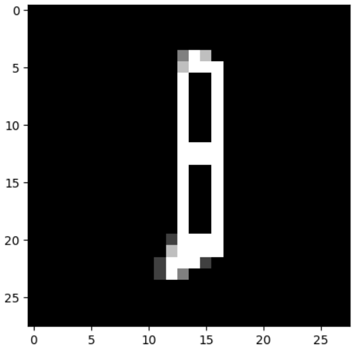
This partially aligns with the findings of Li et al., who showed that mean neuron activations vary *within* a network, while mean activation values *across* networks converge to a nearly identical distribution [37]. In my results, I also showed that the activation values themselves align across independently trained instances with $> 90\%$ correlation.

5.3 Testing Rule Approximation

Rationale. Previous experiments were intended to show the lack of meaningful patterns and the ambiguity of an FFN's representation of data, testing its internal parameters for interpretable structure. Although the prediction accuracy of the tested FFNs has reached 96.59% on testing data, the identified properties of FFNs can result in consequences beyond poor explainability, especially when given modified input data. This experiment aimed to demonstrate this by showing how correlation-based approximations of rules impair both performance and explainability. When inputs were perturbed but remained consistent with the rules defining a digit, a model that fits correlations may have failed to generalise to rules beyond dataset-specific patterns. In such cases, an FFN can produce predictions based on learnt patterns but ignore the underlying rules that generated these data patterns. Consequently, an explanation of the prediction would have to justify correlations by attempting to reconstruct what the rules may have been, rather than providing direct reasons and rules behind the model's decision.

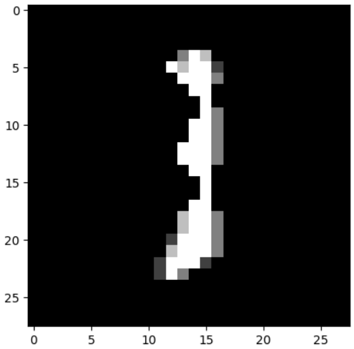
Results. By creating rule-preserving perturbations, I tested whether the FFN's learnt correlations still suffice for predicting the correct class. The four visualised examples clearly demonstrate that the model has made incorrect predictions. In particular, the FFN predicted that the input images are of class '1', which was expected due to the original image being of class '1'. The model has failed to change its prediction on the perturbed inputs. Although further variations of the input also produced predictions other than '1', the model consistently failed to predict a modified digit correctly and showed high entropy in the output scores, i.e., high uncertainty. More generally speaking, it failed to recognise that the changes made to the digit had transformed its class $x^{(i)}$ to $\hat{x}^{(i)}$. Below, examples of the perturbed inputs and the

FFN's predictions as pre-activations are shown.



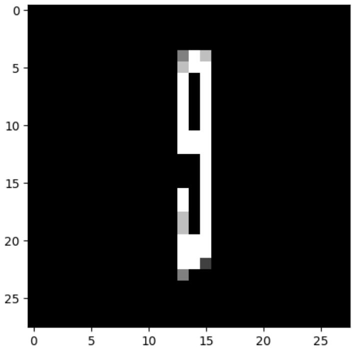
True Class: 8, Model prediction: 1

Class i:	0	1	2	3	4
p_i:	-3.14	7.22	-0.83	3.81	-2.07
Class i:	5	6	7	8	9
p_i:	-1.85	-1.53	-3.58	1.65	0.31



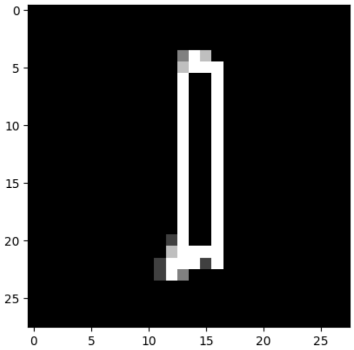
True Class: 3, Model prediction: 1

Class i:	0	1	2	3	4
p_i:	-2.54	7.84	0.99	3.62	-4.34
Class i:	5	6	7	8	9
p_i:	-3.50	-0.26	-2.37	0.10	-0.29



True Class: 9, Model prediction: 1

Class i:	0	1	2	3	4
p_i:	-3.30	8.74	-0.69	1.49	-4.52
Class i:	5	6	7	8	9
p_i:	-1.88	-1.20	-0.53	0.94	-1.65



True Class: 0, Model prediction: 1

Class i:	0	1	2	3	4
p_i:	-3.94	6.60	2.26	4.11	-2.38
Class i:	5	6	7	8	9
p_i:	-2.79	-3.22	-2.57	2.97	0.30

Aligning with my results, Kim Y. has argued that standard neural computation *only approximates* logic through statistical association, questioning whether it goes beyond [31]. In this experiment, I could demonstrate this *limitation* through the model's misclassification of rule-consistent inputs in practice.

6 Discussion

With this paper, I (Q1) ask what it means to explain a model's decision-making and (Q2) aim to identify the underlying causes that make such explanations inherently difficult or impractical. Furthermore, I discuss (Q3) why many models can perform considerably well despite the identified limitations, and (Q4) why a fully explainable model maintaining state-of-the-art performance would be invaluable for safely making use of the advantages of modern AI.

The theoretical arguments from Section 3.2 and the results from testing or demonstrating these theoretical arguments practically in Section 4.4 form the foundation for answering these research questions. Thus, in this section, the results of my experiments, their reconciliation with theory, and the implications for explainability are discussed.

6.1 Non-Compositional Representation

Theory and Experiments. Given the theoretical properties of an FFN's internal workings, I have assessed that, in its training process, a feedforward network has no incentive to learn any specific lower-level patterns, i.e., patterns that compose the higher-level concepts, namely, in the case of my experiments, the geometrical structure of a digit. In particular, the pre-activation sum of a neuron can be composed of any weight configuration, given a fixed input, that minimises the loss function. Experiment 5.1 showed that this lack of required structure does not result in purely random weights, but that there are focused areas where similar weights are clustered, gradually and smoothly changing across the resulting *attention map.*

Nonetheless, these clustered structures appear to have only little similarity across model instances, with a mean correlation of 34.0% in the hidden layers, implying a rather low and mostly coincidental correlation, since pairs of attention maps were chosen to maximise the overall similarity. In the output layer, the correlation reached a mean of 45.6%, which can be attributed to the fact that the output layer contains fewer possible activation sums that *directly* determine the correct class, while the activations of the hidden layers are only *indirectly* related to the output activations. Furthermore, there were no structurally meaningful or recognisable patterns in the attention maps that could be associated with features from the input digits or known concepts. These results show that, while there is a certain clustering behaviour in the weight distribution on the attention maps, there is no input-output relation that is visually meaningful or consistent across model instances in the summands of activation sums. However, testing the statistical properties of the activation sums themselves, instead of their summands, showed that there is, in fact, a strong consistency across model instances: the distribution of activation values remains remarkably consistent at $> 90\%$ correlation. That being said, while the position of an activation within a layer showed little to no consistency, sorting the values revealed that the same activation values were consistently present across model instances. This can be reconciled with the lack of consistent structure in the attention maps: because the weights from layer l are responsible for the positions of activations in layer $l + 1$, their lack of consistency across model instances led to inconsistent positional encoding in layers, meaning that the same activation sums are distributed differently.

This convergence of activation values across independently trained networks was also shown by Li et al., matching what I demonstrated due to the underlying architectural properties of my FFN shared with the CNN architecture used by Li et al. [37].

Implications for Explainability. While the activations have shown consistency, the weights, i.e., the parameters that create the relation between input and prediction, have shown a lack of structurally meaningful or consistent patterns beyond being optimised. This makes the goal of explainability, to understand the reason behind a specific relation of input and output, inaccessible. Although it is possible to visualise and interpret the input-output relations in weight matrices (e.g., saliency, LRP), the vast number of parameters forms functional but inconsistent compositions (across model instances), making explanations beyond interpretation difficult. Specifically, this is due to a lack of *compositionality*: human cognition tends to compose new concepts from known concepts, e.g., the structure of a digit can be explained using a priori knowledge [19, 30], such as what a line, curve, or loop is, or relations such as 'above, 'next to', or 'touching'.

Conversely, an FFN, while *composing* each layer from the previous layer's activations, does not generate predictions based on known or interpretable patterns but is limited to a mostly *arbitrary composition* of weights, as shown in Experiment 5.1 (34.0% correlation across models), making it infeasible for the human brain to understand their meaning. For reference, one activation in my *toy FFN* is composed of 784×32, 32×16, or 16×10 weights: numbers of weights that the human brain could not explain without finding meaningful patterns that are known a priori and consistently present across model instances. An analogy fitting this lack of compositional meaning can be considered: a sentence and its meaning represent an image and its label. Given multiple examples of such sentences and their meanings, interpretability can show which combinations of words share the same meaning without needing to know the individual words. Explainability would mean knowing each word and understanding how they can be used to create any sentence with that particular meaning. The ambiguity predicted and found in the weights of an FFN may not interfere with interpretability, but it does result in an explanation lacking the compositional nature of meaningful concepts, that is, the words of a sentence.

6.2 Correlation versus Causal Reasoning: The Core Limitation

Theory and experiments. Both my theoretical arguments and experiments conclude that, in addition to *compositionality*, the core constraint in explainability is the *causal* nature of human reasoning [19], which is lacking in neural networks. To demonstrate this lack and the limitations of purely correlation-based predictions in both the accuracy of the FFN and its explainability, Experiment 5.3 shows where the FFN falls short.

The results show that the FFN's accuracy can be attributed to the input's alignment with the dataset in *style*, rather than to strong generalisation: inputs perturbed to deviate from the original MNIST-style break the FFN's predictions. The *style* refers to the distribution behaviour of white pixels across a stroke, analogous to the line texture of a pen. The example inputs that were modified to clearly align with the definition of the digits '8', '3', '9', and '0' were intentionally designed to distribute white pixels ($x_i >> 0$) only in the area of the originally unmodified digit '1'. The model consistently predicting class '1' implies that it only correlates a class y with the presence of white pixels in certain input areas but has not learnt which underlying rules define the digit classes. Thus, it failed to recognise that the input structure was modified to not align with class '1'. This is a direct demonstration of weights failing to generalise beyond the scope of training data: instead of reasoning about whether a certain rule applies to the inputs, FFNs build highly nonlinear associations between input data and its labels. This results in the model's limited ability to generalise to variations of patterns that still follow a set of rules when these variations were not present in the training data.

Similarly, this absence of logical rules was found by Dhont, Wolfs, & Verhaegen, who showed that, given COVID-19 chest X-Rays, convolutional neural networks (CNNs) often rely on non-pathological features from outside the lungs, which falsely improve their accuracy scores [16]. My experiment 5.3 shows an analogous failure in which the model's performance evidently relies on irrelevant features, namely the MNIST-specific style in pixel distributions, which leads to inaccurate predictions when tested for recognising whether certain rules apply.

Implications for Explainability. I defined and concluded that the goal of an explanation is to understand *why* the weights have taken on certain values, rather than merely recognising *that* they have these values, which should be distinguished from *interpreting* the model. Not only does the approximation of rules through data patterns result in highly inefficient representations, with the number of parameters beyond intuitive interpretation, but it also lacks the causal grounds that explainability requires. At best, an explanation can recover the rules that originally caused the learnt patterns in the training data. However, there are many cases where a model learns useful patterns beyond human understanding. In such cases, the lack of causal grounds in a model is a major disadvantage. If, for instance, a model makes medical predictions that human clinicians do not understand, it is crucial for the model's predictions to be not only interpretable (making them verifiable by clinicians) but also explainable (allowing clinicians to recognise *why* the predictions were made).

A neural network fails to be explained on the grounds of correlation replacing causation: explainability does not face the problems of poor methodology or high model complexity, but rather that of pure correlation leading to limited generalisation. A trained neural network is an optimal set of parameters that represents *how* inputs relate to outputs in the training data. This does not require consideration of the rules that define *why* these relations were established, which can, depending on the dataset, lead to the existence of input-output relations that follow these rules but are not represented by the trained neural network's parameters.

For the explainability of a model, this means that the weights can only be explained when the rules that create the data patterns can be recovered manually. In many cases, however, this is not feasible when (i) the rules are not known or are too complex to be understood, or (ii) when the model represents incomplete or even misleading input-output relations that ultimately prevent a human interpreter from recovering *what* caused a prediction, thereby failing to explain the model.

6.3 Generalisation of Implications Beyond FFNs

To establish how the identified properties of FFNs that cause a lack in explainability and generalisability extend beyond simple FFNs, their roles in CNNs and transformers were discussed in Section 2.4. This section discusses how the limitations identified in this paper extend to modern architectures. It must again be noted that the following notion of *attention* in transformers is different from the one in *attention maps*, as introduced in my experiment 4.5.

While attention [59] can be visualised as weights between tokens, such visualisations remain limited to interpretability and do not reach explainability: the weights are trained on statistical properties and not on causal structure. Similarly to FFNs, attention lacks grounding in rules or concepts from real world data (e.g., visual, physical, auditory). Instead, it optimises correlations in data that *describe* these concepts but fails to understand or *explain* their relations logically. This absence of *causal inference* explains why large language models (LLMs) like GPT can generate naturally sounding but logically inconsistent texts. They learn patterns in how words describe rules, however, do not learn the rules themselves. Consequently, transformer models cannot be causally explained because their weights represent statistical associations between words, not the true meanings of the words as grounded physically, visually, auditorily, etc.

Kernels in CNNs, on the other hand, can often be visually associated with human-interpretable, and to some extent, meaningful features, such as lines or textures. However, they also represent only correlations between kernel weights and recurring input structures instead of causal relations. The final classification after the convolutional layers is, in fact, made by a fully connected feedforward network, using the preprocessed features and their activation maps to predict the image class *statistically*, not through causal reasoning.

This was further demonstrated in practice by Li et al., who showed that CNN kernels are not consistently aligned across model instances, similarly to the FFN considered in this research, demonstrating a lack of stable structure per model instance [37].

In the same way, virtually every modern architecture, albeit potentially powerful, does not go beyond *simulating* causal reasoning (e.g., large reasoning models) relying on vast amounts of data and immensely large numbers of parameters. The limitations of FFN's in modern architectures, as demonstrated in this research, are either directly inherited *by* the FFN or show analogous behaviour originating from the same fundamental architectural principles.

7 Conclusion and Outlook

My research was conducted with respect to the following research questions introduced in Section 1.5:

(Q1) What problems does the field of explainability concern itself with?

(Q2) Where do the limitations that contribute to these problems originate?

(Q3) Why do current neural architectures work as well as they do, despite these limitations?

(Q4) How could resolving these limitations be valuable?

In this paper, I answer Q1 (H1) by defining the explanation of a model's decision-making as the process of determining the causal origin of a model decision, enabling a human to *understand* a certain decision, *justify* it by verifying that it is logically coherent and aligned with human values, and trust the reliability of the model's future predictions - all of which would greatly benefit humanity (Q4, H4) by allowing the safe use of AI, such as fair legal decisions, reliable medical predictions, or safe self-driving cars, automated and facilitated by machine learning systems.

To this end, I demonstrate my established theoretical arguments regarding the limitations that FNNs impose on explainability through practical experiments on small feedforward networks (H2). I was able to show (Q2) that the weight composition of the model's pre-activations, represented through attention maps (5.1), reconciles with my theoretical prediction P1: since the loss $\mathcal{L}$ optimises only for prediction accuracy, allowing any optimal but structureless composition of weights, only the activation sums are consistent across model instances, while the weights lack meaningful or interpretable patterns and consistency. Through this lack of compositionality, FFNs are misaligned with human cognition, and existing explainability methods lack the compositional structure from which humans would benefit, namely, the composition of new concepts from concepts known a priori (Q2).

Additionally (Q2), I argue that these limitations are inherent to a model that performs correlation instead of causation, given that these are the differences between human cognition and machine intelligence. Correlation-based approximations of rules are at the core of failing explanations (P2): assuming that a set of rules $\mathcal{R}$ generates patterns (i.e., regularities and recurrences) in data X through a function $f(x, \mathcal{R})$, then an FFN approximates this function given samples of the generated data. Consequently, the FFN's prediction accuracy relies heavily on the integrity of the training data, whereas human cognition tends to rely on logically *inferring* rules $\mathcal{R}$, making their predictions significantly more aligned with the original set of rules $\mathcal{R}$ [19]. I could show that inputs following the rules that define a digit of class y_i were falsely classified by an FFN that relied on the distribution of white pixels in the input matrix instead of the rules. Specifically, the FFN learnt characteristics specific to the training data because it relied on correlation rather than applying the rules $\mathcal{R}$ causally (P2). A causal architecture more similar to human cognition would not only avoid such false predictions but also increase the explainability of predictions, providing a direct cause in the data or a rule from which a decision was inferred. While the dataset in my experiment was relatively small, this observed behaviour is not limited to small architectures but is recognisable in modern architectures, merely on different scales [11, 23, 53].

In conclusion, I was able to visually and empirically demonstrate the origins and consequences of an FFN's properties that make explainability a challenge inherent to neural network architectures, answering the central research question Q2 of my research. To answer research questions Q3 and Q4, the following sections discuss the implications for modern architectures applied in practice.

7.1 Why Current Architectures Perform Well Despite These Constraints

Despite the architectural constraints assessed in my research, there is no doubt that state-of-the-art models deserve not only critique but also recognition for having transformed modern day technology. Modern AI has been successfully applied to various domains, demonstrating great usefulness. The question arises: what (Q3) makes modern AI perform as well as it seemingly does, or at least be as useful as it is, despite the found limitations and their generalisation to modern architectures. Apart from being strongly optimised, modern architectures beyond simple FFNs, such as CNNs or transformers, rely on more than just a well-designed architecture, namely, *big data.* Modern Large Language Models (LLMs) often achieve the impression of very general reasoning capabilities and do not seem to be affected by the limitations identified in this paper. However, a major factor allowing such models to appear similar to human cognition is the vast amount of training data (H3). There have been several papers exposing the short-comings of those architectures [11, 23, 53] to reason at human-level intelligence, and the model's dependence on big data aligns with the findings and conclusions of this paper: LLMs like ChatGPT [45] or Grok [47] rely heavily on optimised methods to correlate immensely large amounts of data, represented as so-called tokens through *attention* [59], and do not truly *understand* or *reason* about the learnt data beyond statistical association.

While the above may sound like strong claims about state-of-the-art models, there are empirical demonstrations of this failure. In particular, the Abstraction and Reasoning Corpus for Artificial General Intelligence (ARC-AGI), a non-profit organisation concerned with defining and testing *artificial general intelligence*, has designed visual tasks specifically requiring causal and deliberate reasoning [11] to benchmark reasoning abilities, as discussed in this paper. ARC-AGI defines *general intelligence* as the ability to "efficiently acquire new skills outside of its training data." (F. Chollet, 2019) [10]. Reaching an accuracy of $\geq 85\%$ on their benchmark does not imply that a system has achieved the state of AGI, but failing to do so implies that a system has certainly *not* achieved general intelligence [11]. At present, the highest accuracy achieved within a certain efficiency threshold is 27.08%, while human participants achieve an average score of 66%, ranging up to 100% accuracy.

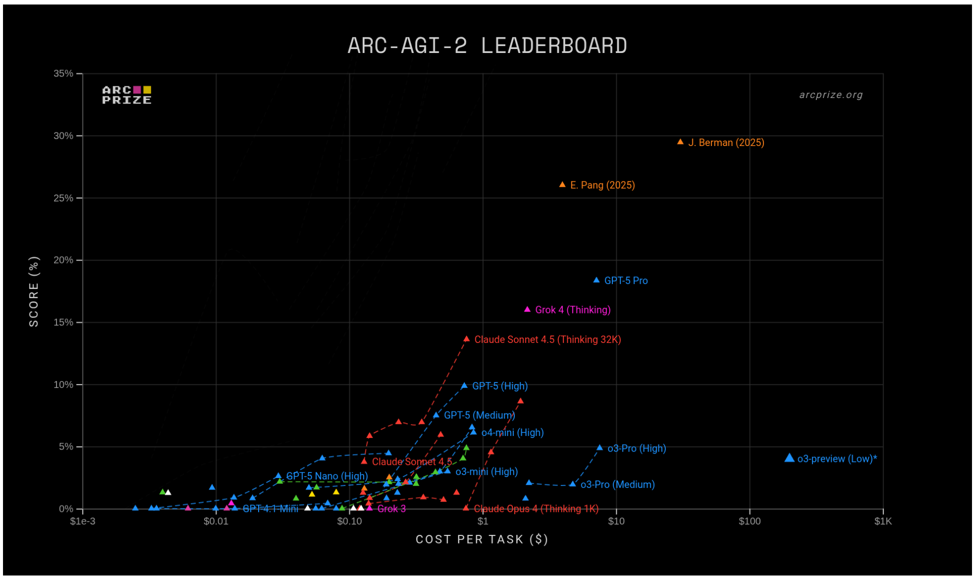


Figure 10: The official leaderboard of all public models on the ARC-AGI-2 competition [65].

In conclusion, this answers research question Q3: there are tasks in which modern systems appear to perform well if given enough data. While extremely useful in tasks that do not require general intelligence (e.g., solving mathematical equations, retrieving and summarising information, generating or classifying images from large datasets), these models drastically fail in tasks that require true causal reasoning, inference, or general intelligence.

This gap is not incidental. It is another consequence of the same architectural constraints that make explanations of a model's parameters misaligned with human cognition, as discussed in this paper. If a system only fits correlations without inferring underlying rules, it will (i) generalise poorly when the patterns vary beyond the dataset, (ii) be hard to explain in human terms because its internal parameters do not align with the compositional and causal nature of human cognition, and (iii) fail to learn from little data (e.g., ARC-AGI-2).

7.2 Why These Limitations Matter

When a model is applied in high-stakes scenarios, it is not enough to show *which* inputs were influential, but it also has to be justified *why* the model's internal decision-making is sound. For example, electrocardiograms (ECGs) are a timely example of a machine learning system that has been clinically deployed to process medical screenings and detect *reduced left ventricular ejection fraction* (LVEF), the percentage of blood pumped out of the heart's left ventricle per contraction [55]. Relying on the ECG's waveforms (voltage differences over time), the model can predict heart failure based on regularities that are not obvious to clinicians. If a clinician cannot understand the model's decisions, it cannot be assumed to be medically accurate or to not accidentally rely on screening artefacts, i.e., irrelevant information (H4).

Another prominent example of the importance of *justifying* model predictions is the previously mentioned work by Dhont, Wolfs, & Verhaegen, who showed that several CNNs deployed to detect COVID-19 on chest X-rays were not only relying on visual features from the lung [16]. Specifically, they used saliency maps to expose the models' reliance on meaningless pixels outside the lungs. When removing these pixels, the accuracy dropped significantly from $> 90\%$ to $< 70\%$.

In addition to medical predictions (Q4, H4), justifying the decision-making of a system becomes crucial in morally or legally critical applications. For instance, AI models have been deployed to predict the financial fines due for committed crimes [56], where it must be ensured that the model does not falsely rely on non-relevant features in the training data, such as the ethnicity of the offender.

Conclusively, the above examples provide an answer to research question Q4: if explanations cannot provide reason, failures may go undetected until exposed by severe consequences. For trustworthy use and more general forms of intelligence, models must be able to reason and justify their decisions. Explanations that target *why*, not only *what*, are therefore essential.

7.3 Toward Human-Like Explainability

While modern AI struggles to be explained at sufficient levels [41], recent research explores architectures with more modular and explainable internal processes, aiming to align more closely with human-like explainability without sacrificing too much performance. These *intrinsically interpretable* architectures do not currently achieve state-of-the-art performance or applicability independently, but they have been incorporated into modern architectures [43].

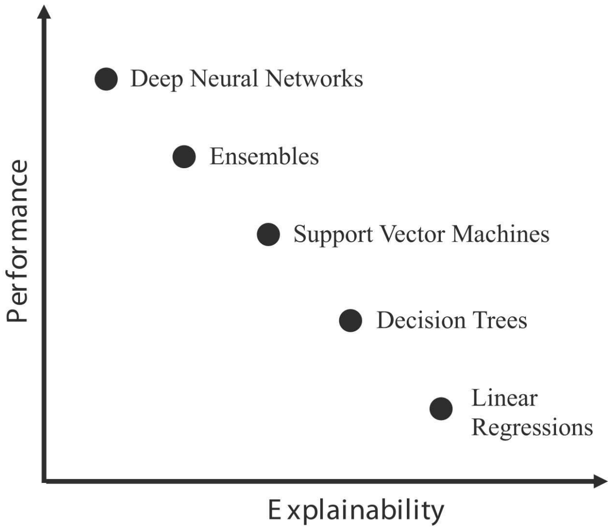


Figure 11: This plot shows different machine learning systems and how their performance and explainability appear to have a correlational relationship (reproduced from [66], CC BY 4.0).

Mixture of Experts (MoE). MoE is an architecture consisting of many small *experts* and a *router* that selects a few experts per input, making the computation *modular* [8]. Different inputs trigger different experts and explicitly route decisions, showing *which* experts handled *which* parts of an input. An experts' role is determined causally by testing their contribution to predictions. In practice, mechanisms such as top-k routing are used to ensure that the model does not rely solely on a few experts but instead chooses multiple experts evenly. This modular routing steers toward human-aligned compositionality without losing practicality. Similarly, **Neural Module Networks** (NMNs) [2] support compositionality by processing each input by wiring together a small set of reusable modules (e.g., `attend[dog]`, `re-attend[above]`, `classify[color]`) into a short program. The resulting execution graph is then a chain of steps ("find dog -> move above -> classify colour"). This aligns closely with how humans explain tasks in terms of concepts and their relations, and it supports causal structure as well as more sequential reasoning.

All together, these approaches trade some performance for *modularity* and a more *causal* nature. Without being recombined with other architectures, they do not yet dominate across state-of-the-art models in the industry but demonstrate a practical path toward explanations that align with human cognition.

7.4 Outlook for Future Research

The ability of modern neural networks to outperform clinicians in certain tasks comes from the availability of data sufficient to approximate complex and hidden rules more efficiently than humans can (Q3). While there are architectures, such as MoE [8], that aim to improve explainability, they still lack the performance that modern neural architectures demonstrate. Therefore, future work should focus on achieving more human-aligned architectures that can confidently and reliably infer rules while maintaining both *performance* and *explainability*. Despite the current trend of Large Language Models (LLMs) in the industry, the currently smaller fields concerned with explainable and causal architectures capable of novel reasoning may be those that bring artificially intelligent systems to new levels of more human-like intelligence in machines. In any event, the field of explainability will naturally gain importance as the legal requirements for AI companies to provide explanations of critical decisions become more urgent (M. El-Assady, personal communication, June 2025).

8 Personal Reflection

Designing and planning the experiments for this project proved more challenging than I had expected. It was difficult to keep a clear overview on the purpose of each experiment, to decide which to include and which to omit, and to avoid accumulating too many variations. In hindsight, I could have established the paper's structure earlier, as doing so would have helped in defining the research questions and saved time during experimental analyses.

That said, several choices worked well. Developing the work in IPython Notebooks (.ipynb) kept the process organised and reproducible. Using a toy network and a simplified task made the testing faster and the results easier to interpret. Most importantly, matching theory with the right experiments helped me demonstrate practical implications, clarify which investigations were genuinely useful, and ultimately strengthen the arguments of the paper.

I have learnt how essential clarity and structure are, and that planning and reflection can be as important as technical skill. Throughout the process of building, rethinking, and rebuilding, I had to critically assess my own results. This experience showed me that research is as much about asking the right questions as about finding answers.

References

[1] Allen-Zhu, Z., Li, Y., & Song, Z. (2018). *A Convergence Theory for Deep Learning via Over-Parameterization.* arXiv:1811.03962 (August 2025).

[2] Andreas, J., et al. (2017) *Neural Module Networks.* arXiv:1511.02799v4 (August 2025).

[3] Baars, B. J. (1988). *A Cognitive Theory of Consciousness.* Cambridge University Press. https://ccrg.cs.memphis.edu/assets/papers/1988/Baars-A%20Cognitive%20Theory%20of%20Consciousness.pdf (19.07.2025).

[4] Bach, S., et al. (2015). *On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation.* PLOS ONE, 10(7): e0130140. doi:10.1371/journal.pone.0130140 https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140 (11.10.2025).

[5] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* Microsoft. https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf (14.08.2025).

[6] Buçinca, Z., et al. (2020). *Proxy Tasks and Subjective Measures Can Be Misleading in Evaluating Explainable AI Systems.* arXiv:2001.08298 (August 2025).

[7] Chang, Y.-C. I. (2025). *A Survey: Potential Dimensionality Reduction Methods.* arXiv:2502.11036 (June 2025).

[8] Chen, Z., et al. (2022) *Towards Understanding Mixture of Experts in Deep Learning.* arXiv:2208.02813v (August 2025).

[9] Cheong, B. C. (2024). *Transparency and accountability in AI systems.* Frontiers in Human Dynamics. https://www.frontiersin.org/journals/human-dynamics/articles/10.3389/fhumd.2024.1421273/full (28.07.2025).

[10] Chollet, F. (2019). *On the Measure of Intelligence.* arXiv:1911.01547 (August 2025).

[11] Chollet, F., et al. (2025). *ARC-AGI-2: A New Challenge for Frontier AI Reasoning Systems.* arXiv:2505.11831 (August 2025).

[12] Colin, J., et al. (2022). *What I cannot predict, I do not understand: A human-centered evaluation framework for explainability methods.* arXiv:2112.04417 (August 2025).

[13] Damirchi, H., et al. (2023). *Independent Modular Networks.* arXiv:2306.01316 (August 2025).

[14] Danks, D., & Davis, I. (2023) *Causal inference in cognitive neuroscience.* WIREs. https://doi.org/10.1002/wcs.1650 (11.10.2025).

[15] Das, A., & Rad, P. (2020). *Opportunities and Challenges in Explainable Artificial Intelligence (XAI): A Survey.* arXiv:2006.11371 (July 2025).

[16] Dhont, S., Wolfs, T., & Verhaegen, J. (2021). *Automatic coronavirus disease 2019 diagnosis based on chest radiography and deep learning – Success story or dataset bias?.* https://doi.org/10.1002/mp.15419 (11.10.2025).

[17] Dumitru, E. (2009). *Visualizing Higher-Layer Features of a Deep Network* https://www.researchgate.net/publication/265022827_Visualizing_Higher-Layer_Features_of_a_Deep_Network (19.07.2025).

[18] Erhan, D., Courville, A., & Bengio, Y. (2010). *Understanding representations learned in deep architectures.* Technical Report, Université de Montréal. https://www.dumitru.ca/files/publications/invariances_techreport.pdf (19.07.2025).

[19] Felin, T., & Holweg, M. (2024). *Theory Is All You Need: AI, Human Cognition, and Causal Reasoning.* SSRN. https://pubsonline.informs.org/doi/10.1287/stsc.2024.0189 (14.08.2025).

[20] Gerber, I. (2025). *Attention Is Not All You Need: The Importance of Feedforward Networks in Transformer Models.* arXiv:2505.06633 (August 2025).

[21] Glorot, X., & Bengio, Y. (2010). *Understanding the difficulty of training deep feedforward neural networks.* Proceedings of Machine Learning Research. `https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf` (19.07.2025).

[22] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press. `https://www.deeplearningbook.org/front_matter.pdf` (21.04.2025).

[23] Gu, Y., et al. (2025). *The Illusion of Readiness: Stress Testing Large Frontier Models on Multimodal Medical Benchmarks.* arXiv:2509.18234 (August 2025).

[24] Harris, C. R., et al. (2020). *Array Programming with NumPy.* Nature, 585, 357–362. arXiv:2006.10256 (August 2025).

[25] Hassija, V., et al. (2023). *Interpreting Black-Box Models: A Review on Explainable Artificial Intelligence.* *Cognitive Computation*, 16, 45–74. DOI:10.1007/s12559-023-10179-8 (August 2025).

[26] He, K., et al. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.* arXiv:1502.01852 (August 2025).

[27] Hinton, G. E. (1991). *Connectionist learning procedures.* `https://doi.org/10.1016/0004-3702(89)90049-0` (August 2025).

[28] Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment.* Computing in Science & Engineering, 9(3), 90–95. doi:10.1109/MCSE.2007.55 `https://www.researchgate.net/publication/3422921_Matplotlib_A_2D_Graphics_Environment` (11.10.2025).

[29] Kares, F., et al. (2025). *What Makes for a Good Saliency Map? Comparing Strategies for Evaluating Saliency Maps in Explainable AI (XAI).* arXiv:2504.17023 (June 2025).

[30] Kaufman, R. A., & Kirsh, D. J. (2022). *Cognitive Differences in Human and AI Explanation.* Department of Cognitive Science, University of California, San Diego. Proceedings of the Annual Meeting of the Cognitive Science Society, 44(44) `https://escholarship.org/uc/item/9p24077n` (14.08.2025).

[31] Kim, Y. (2025). *Standard Neural Computation Alone Is Insufficient for Logical Intelligence.* arXiv:2502.02135 (October 2025).

[32] Kingma, D. P., & Ba, J. L. (2014). *Adam: A Method for Stochastic Optimization.* arXiv:1412.6980 (June 2025).

[33] Kuhn, H. W. (1955). *The Hungarian Method for the Assignment Problem.* Naval Research Logistics Quarterly, 2(1–2), 83–97. `https://web.eecs.umich.edu/~pettie/matching/Kuhn-hungarian-assignment.pdf` (11.10.2025).

[34] LeCun, Y. (2022). *A path towards autonomous machine intelligence* (Version 0.9.2, June 27). Courant Institute of Mathematical Sciences, New York University; Meta - Fundamental AI Research. `https://openreview.net/pdf?id=BZ5a1r-kVsf` (14.08.2025).

[35] LeCun, Y., & Bengio, Y. (1997). *Convolutional Networks for Images, Speech, and Time-Series.* `https://www.researchgate.net/publication/2453996_Convolutional_Networks_for_Images_Speech_and_Time-Series` (August 2025).

[36] LeCun, Y., & Cortes, C. (1998). *The MNIST database of handwritten digits.* The dataset used in this paper. `http://yann.lecun.com/exdb/mnist/` (19.07.2025).

[37] Li, Y., Yosinski, J., Clune, J., Lipson, H., & Hopcroft, J. E. (2016). *Convergent Learning: Do Different Neural Networks Learn the Same Representations?* Published as a conference paper at ICLR 2016. arXiv:1511.07543 (October 2025).

[38] Magai, G., & Ayzenberg, A. (2022). *Topology and geometry of data manifold in deep learning.* arXiv:2204.08624 (August 2025).

[39] Marcinkevičs, R., & Vogt, J. E. (2020). *Interpretability and Explainability: A Machine Learning Zoo Mini-tour.* arXiv:2012.01805 (June 2025).

[40] Melas-Kyriazi, L. (2021). *Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprisingly Well on ImageNet.* arXiv:2105.02723 (August 2025).

[41] Mersha, M., et al. (2024). *Explainable Artificial Intelligence: A Survey of Needs, Techniques, Applications, and Future Direction.* arXiv:2409.00265 (July 2025).

[42] Molnar, C. (2019). *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.* Leanpub. `https://originalstatic.aminer.cn/misc/pdf/Molnar-interpretable-machine-learning_compressed.pdf` (19.07.2025).

[43] Mu, S., & Lin, S. (2025). *A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory, and Applications.* arXiv:2503.07137v3 (June 2025).

[44] Nair, V., & Hinton, G. E. (2010). *Rectified Linear Units Improve Restricted Boltzmann Machines.* `https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf` (May 2025).

[45] OpenAI; Achiam, J., et al. (2023). *GPT-4 Technical Report.* arXiv:2303.08774 (August 2025).

[46] Paszke, A., et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* arXiv:1912.01703 (August 2025).

[47] Perplexity AI, UNU. (2025). *The Capabilities and Limitations of Grok 3 in the Landscape of Advanced Reasoning Models.* Technical Report. `https://c3.unu.edu/wp-content/uploads/2025/02/PerplexityAI_report2.pdf` (11.10.2025).

[48] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). *"Why should I trust you?": Explaining the predictions of any classifier.* arXiv:1602.04938 (August 2025).

[49] Rosenblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain.* Psychological Review, 65(6), 386–408. Cornell Aeronautical Laboratory. `https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf` (19.07.2025).

[50] Ruder, S. (2016). *An Overview of Gradient Descent Optimization Algorithms.* arXiv:1609.04747 (May 2025).

[51] Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Pearson. `https://people.engr.tamu.edu/guni/csce625/slides/AI.pdf` (21.04.2025).

[52] Saeed, W., & Omlin, C. (2023). *Explainable AI (XAI): A Systematic Meta-Survey of Current Challenges and Future Opportunities.* `https://doi.org/10.1016/j.knosys.2023.110273` (21.04.2025).

[53] Shojaee, P., et al. (2023). *The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity.* Apple. arXiv:2506.06941 (August 2025).

[54] Signa, A., et al. (2021). *Cognitive Robots and the Conscious Mind: A Review of the Global Workspace Theory.* Current Robotics Reports, 2(1), 12–27. `https://link.springer.com/article/10.1007/s43154-021-00044-7` (19.07.2025).

[55] Lopez-Jimenez, F. (2025). *A multicenter pragmatic implementation study of AI-ECG-based clinical decision support software to identify low LVEF: Clinical trial design and methods.* American Heart Journal Plus: Cardiology Research and Practice, Volume 54. `https://doi.org/10.1016/j.ahjo.2025.100528` (October 2025).

[56] Tahura, U. S., & Selvadurai, N. (2025). *The Use of Artificial Intelligence in Judicial Decision-Making: The Example of China.* `https://ijlet.org/wp-content/uploads/2025/01/2.3.1.pdf` (10.10.2025).

[57] Touvron, H., et al. (2021). *ResMLP: Feedforward Networks for Image Classification with Data-Efficient Training.* arXiv:2105.03404 (August 2025).

[58] VanRullen, R., & Kanai, R. (2020). *Deep Learning and the Global Workspace Theory.* arXiv:2012.10390 (April 2025).

[59] Vaswani, A., et al. (2017). *Attention Is All You Need.* arXiv:1706.03762 (September 2025).

[60] Wu, M., et al. (2023). *VERIX: Towards verified explainability of deep neural networks.* arXiv:2212.01051 (June 2025).

[61] Y. LeCun. "Before we reach Human-Level AI (HLAI), we will have to reach Cat-Level & Dog-Level AI." Tweet from @ylecun, X (formerly Twitter), February 5, 2023. `https://x.com/ylecun/status/1622300311573651458` (September 2025).

[62] Vlassis, N. N., Sun, W., & De Lorenzis, L. (2020). *Geometric Deep Learning for Computational Mechanics Part I: Anisotropic Hyperelasticity. arXiv preprint arXiv:2001.04292.* https://doi.org/10.48550/arXiv.2001.04292 (12.10.2025). Licensed under CC BY 4.0.

[63] Du, J. (2019). *The Frontier of SGD and Its Variants in Machine Learning. Journal of Physics: Conference Series, 1229(1)*, 012046. https://iopscience.iop.org/article/10.1088/1742-6596/1229/1/012046 (12.10.2025). Licensed under CC BY 3.0.

[64] Zhang, L., Wu, Q., & Zhao, X. (2021). *Accelerating Privacy-Preserving Momentum Federated Learning for Industrial Cyber-Physical Systems. Complex & Intelligent Systems, 7*, 3289–3301. https://doi.org/10.1007/s40747-021-00519-2 (12.10.2025). Open access (CC BY 4.0).

[65] ARC Prize. (2025). *Leaderboard.* Available at: https://arcprize.org/leaderboard (12.10.2025).

[66] Herm, L.-V., et al. (2021). *Stop Ordering Machine Learning Algorithms by Their Explainability! A User-Centered Investigation of Performance and Explainability. International Journal of Information Management, 67*, 102538. https://doi.org/10.1016/j.ijinfomgt.2022.102538 (12.10.2025). Licensed under CC BY 4.0.

List of Figures

All figures were either produced by me and extracted from the programs I have written, or have been attributed to cited papers with open access licenses. The produced figures can be viewed or reproduced as cell outputs in the GitHub repository.

List of Tables

Glossary

Activation function: Nonlinear function applied to the output of a neuron.

Attention map/composition: The matrix of a neuron's activation sum's summands, i.e., the weight×input products, traced back to the input.

Backpropagation: An algorithm that iteratively updates parameters to minimise a loss function.

Causation: The process of one property of data causing or leading to another.

Compositionality: The quality of something being composed of smaller, recognisable components.

Consistent structure: Recognisable or interpretable structure in visual patterns, consistent *across model instances.*

Correlation: The extent to which the same behaviour applies to two separate datasets and the probability that there is causation.

Feedforward Neural Network (FFN): An architecture with information flow in one direction and weighted recombinations of features.

Features: Properties of the input, i.e., in this research, one pixel or combinations of multiple pixels.

High-level / low-level concepts: A higher-level concept consists of lower-level concepts. The *lower* a low-level concept is, the harder it is to find other concepts composing it.

Model: A neural network instance with a specific architecture.

Model instance: One set of parameters in which the model differs from other model instances.

Neuron: A computational unit that transforms inputs via an activation function.

Optical character recognition (OCR): The automatic extraction of textual data from images.

Shallow/Deep layers The depth of a layer is its index in the network, i.e., layers closer to the output layer are deeper. The depth of a neural network is defined by the number of layers.

top-k search: A mechanism choosing the k elements from a set with the highest scores, used to find only the most relevant elements based on their computed importance score.

$\sigma(\cdot)$: A symbol for an activation function applied to a neuron's input (pre-activation).

Appendix A

Note: Mathematical definitions are included in the main paper. The code below reproduces my experiments and visualisations. Notebook cell outputs are omitted. Note that baseline methods are not included here because they are considered only mathematically, although some appear in the public GitHub repository. Because the original code was structured in IPython notebook cells, it may be more suitable to view the code in the GitHub repository. In the appendix, the code cells are indicated through comments. The appendix includes all programs discussed in the paper, with some functions omitted from the main text.

Project Repository

All code, experiments, and visualisations are available at:
`https://github.com/kns357/ANN_MA_Kaito.git`

Experiment Code

Listing 1: **FFN Setup**

```
#===========================================CODE CELL==========================================

import torch
import torch.nn as nn
import torchvision.datasets as datasets
import torchvision.transforms as transforms

#===========================================CODE CELL==========================================

input_size = 784
hidden_size0 = 32
hidden_size1 = 16
out_size = 10

epochs = 10
batch_size = 64
learning_rate = 0.001

#===========================================CODE CELL==========================================

train_dataset = datasets.MNIST(
    root='./data', train=True,
    transform=transforms. ToTensor(),
    download=True
)

test_dataset = datasets.MNIST(
    root='./data',
    train=False,
    transform=transforms.ToTensor()
)

train_loader = torch.utils.data.DataLoader(
    dataset=train_dataset,
    batch_size=batch_size,
    shuffle=True
)

test_loader = torch.utils.data.DataLoader(
    dataset=test_dataset,
    batch_size=batch_size,
    shuffle=False
)

```

```python
#===========================================CODE CELL===========================================

class Net(nn.Module):
    def __init__(self, input_size, hidden_size0, hidden_size1, out_size):
        super(Net, self).__init__()
        self.fc0 = nn.Linear(input_size, hidden_size0)
        self.fc1 = nn.Linear(hidden_size0, hidden_size1)
        self.fc2 = nn.Linear(hidden_size1, out_size)
        self.tanh = nn.Tanh()
        self.init_weights()

    def init_weights(self):
        nn.init.xavier_uniform_(self.fc0.weight, gain=nn.init.calculate_gain('tanh'))
        nn.init.xavier_uniform_(self.fc1.weight, gain=nn.init.calculate_gain('tanh'))
        nn.init.xavier_uniform_(self.fc2.weight, gain=nn.init.calculate_gain('tanh'))

    def forward(self, x):
        out = self.tanh(self.fc0(x))
        out = self.tanh(self.fc1(out))
        out = self.fc2(out)
        return out

#===========================================CODE CELL===========================================

net = Net(input_size, hidden_size0, hidden_size1, out_size)
CUDA = torch.cuda.is_available()
if CUDA:
    net = net.cuda()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

#===========================================CODE CELL===========================================

for epoch in range(epochs):
    correct_train = 0
    running_loss = 0

    for i, (images, labels) in enumerate(train_loader):
        images = images.view(-1, 28*28)
        if CUDA:
            images = images.cuda()
            labels = labels.cuda()

        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        correct_train += (predicted == labels).sum()
        loss = criterion(outputs, labels)
        running_loss += loss.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print('Epoch [{}/{}], Training Loss: {:.3f}, Training Accuracy: {:.3f}%'.format
          (epoch+1, epochs, running_loss/len(train_loader), (100*correct_train.double()/len(train_dataset))))
print("DONE TRAINING!")

torch.save(net.state_dict(), f'02_ffn_mnist_full.pth')

#===========================================CODE CELL===========================================

with torch.no_grad():
```

```
correct = 0

for images, labels in test_loader:
    if CUDA:
        images = images.cuda()
        labels = labels.cuda()

    images = images.view(-1, 28*28)
    outputs = net(images)
    _, predicted = torch.max(outputs.data, 1)
    correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: {} %'.format(100 * correct / len(
↪ test_dataset)))
```

Listing 2: **Neuron Attention**

```
#===========================================CODE CELL===========================================

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import random as ra
import math

#===========================================CODE CELL===========================================

input_size = 784
hidden_size0 = 32
hidden_size1 = 16
out_size = 10
epochs = 10
batch_size = 64
learning_rate = 0.001

#===========================================CODE CELL===========================================

test_dataset = datasets.MNIST(root='./data',
                        train=False,
                        transform=transforms.ToTensor())

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                      batch_size=batch_size,
                                      shuffle=False)

#===========================================CODE CELL===========================================

class Net(nn.Module):
    def __init__(self, input_size, hidden_size0, hidden_size1, out_size):
        super(Net, self).__init__()
        self.fc0 = nn.Linear(input_size, hidden_size0) #784 - 32
        self.fc1 = nn.Linear(hidden_size0, hidden_size1) #32 - 16
        self.fc2 = nn.Linear(hidden_size1, out_size)  #16 - 10
        self.tanh = nn.Tanh()
        self.init_weights()

    def init_weights(self):
        nn.init.xavier_uniform_(self.fc0.weight, gain=nn.init.calculate_gain('tanh'))
        nn.init.xavier_uniform_(self.fc1.weight, gain=nn.init.calculate_gain('tanh'))
        nn.init.xavier_uniform_(self.fc2.weight, gain=nn.init.calculate_gain('tanh'))

    def forward(self, x):
        out = self.tanh(self.fc0(x))
        out = self.tanh(self.fc1(out))
        out = self.fc2(out)
        return out

#===========================================CODE CELL===========================================

net = Net(input_size, hidden_size0, hidden_size1, out_size)
CUDA = torch.cuda.is_available()
if CUDA:
    net = net.cuda()

net.load_state_dict(torch.load("02_ffn_mnist_full.pth")) #loading one model instance

```

```
#===========================================CODE CELL===========================================

def compute_final_contributions(net, activations_fc0, target): #this function was omitted for
↪ the paper
    w1 = net.fc1.weight.detach().cpu()
    w2 = net.fc2.weight.detach().cpu()

    contribs = torch.zeros(activations_fc0.shape[0])

    for i in range(activations_fc0.shape[0]):
        a0 = activations_fc0[i].item()
        total = 0
        for j in range(w1.shape[0]):
            w0 = w1[j, i].item()
            a1 = torch.tanh(torch.tensor(a0 * w0))
            w1_2 = w2[target, j].item()
            total += a1.item() * w1_2
        contribs[i] = total

    return contribs

#===========================================CODE CELL===========================================

inputs, labels = next(iter(test_loader))
dig_to_check = 8 #to find random examples of the digit 8

#search wanted digit:
while True:
    ind = ra.randrange(0, len(inputs))
    if labels[ind] == dig_to_check:
        img = inputs[ind].squeeze()
        break
img = torch.full((28, 28), 255).float() #this can be commented out to process the actual input
↪ of an 8, for neuron attention the image becomes fully white (255)

output = net(img.view(-1).cuda())
print(f"Prediction: {output.argmax()}")
plt.imshow(img, cmap="inferno")

img_flat = img.view(-1)
weights = net.fc0.weight.detach().cpu()
biases = net.fc0.bias.detach().cpu()
contribs = weights * img_flat #[32, 784]

#normalise to [0, 1]
contribs_imgs = (contribs / contribs.max()).view(-1, 28, 28)

mask = torch.where(
    contribs_imgs == 0,
    torch.tensor(0.0, device=contribs_imgs.device),
    torch.tensor(1.0, device=contribs_imgs.device)
) #omitted for paper

contribs_imgs += biases[:, None, None]
#contribs_imgs *= mask #remove bias where background = 0 for visualisation

activations = torch.tanh(net.fc0(img.view(1, -1).cuda())).squeeze()
final_contribution = [round(con.item(), 2) for con in compute_final_contributions(net,
↪ activations, dig_to_check)] #omitted for paper

#plotting the attention maps:
fig, axes = plt.subplots(4, 8, figsize=(16, 8))
vmax = contribs_imgs.max().item()
```

```python

for i, ax in enumerate(axes.flat):
    im = ax.imshow(contribs_imgs[i], cmap='berlin', vmin=-vmax, vmax=vmax)
    ax.set_title(f'Neuron {i}', fontsize=25) #\nActivation: {round(activations[i].item(), 3)}\nContrib to class {dig_to_check}: {final_contribution[i]}', fontsize=8)
    ax.axis('off')

fig.suptitle('Pixel Importance to all 32 hidden neurons after fc0', fontsize=16)
cbar_ax = fig.add_axes([0.9, 0.15, 0.015, 0.4])
fig.colorbar(im, cax=cbar_ax)
plt.tight_layout(rect=[0, 0, 0.9, 1])
plt.show()

#============================================CODE CELL============================================
#repeating the above processes for a new model instance
net1 = Net(input_size, hidden_size0, hidden_size1, out_size)
CUDA = torch.cuda.is_available()
if CUDA:
    net1 = net1.cuda()

net1.load_state_dict(torch.load("stored_model_weights/model_inst_1")) #loading a second model instance for comparison

#============================================CODE CELL============================================

inputs, labels = next(iter(test_loader))
dig_to_check = 8

#search wanted digit:
while True:
    ind = ra.randrange(0, len(inputs))
    if labels[ind] == dig_to_check:
        img = inputs[ind].squeeze()
        break
img = torch.full((28, 28), 255).float()

output = net1(img.view(-1).cuda())
#print(f"Prediction: {output.argmax()}")
#plt.imshow(img, cmap="inferno")

img_flat = img.view(-1)
weights = net1.fc0.weight.detach().cpu()
biases = net1.fc0.bias.detach().cpu()
contribs = weights * img_flat #[32, 784]

#normalise to [0, 1]
contribs_imgs1 = (contribs / contribs.max()).view(-1, 28, 28)

mask = torch.where(
    contribs_imgs1 == 0,
    torch.tensor(0.0, device=contribs_imgs1.device),
    torch.tensor(1.0, device=contribs_imgs1.device)
)

contribs_imgs1 += biases[:, None, None]
#contribs_imgs1 *= mask #remove bias where background = 0 for visualisation

activations = torch.tanh(net1.fc0(img.view(1, -1).cuda())).squeeze()
final_contribution = [round(con.item(), 2) for con in compute_final_contributions(net1, activations, dig_to_check)]

fig, axes = plt.subplots(4, 8, figsize=(32, 16))
vmax = contribs_imgs1.max().item()
```

```python
for i, ax in enumerate(axes.flat):
    im = ax.imshow(contribs_imgs1[i], cmap='berlin', vmin=-vmax, vmax=vmax)
    #ax.set_title(f'Neuron {i}, fontsize=25) #\nActivation: {round(activations[i].item(), 3)}', fontsize=20) #\nContrib to class {dig_to_check}: {final_contribution[i]}', fontsize=20)
    ax.axis('off')

fig.suptitle('Attention of 32 neurons in hidden layer fc0', fontsize=26)
cbar_ax = fig.add_axes([1, 0.075, 0.025, 0.8])
fig.colorbar(im, cax=cbar_ax)
cbar = fig.colorbar(im, cax=cbar_ax)
cbar.ax.tick_params(labelsize=20)

plt.tight_layout(rect=[0, 0, 0, 0])
plt.show()

#==========================================CODE CELL==========================================

import torch.nn.functional as F
from scipy.optimize import linear_sum_assignment

def normalize_map(t):
    return (t - t.mean()) / (t.std() + 1e-8)

def similarity_score(map1, map2): # calculating the similarity score between attention maps of any two models
    a = normalize_map(map1).view(-1)
    b = normalize_map(map2).view(-1)
    #return torch.norm(a - b, p=2).item()
    return F.cosine_similarity(a.unsqueeze(0), b.unsqueeze(0)).item()


mask = (img != 0)
mask_flat = mask.flatten()

flat_imgs = contribs_imgs.view(contribs_imgs.shape[0], -1)
flat_imgs1 = contribs_imgs1.view(contribs_imgs1.shape[0], -1)

masked_imgs = flat_imgs[:, mask_flat]
masked_imgs1 = flat_imgs1[:, mask_flat]

sim_matrix = torch.zeros((32, 32))

for i in range(32):
    for j in range(32):
        sim_matrix[i, j] = similarity_score(masked_imgs[i], masked_imgs1[j])

cost = -sim_matrix.numpy()
row_ind, col_ind = linear_sum_assignment(cost) #Hungarian algorithm
max_vals = sim_matrix[row_ind, col_ind].tolist()

pairs = list(zip(row_ind, col_ind, max_vals))
pairs.sort(key=lambda x: x[2], reverse=True)

#calculate mean:
sum_sims = 0
sims = []
for i, (r, c, sim) in enumerate(pairs):
    sum_sims += sim
    sims.append(sim)
```

```
plt.figure(figsize=(10, 160))

for i, (r, c, sim) in enumerate(pairs):
    plt.subplot(32, 2, i*2+1)
    plt.imshow(contribs_imgs[r], cmap="berlin", vmin=-1, vmax=1)
    plt.title(f"Similarity: {round(sim, 2)}")
    plt.subplot(32, 2, i*2+2)
    plt.imshow(contribs_imgs1[c], cmap="berlin", vmin=-1, vmax=1)
    plt.colorbar()

    if i > 4:
        break


sum_mean = sum_sims / len(pairs)
sum_med = float(torch.median(torch.Tensor(sims)))
sum_max = max(sims)
sum_min = min(sims)

sum_mean, sum_med, sum_max, sum_min

#===========================================CODE CELL==========================================
#repeating the above for deeper layers (above was fc0, following are fc1, fc2)
weights_fc1 = net.fc1.weight.detach().cpu().unsqueeze(-1).unsqueeze(-1)
biases_fc1 = net.fc1.bias.detach().cpu()

combined = (weights_fc1 * contribs_imgs.unsqueeze(0)).sum(dim=1)
max_val = combined.max()
combined = combined / max_val

mask = torch.where(
    combined == 0,
    torch.tensor(0.0, device=combined.device),
    torch.tensor(1.0, device=combined.device)
)

combined += biases_fc1[:, None, None]
#combined *= mask

combined_copy = combined.clone()

per_img_max = combined.view(len(combined), -1).max(dim=1)[0]
thresholds = (0.25 * per_img_max).view(len(combined), 1, 1)

combined = torch.where(
    combined == 0,
    torch.tensor(0.0, device=combined.device),
    torch.where(
        torch.abs(combined) > thresholds,
        combined,
        torch.tensor(0.0, device=combined.device)
    )
)
vmax = combined.max().item()

weights_fc2 = net.fc2.weight.detach().cpu()
activations = [torch.tanh(mat.sum()).squeeze() for mat in combined_copy]

contribs_out_class = [round(activations[idx].item() * weights_fc2[dig_to_check, idx].item(), 2)
↪ for idx in range(16)]

fig, axes = plt.subplots(4, 4, figsize=(12, 12))
for idx, ax in enumerate(axes.flat):
    im = ax.imshow(combined[idx].cpu(), cmap='berlin', vmin=-vmax, vmax=vmax)
    ax.axis('off')
    ax.set_title(f'Neuron {idx}', fontsize=25) #\nActivation: {round(activations[idx].item(), 2)}\nContrib. to class {dig_to_check}: {contribs_out_class[idx]}')

cbar_ax = fig.add_axes([1, 0.15, 0.015, 0.4])
fig.colorbar(im, cax=cbar_ax)

plt.tight_layout()
plt.show()

#==========================================CODE CELL==========================================

weights_fc1 = net1.fc1.weight.detach().cpu().unsqueeze(-1).unsqueeze(-1)
biases_fc1 = net1.fc1.bias.detach().cpu()

combined1 = (weights_fc1 * contribs_imgs.unsqueeze(0)).sum(dim=1)
max_val = combined1.max()
combined1 = combined1 / max_val

mask = torch.where(
    combined1 == 0,
    torch.tensor(0.0, device=combined1.device),
    torch.tensor(1.0, device=combined1.device)
)

combined1 += biases_fc1[:, None, None]
#combined1 *= mask

combined1_copy = combined1.clone()

per_img_max = combined1.view(len(combined1), -1).max(dim=1)[0]
thresholds = (0.25 * per_img_max).view(len(combined1), 1, 1)

combined1 = torch.where(
    combined1 == 0,
    torch.tensor(0.0, device=combined1.device),
    torch.where(
        torch.abs(combined1) > thresholds,
        combined1,
        torch.tensor(0.0, device=combined1.device)
    )
)

vmax = combined1.max().item()

weights_fc2 = net1.fc2.weight.detach().cpu()
activations = [torch.tanh(mat.sum()).squeeze() for mat in combined1_copy]

contribs_out_class = [round(activations[idx].item() * weights_fc2[dig_to_check, idx].item(), 2) for idx in range(16)]

fig, axes = plt.subplots(4, 4, figsize=(12, 12))
for idx, ax in enumerate(axes.flat):
    im = ax.imshow(combined1[idx].cpu(), cmap='berlin', vmin=-vmax, vmax=vmax)
    ax.axis('off')
    ax.set_title(f'Neuron {idx}', fontsize=20) #\nActivation: {round(activations[idx].item(), 2)}\nContrib. to class {dig_to_check}: {contribs_out_class[idx]}')

cbar_ax = fig.add_axes([1, 0.075, 0.05, 0.8])
fig.colorbar(im, cax=cbar_ax)
fig.suptitle('Attention of 16 neurons in hidden layer fc1', fontsize=26)
cbar = fig.colorbar(im, cax=cbar_ax)
```

```
cbar.ax.tick_params(labelsize=20)
plt.tight_layout(rect=[0, 0, 1, 0.98])
plt.show()

#==========================================CODE CELL==========================================

import torch.nn.functional as F
from scipy.optimize import linear_sum_assignment

def normalize_map(t):
    return (t - t.mean()) / (t.std() + 1e-8)

def similarity_score(map1, map2):
    a = normalize_map(map1).view(-1)
    b = normalize_map(map2).view(-1)
    #return torch.norm(a - b, p=2).item()
    return F.cosine_similarity(a.unsqueeze(0), b.unsqueeze(0)).item()


mask = (img != 0)
mask_flat = mask.flatten()

flat_imgs = combined.view(combined.shape[0], -1)
flat_imgs1 = combined1.view(combined1.shape[0], -1)

masked_imgs = flat_imgs[:, mask_flat]
masked_imgs1 = flat_imgs1[:, mask_flat]

sim_matrix = torch.zeros((16, 16))

for i in range(16):
    for j in range(16):
        sim_matrix[i, j] = similarity_score(masked_imgs[i], masked_imgs1[j])

cost = -sim_matrix.numpy()
row_ind, col_ind = linear_sum_assignment(cost)
max_vals = sim_matrix[row_ind, col_ind].tolist()

pairs = list(zip(row_ind, col_ind, max_vals))
pairs.sort(key=lambda x: x[2], reverse=True)


plt.figure(figsize=(10, 160))

for i, (r, c, sim) in enumerate(pairs):
    if r >= 16: break

    plt.subplot(32, 2, i*2+1)
    plt.imshow(combined[r], cmap="berlin", vmin=-1, vmax=1)
    plt.title(f"Similarity: {round(sim, 2)}")
    plt.subplot(32, 2, i*2+2)
    plt.imshow(combined1[c], cmap="berlin", vmin=-1, vmax=1)
    plt.colorbar()

    if i > 4:
        pass

print(len(pairs))

#calculate mean:
sum_sims = 0
sims = []
for i, (r, c, sim) in enumerate(pairs):
    sum_sims += sim
    sims.append(sim)

sum_mean = sum_sims / len(pairs)
sum_med = float(torch.median(torch.Tensor(sims)))
sum_max = max(sims)
sum_min = min(sims)

sum_mean, sum_med, sum_max, sum_min

#===========================================CODE CELL==========================================

weights_fc2 = net.fc2.weight.detach().cpu().unsqueeze(-1).unsqueeze(-1)
biases_fc2 = net.fc2.bias.detach().cpu()

combined_out = (weights_fc2 * combined1.unsqueeze(0)).sum(dim=1)
max_val = combined_out.max()
combined_out = combined_out / max_val

mask = torch.where(
    combined_out == 0,
    torch.tensor(0.0, device=combined_out.device),
    torch.tensor(1.0, device=combined_out.device)
)

combined_out += biases_fc2[:, None, None]
#combined_out *= mask

combined_out_copy = combined_out.clone()

per_img_max = combined_out.view(len(combined_out), -1).max(dim=1)[0]
thresholds = (0.25 * per_img_max).view(len(combined_out), 1, 1)

combined_out = torch.where(
    combined_out == 0,
    torch.tensor(0, device=combined_out.device),
    torch.where(
        torch.abs(combined_out) > thresholds,
        combined_out,
        torch.tensor(0, device=combined_out.device)
    )
)

vmax = combined_out.max().item()

fig, axes = plt.subplots(2, 5, figsize=(15, 5))
for idx, ax in enumerate(axes.flat):
    im = ax.imshow(combined_out[idx].cpu(), cmap='berlin', vmin=-vmax, vmax=vmax)
    ax.axis('off')
    ax.set_title("out_neuron")

cbar_ax = fig.add_axes([1.0, 0.15, 0.015, 0.7])
fig.colorbar(im, cax=cbar_ax)
plt.tight_layout()
plt.show()

#===========================================CODE CELL==========================================

weights_fc2 = net1.fc2.weight.detach().cpu().unsqueeze(-1).unsqueeze(-1)
biases_fc2 = net1.fc2.bias.detach().cpu()

combined_out1 = (weights_fc2 * combined1.unsqueeze(0)).sum(dim=1)
max_val = combined_out1.max()
```

```python
combined_out1 = combined_out1 / max_val

mask = torch.where(
    combined_out1 == 0,
    torch.tensor(0.0, device=combined_out1.device),
    torch.tensor(1.0, device=combined_out1.device)
)

combined_out1 += biases_fc2[:, None, None]
#combined_out1 *= mask

combined_out1_copy = combined_out1.clone()

per_img_max = combined_out1.view(len(combined_out1), -1).max(dim=1)[0]
thresholds = (0.25 * per_img_max).view(len(combined_out1), 1, 1)

combined_out1 = torch.where(
    combined_out1 == 0,
    torch.tensor(0, device=combined_out1.device),
    torch.where(
        torch.abs(combined_out1) > thresholds,
        combined_out1,
        torch.tensor(0, device=combined_out1.device)
    )
)

vmax = combined_out1.max().item()

fig, axes = plt.subplots(2, 5, figsize=(15, 8))
for idx, ax in enumerate(axes.flat):
    im = ax.imshow(combined_out1[idx].cpu(), cmap='berlin', vmin=-vmax, vmax=vmax)
    ax.axis('off')
    ax.set_title(f"Class: {idx}", fontsize=20)

cbar_ax = fig.add_axes([1, 0.12, 0.025, 0.7])
fig.colorbar(im, cax=cbar_ax)
fig.suptitle('Attention of 10 neurons in output layer fc2', fontsize=26)
cbar = fig.colorbar(im, cax=cbar_ax)
cbar.ax.tick_params(labelsize=20)
plt.tight_layout(rect=[0, 0, 1, 1])
plt.show()

#==========================================CODE CELL=========================================

import torch.nn.functional as F
from scipy.optimize import linear_sum_assignment

def normalize_map(t):
    return (t - t.mean()) / (t.std() + 1e-8)

def similarity_score(map1, map2):
    a = normalize_map(map1).view(-1)
    b = normalize_map(map2).view(-1)
    #return torch.norm(a - b, p=2).item()
    return F.cosine_similarity(a.unsqueeze(0), b.unsqueeze(0)).item()


mask = (img != 0)
mask_flat = mask.flatten()

flat_imgs = combined_out.view(combined_out.shape[0], -1)
flat_imgs1 = combined_out1.view(combined_out1.shape[0], -1)

```

```python
masked_imgs = flat_imgs[:, mask_flat]
masked_imgs1 = flat_imgs1[:, mask_flat]

sim_matrix = torch.zeros((10, 10))

for i in range(10):
    for j in range(10):
        sim_matrix[i, j] = similarity_score(masked_imgs[i], masked_imgs1[j])

cost = -sim_matrix.numpy()
row_ind, col_ind = linear_sum_assignment(cost)
max_vals = sim_matrix[row_ind, col_ind].tolist()

pairs = list(zip(row_ind, col_ind, max_vals))
pairs.sort(key=lambda x: x[2], reverse=True)


plt.figure(figsize=(10, 160))

for i, (r, c, sim) in enumerate(pairs):
    if r >= 16: break

    plt.subplot(32, 2, i*2+1)
    plt.imshow(combined_out[r], cmap="berlin", vmin=-1, vmax=1)
    plt.title(f"Similarity: {round(sim, 2)}")
    plt.subplot(32, 2, i*2+2)
    plt.imshow(combined_out1[c], cmap="berlin", vmin=-1, vmax=1)
    plt.colorbar()

    if i > 4:
        break


#calculate mean:
sum_sims = 0
sims = []
for i, (r, c, sim) in enumerate(pairs):
    sum_sims += sim
    sims.append(sim)

sum_mean = sum_sims / len(pairs)
sum_med = float(torch.median(torch.Tensor(sims)))
sum_max = max(sims)
sum_min = min(sims)

sum_mean, sum_med, sum_max, sum_min
```

Listing 3: **Activation Variation Across Model Instances**

```python
#===========================================CODE CELL===========================================

import torch
import torch.nn as nn
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn.functional as F

import numpy as np
from scipy.stats import pearsonr
import matplotlib.pyplot as plt

import os
import re
import random as ra

#===========================================CODE CELL===========================================

input_size, h0, h1, out_size = 784, 32, 16, 10

#===========================================CODE CELL===========================================

ds = datasets.MNIST(root='./data', train=False,
                  transform=transforms.ToTensor(), download=True)
loader = torch.utils.data.DataLoader(ds, batch_size=1000, shuffle=False)
images, _ = next(iter(loader))
images = images.view(-1, 784)

#===========================================CODE CELL===========================================

class Net(nn.Module):
    def __init__(self, input_size, hidden_size0, hidden_size1, out_size):
        super(Net, self).__init__()
        self.fc0 = nn.Linear(input_size, hidden_size0) #784 - 32
        self.fc1 = nn.Linear(hidden_size0, hidden_size1) #32 - 16
        self.fc2 = nn.Linear(hidden_size1, out_size)  #16 - 10
        self.tanh = nn.Tanh()
        self.init_weights()

    def init_weights(self):
        nn.init.xavier_uniform_(self.fc0.weight, gain=nn.init.calculate_gain('tanh'))
        nn.init.xavier_uniform_(self.fc1.weight, gain=nn.init.calculate_gain('tanh'))
        nn.init.xavier_uniform_(self.fc2.weight, gain=nn.init.calculate_gain('tanh'))

    def forward(self, x):
        a0 = self.tanh(self.fc0(x))
        a1 = self.tanh(self.fc1(a0))
        out = self.fc2(a1)
        return a0, a1

#===========================================CODE CELL===========================================

#examples of two model instances being loaded for comparison
netA = Net(input_size, h0, h1, out_size)
netA.load_state_dict(torch.load('stored_model_weights/model_inst_0'))
netA.eval()
netB = Net(input_size, h0, h1, out_size)
netB.load_state_dict(torch.load('stored_model_weights/model_inst_4'))
netB.eval()

#===========================================CODE CELL===========================================
#collecting activations
```

```python
actsA0, actsA1, actsB0, actsB1 = [], [], [], []
for img in images:
    a0, a1 = netA(img.unsqueeze(0))
    b0, b1 = netB(img.unsqueeze(0))
    actsA0.append(a0.detach().numpy().flatten())
    actsA1.append(a1.detach().numpy().flatten())
    actsB0.append(b0.detach().numpy().flatten())
    actsB1.append(b1.detach().numpy().flatten())
actsA0 = np.stack(actsA0); actsA1 = np.stack(actsA1)
actsB0 = np.stack(actsB0); actsB1 = np.stack(actsB1)

#colecting similarities
raw_cos0, raw_corr0, sorted_cos0, sorted_corr0 = [], [], [], []
raw_cos1, raw_corr1, sorted_cos1, sorted_corr1 = [], [], [], []

for a0, b0, a1, b1 in zip(actsA0, actsB0, actsA1, actsB1):
    #raw comparison of distribution
    raw_cos0.append(F.cosine_similarity(torch.tensor(a0), torch.tensor(b0), dim=0).item())
    raw_corr0.append(pearsonr(a0, b0)[0])
    raw_cos1.append(F.cosine_similarity(torch.tensor(a1), torch.tensor(b1), dim=0).item())
    raw_corr1.append(pearsonr(a1, b1)[0])

    #sorted comparison of values
    sa0 = np.sort(a0)[::-1].copy()
    sb0 = np.sort(b0)[::-1].copy()
    sorted_cos0.append(F.cosine_similarity(torch.tensor(sa0), torch.tensor(sb0), dim=0).item())
    sorted_corr0.append(pearsonr(sa0, sb0)[0])

    sa1 = np.sort(a1)[::-1].copy()
    sb1 = np.sort(b1)[::-1].copy()
    sorted_cos1.append(F.cosine_similarity(torch.tensor(sa1), torch.tensor(sb1), dim=0).item())
    sorted_corr1.append(pearsonr(sa1, sb1)[0])

#===========================================CODE CELL===========================================

def report(layer, raw_cos, raw_cor, sorted_cos, sorted_cor):
    print(f"Layer {layer} raw cosine similarity: {np.mean(raw_cos):.3f} pm {np.std(raw_cos):.3f}
    ↪ ")
    print(f"Layer {layer} sorted cosine similarity: {np.mean(sorted_cos):.3f} pm {np.std(
    ↪ sorted_cos):.3f}")
    print(f"Layer {layer} raw Pearson correlation: {np.mean(raw_cor):.3f} pm {np.std(raw_cor):.3
    ↪ f}")
    print(f"Layer {layer} sorted Pearson correlation: {np.mean(sorted_cor):.3f} pm {np.std(
    ↪ sorted_cor):.3f}\n")

report(0, raw_cos0, raw_corr0, sorted_cos0, sorted_corr0)
report(1, raw_cos1, raw_corr1, sorted_cos1, sorted_corr1)
```

Listing 4: **Testing Rule Approximation**

```
#===========================================CODE CELL===========================================

import torch
import torch.nn as nn
import torchvision.datasets as datasets
import torchvision.transforms as transforms

#===========================================CODE CELL===========================================

input_size = 784
hidden_size0 = 32
hidden_size1 = 16
out_size = 10

epochs = 10
batch_size = 1000
learning_rate = 0.001

#===========================================CODE CELL===========================================

train_dataset = datasets.MNIST(root='./data',
                        train=True,
                        transform=transforms.ToTensor(),
                        download=True)

test_dataset = datasets.MNIST(root='./data',
                        train=False,
                        transform=transforms.ToTensor())

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                      batch_size=batch_size,
                                      shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                      batch_size=batch_size,
                                      shuffle=False)

#===========================================CODE CELL===========================================

class Net(nn.Module):
    def __init__(self, input_size, hidden_size0, hidden_size1, out_size):
        super(Net, self).__init__()
        self.fc0 = nn.Linear(input_size, hidden_size0) #784 - 32
        self.fc1 = nn.Linear(hidden_size0, hidden_size1) #32 - 16
        self.fc2 = nn.Linear(hidden_size1, out_size)  #16 - 10
        self.tanh = nn.Tanh()
        self.init_weights()

    def init_weights(self):
        nn.init.xavier_uniform_(self.fc0.weight, gain=nn.init.calculate_gain('tanh'))
        nn.init.xavier_uniform_(self.fc1.weight, gain=nn.init.calculate_gain('tanh'))
        nn.init.xavier_uniform_(self.fc2.weight, gain=nn.init.calculate_gain('tanh'))

    def forward(self, x):
        out = self.tanh(self.fc0(x))
        out = self.tanh(self.fc1(out))
        out = self.fc2(out)
        return out

#===========================================CODE CELL===========================================

net = Net(input_size, hidden_size0, hidden_size1, out_size)
```

```python
CUDA = torch.cuda.is_available()
if CUDA:
    net = net.cuda()

#===========================================CODE CELL===========================================

# building the PyGame interface to modify inputs
import pygame
from PIL import Image
import numpy as np
import time

def ensure_hw(img_t):
    if isinstance(img_t, torch.Tensor):
        t = img_t.detach().cpu().float()
        if t.ndim == 3 and t.shape[0] == 1:
            t = t[0]
        assert t.shape == (28, 28), f"wrong shape"
        t = t.clamp(0, 1)
        return t
    else:
        raise TypeError("not a Tensor")

def tensor_to_surface(t28, scale):
    arr = (t28.numpy() * 255).astype(np.uint8)
    surf = pygame.Surface((28*scale, 28*scale))
    for y in range(28):
        for x in range(28):
            v = arr[y, x]
            color = (v, v, v)
            pygame.draw.rect(surf, color, (x*scale, y*scale, scale, scale))
    return surf

def draw_grid(screen, scale, color=(40,40,40)):
    for x in range(29):
        pygame.draw.line(screen, color, (x*scale, 0), (x*scale, 28*scale))
    for y in range(29):
        pygame.draw.line(screen, color, (0, y*scale), (28*scale, y*scale))

def main(img1, scale=16, show_grid=True):
    t = ensure_hw(img1)
    t_orig = t.clone()
    pygame.init()
    pygame.display.set_caption("MNIST editor with mouse clicks: Left=Black, Right=White, Mid=Toggle")
    screen = pygame.display.set_mode((28*scale, 28*scale))
    clock = pygame.time.Clock()

    def redraw():
        surf = tensor_to_surface(t, scale)
        screen.blit(surf, (0,0))
        if show_grid:
            draw_grid(screen, scale)
        pygame.display.flip()

    def set_pixel_from_mouse(pos, button):
        mx, my = pos
        x = mx // scale
        y = my // scale
        if x < 0 or x >= 28 or y < 0 or y >= 28:
            return
        if button == 1:
            t[y, x] = 0.0
        elif button == 3:
            t[y, x] = 1.0
        elif button == 2:
            t[y, x] = 0.0 if t[y, x] >= 0.5 else 1.0

    running = True
    first = True
    t.copy_(t_orig)
    redraw()
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

            elif event.type == pygame.MOUSEBUTTONDOWN:
                set_pixel_from_mouse(event.pos, event.button)
                redraw()

            elif event.type == pygame.MOUSEMOTION:
                if event.buttons[0] or event.buttons[1] or event.buttons[2]:
                    btn = 1 if event.buttons[0] else (3 if event.buttons[2] else 2)
                    set_pixel_from_mouse(event.pos, btn)
                    redraw()

            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
                    running = False
                elif event.key == pygame.K_r:
                    t.copy_(t_orig)
                    redraw()
            if first:
                time.sleep(1)
                first = False
                t.copy_(t_orig)
                redraw()

        clock.tick(120)
    pygame.quit()
    return t

#===========================================CODE CELL===========================================

import random as ra
import matplotlib.pyplot as plt

inputs, labels = next(iter(test_loader))
dig_to_check = 1 #choosing random digit of class 1 as noted in paper

while True:
    ind = 772 #ra.randrange(0, len(inputs)) #targeting a specific input of class 1
    if labels[ind] == dig_to_check:
        img1 = inputs[ind].squeeze()
        break

plt.imshow(img1, cmap="gray")
print(ind)

#===========================================CODE CELL===========================================

edited = main(img1)
plt.imshow(edited, cmap="gray")
pr = net(edited.cuda().view(-1, 28*28))
print(f"Prediction Model: {torch.argmax(pr)}\n{pr}")
```