RG and Sampling

Iman Faisal (igf23)
Word Count: 6440

Department of Physics, Cambridge

Aim

The aim of this data analysis project was to investigate and reproduce the results of the following paper:

- Hu HY, Wu D, You YZ, Olshausen B, Chen Y. RG-Flow: a hierarchical and explainable flow model based on renormalization group and sparse prior. Machine Learning: Science and Technology. 2022 aug;3(3):035009. [1]

1 Background

1.1 The Renormalisation Group (RG)

The renormalisation group emerged from statistical physics as a systematic method to address phenomena that involve many length scales, such as phase transitions. Pioneering ideas by Kadanoff in the 1960s introduced the notion of coarse-graining; grouping together nearby microscopic variables (e.g. spins on a lattice) into "blocks" to build an effective theory for the system's large-scale behaviour [2]. This approach was later formalised by Wilson's RG theory in the 1970s, which revolutionised the understanding of critical phenomena [3]. In essence, the RG provides a mathematical zoom-out procedure; one progressively integrates out or averages over short-range degrees of freedom, yielding new effective parameters that describe the physics at a longer length scale. This procedure can be iterated repeatedly, producing a hierarchy of models indexed by scale. Each RG step performs a scale transformation, and tracks how the system's description, e.g. its Hamiltonian or coupling constants, changes under successive coarse-graining. RG can be interpreted as defining a flow in the space of theories or Hamiltonians. As we flow through scales, microscopic details are washed out and only the dominant interactions remain. The collection of all such transformations forms a semigroup, since the coarse-graining operations are not exactly invertible.

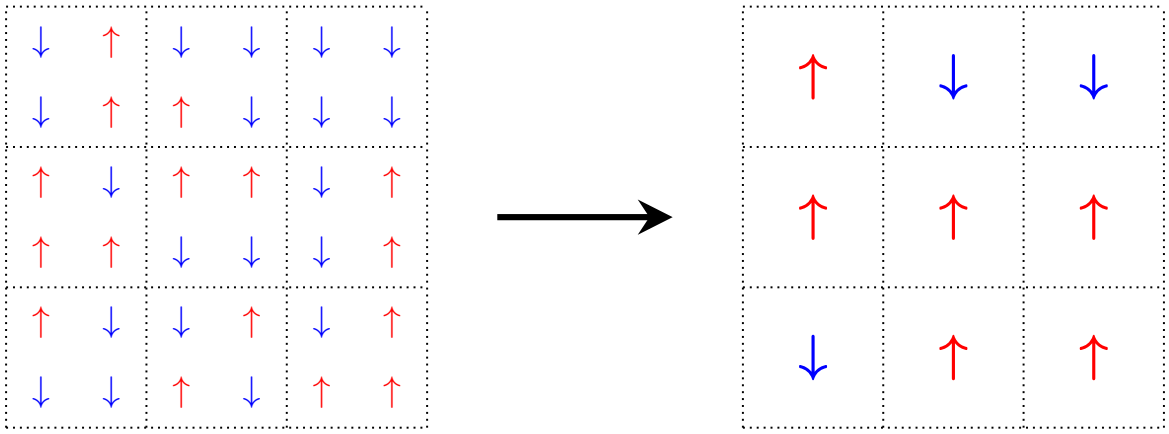

Figure 1: Illustration of Kadanoff block-spin coarse-graining transformation. The original lattice is divided into blocks and the majority vote defines the coarse-grained block-spin. In the case of a tie, ↑ wins.

Consider a system defined by a Hamiltonian H and an action S for microscopic variables σ. An RG step defines a mapping from the microscopic variables σ to new coarse-grained variables μ. For example, μ might represent a block spin equal to the average of spins within a block of the original lattice. This mapping is often stochastic in nature, meaning we sum over or integrate out the fine degrees of freedom and yields an effective Hamiltonian H' or equivalently effective action S_{eff} governing μ. Mathematically, if $T[\mu|\sigma]$ denotes the coarse-graining rule, a conditional probability or delta-functional that specifies how μ is chosen given the underlying σ configuration, one can require that the partition function is invariant under this transformation. In other words, the effective theory is defined such that the statistical weight summed over fine variables equals the weight in terms of coarse variables [4]. Formally, in the Kadanoff scheme, one has:

$$e^{-S_{\text{eff}}[\mu]} = \sum_{\{\sigma\}} T[\mu|\sigma]\, e^{-S[\sigma]}, \tag{1}$$

which ensures that

$$Z = \sum_{\{\sigma\}} e^{-S[\sigma]} = \sum_{\{\mu\}} e^{-S_{\text{eff}}[\mu]}. \tag{2}$$

This shows that the overall probability distribution remains the same after coarse-graining, but now expressed in terms of the new variables μ with their effective action. The Wilson RG scheme can be thought of as the continuum limit of this. By repeatedly applying such transformations, we generate a sequence $H \to H' \to H'' \to \cdots$ of Hamiltonians. The flow of coupling constants under these transformations can be described by RG flow equations, for example, $\frac{dg}{d\ln b} = \beta(g)$ in the continuum limit, with β a beta-function [5]. Of particular importance are fixed points of the RG flow. At a fixed point, the system looks statistically self-similar under scale transformations. Continuous phase transitions correspond to such non-trivial fixed points, where the correlation length diverges and the system exhibits scale invariance. Small deviations from the fixed point are associated with relevant or irrelevant directions: relevant perturbations grow under coarse-graining driving the system away from criticality, while irrelevant ones fade out [6]. This motivates why many microscopic details do not affect macroscopic critical behaviour.

A key element of implementing RG is choosing the right coarse-graining scheme and identifying the appropriate order parameters or collective variables that capture the essential physics. For example, in a magnetic system, rather than keeping track of every spin, one might retain only block magnetisations or other slowly varying combinations. If chosen well, the coarse variables reveal emergent structures. After many RG steps, one might be left with just a few parameters that distinguish different phases.

1.2 The Multiscale Entanglement Renormalisation Ansatz (MERA)

In quantum mechanics, entanglement describes a phenomenon wherein the quantum state of a composite system cannot be factorised into a product of the states of its individual subsystems. The state space of a composite system is the tensor product of the state spaces of its subsystems, $V_A \otimes V_B$. While this space includes separable product states of the form $|\psi\rangle\,|\phi\rangle$, where $|\psi\rangle \in V_A$ and $|\phi\rangle \in V_B$, it also contains linear combinations of these product states. A state is considered entangled if it is a linear combination,

$$\sum_{i,j} a_{ij}\, |\psi_i\rangle\, |\phi_j\rangle\,, \tag{3}$$

that cannot be simplified into the form of a single tensor product, $|\Psi\rangle\,|\Phi\rangle$. For example, the two-qubit Bell state,

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \tag{4}$$

is entangled as it cannot be written as a tensor product of any two single-qubit states [7].

Real-space RG techniques in quantum many-body physics encounter significant difficulties in the presence of long-range entanglement [8]. These methods typically truncate degrees of freedom as one coarse-grains the system, and while they excel for short-range entangled phases, they struggle at critical points or in topologically ordered states where entanglement extends across all length scales. A straightforward real-space coarse-graining scheme suffers from entanglement accumulation; short-ranged entanglement between neighbouring blocks is not removed but rather carried into the renormalised degrees of freedom. As one iterates the RG, these leftover entanglements persist and accumulate, causing an explosive growth of the effective state space dimension at each scale. This makes naive tree tensor network approaches

impractical except in the simplest cases (e.g. 1D non-critical systems), and moreover leads to non-universal, scheme-dependent RG flows. The multiscale entanglement renormalisation anstaz (MERA) addresses the entanglement accumulation problem by modifying the real-space RG procedure to explicitly remove inter-block entanglement before each coarse-graining step [9].

The MERA is a tensor network state with a distinctive layered, hierarchical structure built from two types of tensors: disentanglers and decimators[1]. Each layer of the MERA alternates between these two operations, reflecting a single step of entanglement renormalisation. Stacking many such layers produces a pyramidal network that spans multiple length scales, from the microscopic lattice (bottom layer) to a highly coarse-grained description (top layer). A d-dimensional lattice is embedded into a $(d+1)$-dimensional MERA network, where the extra dimension corresponds to RG scale. The MERA thus explicitly organises degrees of freedom by scale, which handles the multiscale entanglement.

Disentanglers (u): A disentangler is a unitary operator that acts on the Hilbert space of neighbouring sites at the interface between blocks. Its role is to reorganise local degrees of freedom such that any short-range entanglement across the block boundary is reduced or eliminated. In a 1D binary MERA each u acts on two neighbouring sites (one from each adjacent block) and maps them to two "decoupled" effective sites:

$$u:\ \mathcal{H}_{s_1'} \otimes \mathcal{H}_{s_2'} \longrightarrow \mathcal{H}_{s_1} \otimes \mathcal{H}_{s_2}, \qquad u^\dagger u = u u^\dagger = \mathbb{I} \tag{5}$$

where the dagger denotes Hermitian conjugation. The unitary condition $u^\dagger u = uu^\dagger = I$ ensures that u preserves information, no truncation occurs at this stage. Rather than discarding any part of the state, the disentangler rotates the basis of the system to one that minimises entanglement between the outgoing subsystems. u attempts to put highly entangled pairs on the same side of a block boundary, converting inter-block entanglement into intra-block entanglement. By the time the coarse-graining map is applied, the short-range entanglement has largely been reduced if not removed, so it will not propagate to larger scales. Applying all disentanglers in a layer (one per adjacent-block pair) is a local unitary circuit $U = \bigotimes u$ acting at that scale, which does not change universal properties but changes the "entanglement gauge" of the state [10, 11].

Decimator (w): Following each disentangling stage, another operator performs the actual coarse-graining or decimation of degrees of freedom. A decimator w maps a block of sites to a single site (or a smaller block) at the next, more coarse-grained scale. For example, in a binary MERA one often groups pairs of sites into one effective site. The isometry is a linear map

$$w:\ \mathcal{H}_{i_1} \otimes \cdots \otimes \mathcal{H}_{i_b} \longrightarrow \mathcal{H}_j, \qquad w^\dagger w = \mathbb{I} \tag{6}$$

with b the number of sites being merged. The constraint $w^\dagger w = I$ means w is an isometric embedding of the smaller Hilbert space into the larger one. This means w preserves inner products on the subspace it retains. Unlike u, the disentangler is generally not unitary onto the full input space ($w, w^\dagger \neq I$); instead it truncates the Hilbert space by projecting onto a subspace spanned by the lowest-energy (or most relevant) states of the block. Thus, the decimator coarse-grains the lattice; b sites at level ℓ are mapped to one site at level $\ell + 1$, reducing the total number of sites. By successive decimation across layers, the lattice size is exponentially compressed.

[1] also called isometries in the literature

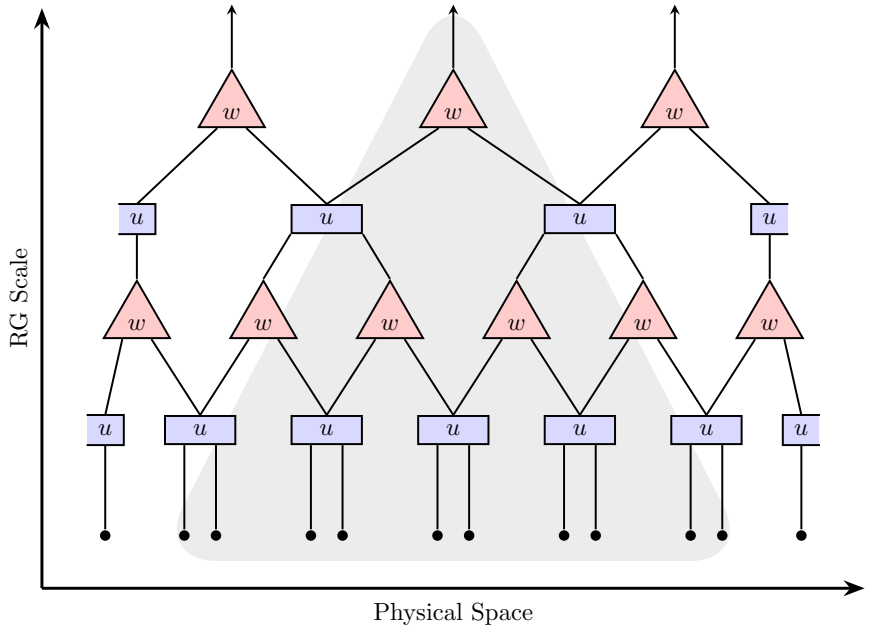

Figure 2: Binary MERA ($b = 2$) for a one-dimensional lattice. Black dots at the bottom are the physical sites. Each layer alternates unitary disentanglers u (blue squares) that remove short-range entanglement with decimators w (red triangles) that coarse-grain the system. The light-grey area represents the causal cone of a tensor. Only the tensors inside this region influence the value.

For a finite system, alternating u, w operation layers are applied until only $O(1)$ sites remain. The top tensor carries no entanglement and serves to cap off the network. It can be used to calulate expectation values of operators, $\langle \psi_{\text{MERA}} | \mathcal{O} | \psi_{\text{MERA}} \rangle$ where $\mathcal{O}$ is some arbitrary operator. The gates form a causal structure that constrains how information and correlations propagate. Each local region of the lattice is influenced by only a finite cone of tensors in the previous layers, analogous to a light cone or receptive field in a convolutional neural network. Because disentanglers are local gates, they broaden the causal cone only by a finite amount per layer, and because decimators reduce the number of degrees of freedom, the cone narrows as one ascends. Consequently, two far-separated regions of the original lattice have their causal cones meet only at high layers, implying that their correlation must be mediated through many intermediaries layers. This structural feature tends to suppress long-range correlations in a controlled way.

The result of the full MERA tensor network is a multiscale tensor representation of the many-body wavefunction. One can think of preparing the state by starting from the top and applying the MERA layers in reverse, decimators acting as unitaries "expanding" the system, and disentanglers acting as entangling unitaries. In this sense, MERA is both a variational ansatz for ground-state wavefunctions and a real-space RG circuit that incrementally builds up entanglement at longer and longer ranges.

1.3 Machine Learning and the Renormalisation Group

Deep neural networks and the renormalisation group share structural and mathematical parallels. Both systems perform hierarchical transformations across multiple scales, discarding irrelevant details while preserving essential features that govern large-scale behaviour. This correspondence indicates that deep learning architectures can be understood as computational implementations of RG principles and RG principles can be used to help design neural architectures [12, 13].

1.3.1 Statistical Physics and Deep Neural Networks

Deep neural networks and their architectures are often studied in an empirical manner, with limited theoretical understanding of why they perform so effectively. The training process involves optimising high-dimensional, non-convex loss surfaces, where there is no formal guarantee of convergence to a suitable minimum or of generalisation to unseen data [14]. Effective field theory and the renormalisation group offer a principled framework through which to analyse this apparent "unreasonable effectiveness" of deep learning.

Consider a multilayer neural network with preactivation variables $z^{(\ell)}$ at layer ℓ. Each layer applies a nonlinear transformation that mirrors RG's scale transformations, progressively coarse-graining the data representation. Just as RG integrates out microscopic degrees of freedom to reveal dominant long-range correlations, deeper network layers capture increasingly abstract features while discarding unnecessary microscopic detail. This process can be formalised through RG-like recursion relations for network parameters. In the infinite-width limit, where every layer contains infinitely many neurons, the central limit theorem ensures that preactivations become Gaussian [15]. The system's behaviour is then completely characterised by the two-point kernel $K^{(\ell)}_{\alpha\beta} = \langle z^{(\ell)}_\alpha z^{(\ell)}_\beta \rangle$, which measures correlations between preactivations at different input locations. This kernel obeys a deterministic recursion:

$$K^{(\ell+1)}_{\alpha\beta} = C_b + C_W \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\Sigma^{(\ell)}_{\alpha\beta})}[\sigma(u)\sigma(v)] \tag{7}$$

where $\Sigma^{(\ell)}_{\alpha\beta}$ is constructed from the previous kernel, C_b and C_W represent bias and weight variances respectively, and σ is the activation function. This equation is formally identical to a discrete RG transformation, with network depth ℓ playing the role of the logarithmic scale parameter. Fixed points of this map correspond to critical initialisations where signal variance neither explodes nor vanishes as depth increases [16, 17]. At such fixed points, the network maintains stable signal propagation through arbitrarily deep architectures, analogous to how RG fixed points govern the scale-invariant behaviour observed at phase transitions. Real networks possess finite width n_ℓ, introducing deviations from Gaussianity that scale with the depth-to-width ratio $r = L/\bar{n}$. These finite-width corrections act as weak interactions that dress the kernel flow while preserving its essential structure [18]. From an RG perspective, they can be interpreted as irrelevant operators; they modify quantitative details but do not alter the large-scale dynamics provided $r \ll 1$.

The training dynamics of deep neural networks can also be studied theoretically as a second RG flow. During gradient descent, the neural tangent kernel (NTK) Θ evolves alongside the structural kernel. The NTK characterises how the network's output changes with respect to small perturbations in the input, and governs the learning dynamics by determining how parameter updates affect the network's predictions. In the infinite-width limit, Θ remains constant throughout training, yielding a kernel-based learning rule where the network's behaviour is completely determined by its initial kernel. At finite width, Θ drifts according to flow equations that run in parallel with the structural kernel flow. Stable training corresponds to operating near a joint fixed manifold of (K, Θ) where gradients propagate without attenuation and the learning dynamics remain well-behaved [19].

The connection to criticality emerges from the fixed point structure: initialising near critical fixed points maximises the depth over which correlations survive, placing the network at the edge of chaos. This critical regime enables the network to propagate information effectively through many layers while avoiding the extremes of vanishing or exploding gradients. Away from criticality, networks either lose signal strength exponentially with depth (ordered phase) or exhibit chaotic sensitivity to inputs (disordered phase). Different activation functions define distinct universality classes, each with characteristic scaling behaviours and fixed point structures. For instance, ReLU networks sit at a marginal fixed point supporting power-law correlation decay, while tanh networks flow to ordered or disordered phases depending on the initialisation parameters (C_W, C_b). Architectural modifications such as dropout, residual connections, and weight scaling can be interpreted as RG perturbations that shift the network within its critical basin, potentially stabilising training or improving expressivity.

This RG interpretation can help explain several empirical observations in deep learning: early layers detect local patterns while deeper layers capture global structure, reflecting the hierarchical nature of RG coarse-graining; gross hyperparameter choices matter more than microscopic details, consistent with RG universality; and noise injections such as stochastic depth act like finite temperature, stabilising the network against unstable directions in parameter space.

1.3.2 MERA as a Deep Learning Architecture

The quantum MERA formalism can be simplified to a classical probabilistic version called the correlation renormalization ansatz (CORA) [20]. In CORA, quantum entanglement is replaced by classical correlations, transforming the architecture into a hierarchical generative model for probability distributions. Each layer consists of stochastic maps that progressively build correlations at increasing length scales, mirroring how deep networks construct complex features from simpler components.

MERA and other tensor network architectures have been applied to a variety of problems; from natural language processing, generative modelling, and unsupervised learning [21, 22, 23]

1.4 Normalising Flows

Normalising flows are a class of generative models that construct a complex data distribution by applying a sequence of invertible transformations (diffeomorphisms) to a simple base distribution. Formally, if x denotes an observed data vector and z a latent vector drawn from a known prior $p_Z(z)$ (e.g. a standard Gaussian), a normalising flow defines a smooth, bijective mapping $f : \mathcal{X} \to \mathcal{Z}$ between the data space and latent space. This invertible mapping f (with inverse f^{-1}) allows any data point x to be encoded as $z = f(x)$ and decoded back via $x = f^{-1}(z)$. Because of the one-to-one correspondence, one can evaluate exact data likelihoods using the change-of-variables formula. If $z = f(x)$, the densities satisfy

$$p_X(x) = p_Z(z) \left| \det \frac{\partial z}{\partial x} \right| = p_Z(f(x)) \left| \det J_f(x) \right|, \tag{8}$$

where $J_f(x) = \partial f(x)/\partial x$ is the Jacobian of f at x. Taking logs yields the log-likelihood for a single data point:

$$\ln p_X(x) = \ln p_Z(f(x)) + \ln \left| \det \frac{\partial f(x)}{\partial x} \right|. \tag{9}$$

In practice f is implemented as a chain of invertible transformations $f = f_1 \circ f_2 \circ \cdots \circ f_K$, so that by repeated application of the rule one obtains $\ln p_X(x) = \ln p_Z(z) + \sum_{k=1}^{K} \ln |\det J_{f_k}|$. Each f_k (often called a bijector or flow layer) is chosen to be an invertible function with tractable Jacobian determinant, such that the total log-density is easy to compute. This framework enables exact and efficient density evaluation, setting normalising flows apart from other generative models like GANs or VAEs which do not use exact likelihoods.

Training objective: Given a dataset $\{x_i\}_{i=1}^N$, one can fit a normalising flow by maximum likelihood. The model's parameters (in the bijective layers and the prior) are optimised to maximise the total log-likelihood $\sum_i \ln p_X(x_i)$, or equivalently to minimise the average negative log-likelihood (NLL) loss. The loss for a dataset of size N can be written as

$$\mathcal{L}_{\text{NLL}} = -\frac{1}{N} \sum_{i=1}^{N} \ln p_X(x_i), \tag{10}$$

which by the change-of-variable expands to

$$\mathcal{L}_{\text{NLL}} = -\frac{1}{N} \sum_{i=1}^{N} \left[\ln p_Z(f(x_i)) + \ln \left| \det J_f(x_i) \right| \right]. \tag{11}$$

Minimising this loss encourages f to transform the data into latent variables $z_i = f(x_i)$ that follow the chosen prior distribution p_Z as closely as possible. In other words, training a flow amounts to learning an invertible reparametrisation of the data that "Gaussianises" the distribution. As long as each component f_k of the flow has a tractable Jacobian determinant (e.g. a triangular Jacobian as in coupling-layer architectures), the log-likelihood is exact and efficient to compute, allowing the model to be trained by standard gradient-based optimisation [24]. Once trained, one can generate samples from the data distribution by drawing from the latent distribution $z \sim p_Z$ and applying the inverse transform: $x = f^{-1}(z)$.

2 Neural Architecture

2.1 RG-Flow

RG-Flow's neural architecture is directly inspired by the MERA. MERA organises computations in a layered, hierarchical fashion using alternating disentangler and decimator transformations to progressively coarse-grain a system while mitigating short-range entanglement. RG-Flow can be considered as a classical implementation of this idea, replacing quantum entanglement with classical statistical correlations. This is referred to as the Correlation Renormalisation Ansatz (CORA) by Bény (2013) [20]. RG-Flow utilises classical normalising flow transformations, providing a deterministic and invertible mapping between observed data and latent representations.

Each layer of the RG-Flow architecture corresponds to one step of an RG transformation, implemented as an invertible, bijective map operating on a local subset of variables. Formally, if we denote by $x^{(h)}$ the variables at scale h with $x^{(0)} = x$ as the original observed input, the forward renormalisation transformation R_h produces a more coarse-grained representation $x^{(h+1)}$ while discarding fine-scale detail into latent variables $z^{(h)}$. This can be expressed as a recurrence relation for each layer h of the model:

$$x^{(h+1)}, z^{(h)} = R_h\left(x^{(h)}\right), \tag{12}$$

where R_h is an invertible and trainable local transformation. Here $x^{(h+1)}$ contains only the coarse-grained degrees of freedom passed on to the next layer, whereas $z^{(h)}$ represents the discarded fine-grained information at that scale. By construction, R_h is bijective and hence invertible; we denote its inverse as $G_h = R_h^{-1}$, which performs the corresponding generative step. The inverse transform G_h takes the coarse variables from the next scale together with the latent variables from scale h and reconstructs the finer-scale representation:

$$x^{(h)} = G_h\left(x^{(h+1)},\, z^{(h)}\right). \tag{13}$$

Stacking these transformations across all levels yields a multiscale normalising flow that factorises the generation of x into a sequence of hierarchical steps. At the top, most coarse layer, there is no further coarse-graining above, so the entire highest-level representation is generated from a latent vector $z^{(H)}$ alone. The full generative process is then given by composing all the inverse RG layers in sequence:

$$x = G_0\left(G_1\left(\dots G_H\left(z^{(H)}\right)\dots, z^{(1)}\right), z^{(0)}\right), \tag{14}$$

where $z^{(H)}, \dots, z^{(0)}$ are the collection of independent latent variables at all scales. In simpler terms, RG-Flow's generator G first synthesises the most coarse-grained structure from the highest-level latent $z^{(H)}$, then sequentially injects finer details $z^{(H-1)}, \dots, z^{(0)}$ through the series of inverse transforms $G_H, G_{H-1}, \dots, G_0$ until the full-resolution data x is obtained. This layered construction ensures that only a subset of variables at each scale are propagated upward, while all remaining variables are treated as latent information specific to that scale.

Each RG-layer h of the RG-Flow model consists of disentangler and decimator operations, $R_h = R_h^{\text{dec}} \circ R_h^{\text{dis}}$. These are directly analogous to the gates of the same name in MERA. These are applied as local transformations with an $m \times m$ kernel. These transformations use periodic boundary conditions (circular wrapping), which ensures translational invariance and avoids boundary artefacts. The disentangler and decimator are both implemented as bijective normalising flows mapping $C \times m \times m$ tensors to $C \times m \times m$ tensors, preserving the number of variables. The kernel size, m, can be considered as the number of sites merged, b, in MERA.

The coarse-graining operation performed by the decimator is achieved by selecting only a fraction of the transformed variables to propagate to the subsequent RG-layer. Specifically, for a 2D kernel of size $m = 4$, the decimator partitions the output into distinct interleaved sub-lattices, selecting one subset to form the coarse variables $x^{(h+1)}$ passed upwards, with the remaining pixels constituting latent variables $z^{(h)}$ at scale h. For instance, the pixel located at $(0,0)$, $(0,2)$, $(2,0)$ and $(2,2)$ within each 4×4 kernel is passed to the next layer, while all other pixels within the kernel are designated as latent variables.

Formally, the operation of the disentangler (R_h^{dis}) and the decimator (R_h^{dec}) at scale h is given by:

$$\left\{y^{(h)}_{r_{a,b}}\right\}_{(a,b)\in\{0,\dots,m-1\}^2} \;=\; R_h^{\text{dis}}\left(\left\{x^{(h)}_{r_{a,b}}\right\}_{(a,b)\in\{0,\dots,m-1\}^2}\right) \tag{15a}$$

$$\left\{x^{(h+1)}_{r'_{a,b}}\right\}_{(a,b)\in\{0,2,\ldots,m-2\}^2},\ \left\{z^{(h)}_{r'_{a,b}}\right\}_{(a,b)\in\{0,\ldots,m-1\}^2\setminus\{0,2,\ldots,m-2\}^2} = R_h^{\mathrm{dec}}\left(\left\{y^{(h)}_{r'_{a,b}}\right\}_{(a,b)\in\{0,\ldots,m-1\}^2}\right) \quad (15b)$$

where $y^{(h)}$ denotes intermediate variables between the disentangler and decimator transformations [1]. The spatial positions for pixels are given by:

$$r_{a,b} = 2^h\left(mp + \frac{m}{2} + a,\ mq + \frac{m}{2} + b\right), \quad (15)$$

$$r'_{a,b} = 2^h\left(mp + a,\ mq + b\right), \quad (16)$$

where $p, q \in \{0, 1, \ldots, L/(2^h m) - 1\}$ index the spatial blocks, and the pixel indices a, b range over the kernel size.

These indexing equations can be directly interpreted through the framework of convolutional neural networks (CNNs), which employ kernels characterised by a kernel size $k \times k$, dilation d, and stride s. The general formula for convolutional indexing is:

$$(i, j) = (sp + da, sq + db), \quad a, b \in 0, \ldots, k - 1. \quad (17)$$

Comparing term-by-term reveals the following correspondences:

- **Kernel size**: The RG-Flow equations explicitly identify $m \times m$ blocks, directly corresponding to a convolutional kernel of size $k = m$.
- **Dilation**: The factor of 2^h in the RG-Flow indexing aligns with the dilation parameter $d = 2^h$.
- **Stride**: A spatial offset of $m2^h$ pixels when moving between adjacent blocks (p, q) exactly matches a convolutional stride of $s = m2^h$.
- **Pixel offset**: An additional offset of $m/2$ present in the disentangler indexing can be readily interpreted as a kernel centring adjustment or padding shift in a CNN.

Thus, the RG-Flow sampling strategy is formally equivalent to employing convolutional kernels with size m, stride $m2^h$, and dilation 2^h. The extra offset introduced in the disentangler merely serves as a centring adjustment, a common technique in CNNs achieved through padding.

To make this explicit, we define the patch extraction operator:

$$\mathrm{Patch}_h(p, q) = \left\{x_{m2^h p + 2^h a,\ m2^h q + 2^h b}\right\}_{a,b=0}^{m-1}. \quad (18)$$

Then the disentangler and decimator transformations become:

$$y^{(h)}_{p,q} = R_h^{\mathrm{dis}}\left(\mathrm{Patch}_h(p, q)\right), \quad (19)$$

$$\left(x^{(h+1)}_{p,q},\ z^{(h)}_{p,q}\right) = R_h^{\mathrm{dec}}\left(\mathrm{Patch}_h(p, q)\right). \quad (20)$$

These patch-based operations can be efficiently implemented using PyTorch's built-in functions, `torch.nn.Unfold` and `torch.nn.Fold`. The spatial indexing required by the RG-Flow transformations corresponds to extracting local patches of the input tensor and rearranging them into a batch dimension. The function `torch.nn.Unfold` performs exactly this operation, producing a structured representation where patches are collected and arranged systematically for simultaneous processing. Subsequently, after applying the disentangler and decimator transformations, the processed patches can be recombined into their original spatial configuration using `torch.nn.Fold`.

This method offers significant computational advantages compared to a naive array-based implementation. Naive implementations relying on explicit Python loops or manual indexing lead to inefficient memory access patterns and a substantial computational overhead, particularly when processing large-scale data. In contrast, `torch.nn.Unfold` and `torch.nn.Fold` utilise optimised, low-level Cuda routines that leverage contiguous memory layouts, parallel processing, and hardware acceleration (e.g., GPU parallelisation). These optimisations lead to improved cache efficiency, faster execution, and reduced memory usage. Thus, using PyTorch's built-in functions not only simplifies the code but also significantly enhances the computational efficiency.

This method allows for one $B \times C \times H \times W$ tensor to be passed through the network without the need of additional coarse graining logic. This keeps computation on the GPU allowing for faster training and generation.

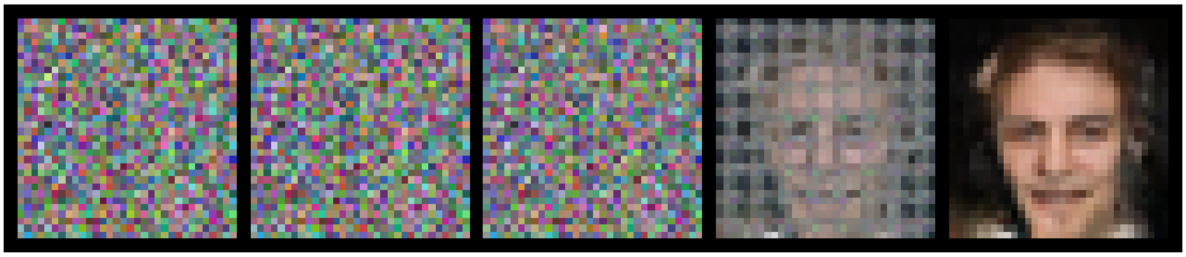

Figure 3: Samples as they propagate through the network; *right* $\rightarrow$ *left* shows generation, *left* $\rightarrow$ *right* shows training dynamics.

2.2 Coupling Layers

Each coupling layer in RG-Flow is based on the RealNVP (RNVP) scheme of splitting the input into two parts and applying an affine transformation to one part conditioned on the other. In the implementation, the input image is first divided into small patches and a fixed checkerboard mask is applied to partition each patch into two interleaved subsets x_1 and x_2 (each comprising half of the patch's variables). The coupling transform then keeps one subset unchanged while scaling and shifting the other subset using functions of the first subset. For example, given x_1 as input to the coupling network, the other half is updated as:

$$x_2' = x_2 \odot \exp(\tanh(s(x_1))) + t(x_1), \tag{21}$$

where $s(\cdot)$ and $t(\cdot)$ are learnable scale and translation functions, and $\odot$ denotes elementwise multiplication. The tanh in the coupling layer clamps the effective scale to $\tilde{s} \in (-1, 1)$. This affine coupling is bijective since x_1 remains unchanged and x_2 can be recovered by inverting the scale and shift using the same $s(x_1)$ and $t(x_1)$. After this step, RG-Flow performs a second coupling in the reverse direction: using the updated x_2' to affinely transform x_1, producing x_1'. Finally, x_1' and x_2' are concatenated to form the output of the coupling block. In this way, a single RNVP block consists of a pair of affine coupling sub-layers (one in each direction), ensuring that both halves of the data interact and no variables remain static through the block. In RG-Flow, different scales of the hierarchy use different numbers of coupling blocks (e.g. more at the lowest level where inputs are larger), to adequately model the complexity at each scale.

The scale and translation functions $s(x)$ and $t(x)$ within each coupling layer are implemented as neural networks with residual block architectures. Each of these networks is a small fully-connected residual network with a preactivation ResNet structure. In the reported implementation for the CelebA dataset, one coupling network consists of $n_{\text{res}} = 4$ sequential residual blocks. Each residual block has three dense linear layers with dimensions $48 \to 512 \to 512 \to 48$. 48 corresponds to $C \times m \times m$ for $m = 4$ and $C = 3$. Between these linear layers, SiLU activation functions with a learnable weight are applied. Because the residual blocks use a pre-activation design, all non-linearities are applied inside the block before the addition of the skip connection. The skip path itself is an identity, there is no activation on the skip. The skip connections are scaled by $\sqrt{2}$:

$$y = \frac{f(x) + x}{\sqrt{2}}. \tag{22}$$

This scaled residual connection is done to improve training stability and signal propagation in the deep network. By averaging the output with the input, the network avoids amplifying the variance through many layers, effectively normalising the output of each block to a similar scale as its input. This prevents the signal from growing or vanishing as it passes through numerous residual blocks, thus promoting stable gradients and convergence during training. The $1/\sqrt{2}$ factor ensures that if $f(x)$ has a similar scale to x, their sum will have roughly the same standard deviation as each individually, which guards against activation values blowing up in deeper compositions. This is a common alternative to batch or layer normalisations [25, 26]. Importantly, this form of residual addition does not hinder invertibility of the

overall coupling layer since the coupling layer's invertibility is guaranteed analytically by the affine transformations.

Furthermore, to stabilise training, all linear layers in the residual blocks are given careful initialization and normalization. The implementation uses Kaiming (He) initialisation for the weights and applies weight normalization to each linear layer's weights. Weight normalization reparameterises each weight vector with a learned scale, effectively treating the overall layer gain as a trainable parameter. This technique, in conjunction with the scaled residual connections, helps maintain a healthy flow of gradients and prevents any single layer from dominating the signal.

Gradient checkpointing is employed between coupling layers to significantly reduce memory requirements during training. Instead of storing all intermediate activations necessary for gradient computation, selected activations are discarded during the forward pass and recomputed when needed in the backward pass. This approach effectively trades additional computation for substantial memory savings.

2.3 Prior Distributions

The choice of prior distribution in normalising flows exerts a influence on both latent-space geometry and the statistical properties of the learned model. In this work we consider two candidates: an isotropic Gaussian and a Laplacian distribution. The Gaussian prior is light-tailed, with density decaying as $\exp\left(-\frac{1}{2}z^2\right)$, whereas the Laplacian prior, with density $\propto \exp(-|z|)$, displays both a sharper central peak and heavier exponential tails; this allocates more mass near zero (promoting sparsity) while still assigning appreciable probability to extreme latent values.

Beyond this qualitative difference, the two priors impose distinct regularisation regimes. The isotropic Gaussian is spherically symmetric. Any orthogonal rotation of the latent vector leaves its probability unchanged, so models trained with it are indifferent to how information is distributed across coordinates and typically learn dense, entangled codes governed by an L^2 penalty. In contrast, the factorised Laplacian is axis-aligned and separable; mixing or rotating coordinates lowers joint probability because independence is enforced component-wise. Optimising under this prior is therefore equivalent to an L^1-style penalty, which encourages each latent variable either to remain near zero or to take a large task-relevant value, yielding sparse and more interpretable encodings.

Heavier tails also endow the Laplacian-based model with practical advantages. Because a Lipschitz normalising flow cannot produce output tails heavier than those of its base distribution, a Gaussian prior fundamentally limits the model's ability to capture heavy-tailed data. Moreover, the squared-error penalty associated with Gaussian tails gives unbounded influence to outliers, making training sensitive and often necessitating heuristic gradient clipping, whereas the Laplacian's absolute-error loss has a bounded influence function that automatically caps the gradient contribution of any single sample. Empirically, replacing a Gaussian base with a heavy-tailed alternative (Laplace or Student-t) permits stable optimisation at higher learning rates and yields better held-out likelihoods [27].

To control the strength of regularisation in a uniform manner we introduce a scalar temperature $T \in (0, \infty]$ into both base measures. Under this scheme the Gaussian prior becomes

$$p_T^{\text{Gauss}}(z) = \frac{1}{(2\pi T)^{D/2}} \exp\Big(-\tfrac{1}{2T}\|z\|_2^2\Big), \tag{23}$$

so that its covariance shrinks linearly with T. For the Laplacian prior we use

$$p_T^{\text{Lap}}(z) = \prod_{d=1}^{D} \frac{1}{2T} \exp\Big(-\tfrac{1}{T}|z_d|\Big). \tag{24}$$

Lower temperatures compress mass toward the origin and correspondingly tighten the effective L^2 or L^1 penalty imposed by the Gaussian and Laplacian bases. Here D is over (h, i, j, C), ever RG level, pixels, and colour channel.

2.4 Datasets

We consider three datasets within this study; the CelebA dataset, and two synthetic datasets created by Hu et al in the original work [28, 1].

2.4.1 CelebA

The CelebA dataset consists of face images aligned and cropped to focus on facial features. To prepare the dataset for training, the following preprocessing steps are applied. Initially, a central crop of dimensions 148×148 pixels is extracted from each original image, removing unnecessary background details and preserving consistent facial alignment. Subsequently, the cropped images undergo spatial downsampling to 32×32 pixels using bicubic interpolation. The data is further augmented through horizontal flipping, applied with a probability of 0.5.

Figure 4: Examples of the processed CelebA dataset.

The pixel intensities, originally discrete 8-bit integers in the range [0, 255], are mapped onto a continuous domain compatible with normalising flows. To achieve this, a three-step process is applied:

1. **Dequantisation:** Pixels are scaled into the $[0, 1]$ interval and then continuous uniform noise $u \sim \mathcal{U}[0, 1)$ is added to remove discretisation artifacts:

$$x_0 = \frac{x_{\text{uint8}} + u}{256}, \quad x_0 \in (0, 1). \tag{25}$$

2. **Constraint Application:** To ensure numerical stability and avoid boundary issues during training, the data are linearly scaled into a restricted range $[0.05, 0.95]$:

$$x_r = \alpha + \beta x_0, \quad \alpha = 0.05, \quad \beta = 0.9. \tag{26}$$

3. **Inverse Sigmoid (Logit) Transformation:** Finally, data in the constrained interval are transformed onto an unbounded domain via the logit function:

$$y = \log \frac{x_r}{1 - x_r}. \tag{27}$$

The log-determinant of the Jacobian (LDJ) associated with this composite transform is calculated explicitly as:

$$\log |\mathcal{J}| = \sum_i \left[\log \beta - \log x_{r,i} - \log(1 - x_{r,i}) \right]. \tag{28}$$

This can be inserted into the bijector, allowing exact likelihood computations required for training normalising flow models. The complete transformation sequence is invertible, enabling straightforward reconstruction of generated samples back into image space.

The application of the logit transformation is needed to reconcile the difference in support between the data distribution and the latent prior distribution. The prior distribution are defined on $\mathbb{R}^D$. The image data is on a compact, bounded domain $[0.05, 0.95]^D$ after scaling. Without further transformations, this discrepancy creates a fundamental mismatch, as the prior assigns significant probability mass to regions inaccessible by the data, while simultaneously offering negligible density in the data's bounded region. Such domain incompatibility adversely affects model training, leading to numerical instabilities and inefficiencies in density estimation near boundaries. The logit transformation mitigates this issue by smoothly mapping the restricted interval onto the unbounded domain. This ensures compatibility with the priors.

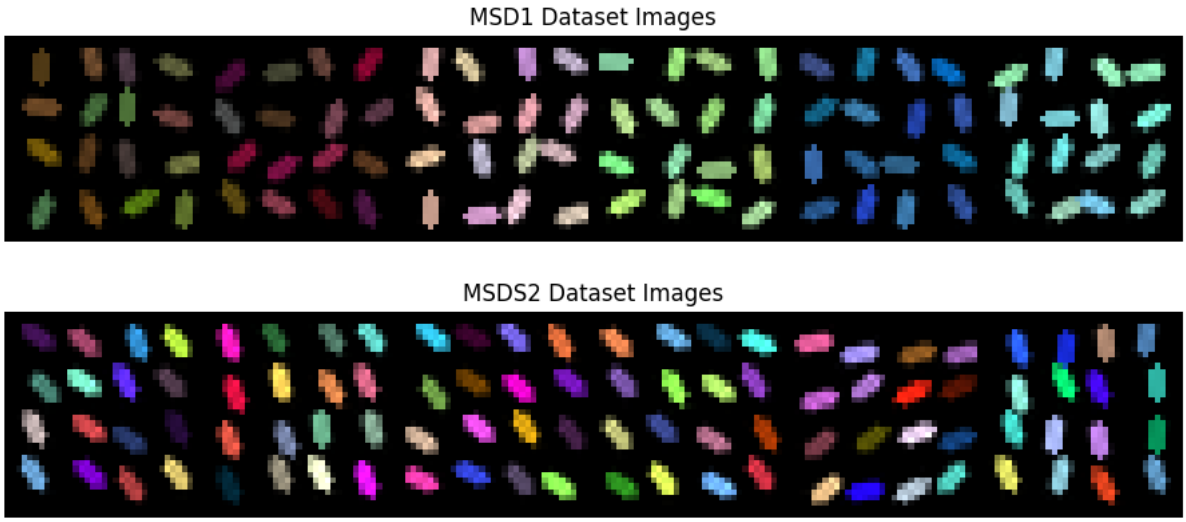

Figure 5: Examples of the synthetic datasets.

2.4.2 Synthetic Datasets (MSDS1 and MSDS2)

The synthetic datasets, MSDS1 and MSDS2, were employed as benchmarks to evaluate the hierarchical disentanglement capabilities of RG-Flow. Each dataset comprises 10^5 RGB images, each with dimensions 32×32 pixels. Images in both datasets feature 16 ellipses arranged on a slightly perturbed 4×4 grid.

In MSDS1, the ellipses share a common colour sampled globally for each image, while their orientations vary independently per ellipse. Conversely, in MSDS2, all ellipses share a global orientation, whereas their colours are independently drawn per ellipse. This construction yields distinct global-local factorisations, enabling controlled assessments of hierarchical feature separation.

The synthetic nature of these datasets ensures statistical consistency across the images. Both datasets were used with the dequantisation transformation as outlined in Section 2.4.1.

2.5 Hyperparameters and Training

In all experiments we use $m = 4$ and $n_{\text{res}} = 4$ with the hidden unit size set to 512. For the CelebA dataset we use $n_{\text{layer}} = [8, 6, 4, 2]$. For the synthetic dataset we use $n_{\text{layer}} = 4$ for all h. The CelebA model was trained for 100 epochs and the synthetic models for 60. The training employed the AdamW optimiser with an initial learning rate of 10^{-3} and weight decay of 5×10^{-5}. Gradient clipping was applied with a global norm threshold of 1.

We differ from the original work and use a batch size of 512 rather than 64. This is the maximum batch size that could fit in VRAM on a Nvidia 3060 Ti graphics card and this minimise training time. This was done due to compute time limitations. We note that this can

3 Reproducibility Challenges

Several key details necessary for accurate reproduction were omitted from the original paper, significantly affecting convergence and training stability. These issues include:

- **Normalization Factor**: A normalization factor of $\sqrt{2}$ within coupling layers was not explicitly stated. Omitting this factor results in non-convergent behaviour, likely due to shattered gradients.
- **Preactivation Scheme**: The original paper does not clarify the use of a preactivation scheme (batch normalization before activation). Implementing this scheme substantially affects training stability and convergence speed.
- **SiLU Activations**: The original work specifies SiLU activations without indicating whether these activations include learnable weight or gain factors. Standard implementations (e.g., PyTorch) do

not inherently include these parameters, significantly altering the parameter count and training dynamics.

- **Scaling Factor (tanh)**: The application of a hyperbolic tangent (`tanh`) to the scaling factor s in affine coupling layers was not explicitly mentioned. This detail greatly impacts gradient stability and training dynamics.
- **Coupling Networks**: Figure A1 from the original paper is ambiguous regarding whether one or two affine coupling networks are used per coupling layer. This uncertainty required additional experimentation, revealing significant performance differences based on implementation.
- **Dequantisation**: The dequantisation scheme stated in Section 2.4.1 is present in the original work but not stated in the publication.
- **Model Parameter Count**: The original authors report "approximately one million parameters", likely referring to a single RG-layer rather than the entire model. Both the original and reproduced models contain exactly 10,064,780 parameters. This was not immediately obvious and had to be verified by checking their public code. This makes a significant impact on model performance.

4 Results

Figure 6: Samples generated from the Gaussian prior RG-Flow with $T = 0.75$

4.1 Metrics

The performance of our RG-Flow replication is assessed quantitatively using two established metrics commonly applied in evaluating generative models: Bits-per-Dimension (BPD) and the Fréchet Inception Distance (FID).

4.1.1 Bits-per-Dimension (BPD)

Bits-per-Dimension quantifies the efficiency of a generative model in capturing the data distribution by measuring the negative log-likelihood (NLL) per dimension, expressed in units of bits. For a given dataset, $\{x_i\} \subseteq \mathbb{R}^D$, the BPD is defined as:

$$\text{BPD} = -\frac{1}{ND \log 2} \sum_{i=1}^{N} \log p_\theta(x_i), \tag{29}$$

where $p_\theta(x_i)$ is the latent probability density function predicted by the flow model, D denotes the dimensionality of the data, and the factor $\log 2$ converts the measurement from nats to bits. Lower BPD values indicate superior model performance, signifying that the model assigns higher likelihoods to the observed data points.

4.1.2 Fréchet Inception Distance (FID)

In contrast to BPD, which assesses distribution fidelity directly, FID provides a perceptual measure of realism by evaluating the similarity between generated and real images within the feature space of a pre-trained Inception-v3 network. The Inception-v3 network, trained extensively on ImageNet, captures high-level semantic and visual features within images, enabling a comparison that aligns closely with human visual perception.

Specifically, FID is computed using the following equation:

$$\text{FID} = |\mu_r - \mu_g|_2^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}\right), \tag{30}$$

where $(\mu_r, \sum_r)$ and $(\mu_g, \sum_g)$ represent the mean and covariance matrices of features extracted from the real and generated image sets, respectively. Both groups of images are passed in batches through the pre-trained Inception-v3 model, outputting a 2048-dimensional vector. This vector can be considered an encoding of the image's semantic content. A smaller FID indicates that the distributions of real and generated images closely match, suggesting better perceptual quality and realism. The metric thus penalises common issues such as blurring, lack of diversity, and mode collapse, making it a robust and sensitive measure of generative performance.

Dataset	BPD (Laplacian)	BPD (Gaussian)	FID (Laplacian)	FID (Gaussian)
CelebA	3.7238	3.7432	83.815	80.418
MSDS1	1.8562	1.8505	103.164	103.637
MSDS2	1.9375	1.9457	105.798	106.914

Table 1: Comparison of bits per dimension (BPD) and Fréchet Inception Distance (FID) across different datasets using Laplacian and Gaussian priors.

4.2 Latent Space Investigation

To verify whether our implementation of RG-Flow successfully learns a hierarchical, scale-separated representation, we examined the latent variables through receptive field analysis as defined by Hu et al. [1]. Receptive fields indicate the region in pixel space influenced by perturbing a single latent variable, thus providing direct insight into the hierarchical organisation of the learnt representation.

$$\text{RF}_l = \mathbb{E}_{z_l \sim p_Z(z)} \left| \frac{\partial G(z)}{\partial z_l} \right|_c \tag{31}$$

We selected a representative subset of latent variables across each hierarchical level and perturbed these individually with a unit amplitude impulse. The corresponding changes in image space were computed via the inverse RG transformation, generating heatmaps that visualise the extent and magnitude of the affected regions. The results of this analysis are presented in Figure 7, with rows corresponding to hierarchical RG levels from fine ($h = 0$) to coarse ($h = 3$), and columns representing different spatial locations at each level.

At the coarsest scale ($h = 3$), latent perturbations influence broad, global structures within the image, including general head shape, background lighting, and coarse facial features. Descending through the hierarchy, variables at intermediate level ($h = 2$) impact mid-scale facial components such as eyes, nose, and mouth, reflecting attributes like facial expression and gaze direction. At the finer scales ($h = 1$ and 2), perturbations become increasingly localised, primarily affecting small, distinct patches corresponding to fine-grained features, such as highlights in pupils or subtle shading around facial features.

Key observations from this analysis include:

1. **Hierarchical Scale Separation**: Receptive fields expand systematically with increasing RG level, approximately doubling in spatial extent at each successive layer, consistent with theoretical expectations of a renormalisation group procedure.
2. **Localisation and Disentanglement**: At the finest scales ($h \leq 1$), receptive fields remain highly localised with minimal overlap, demonstrating effective statistical independence among latent variables at these levels.
3. **Semantic Organisation**: Mid- to high-level variables ($h = 2, 3$) correspond to coherent semantic features rather than arbitrary pixel combinations, indicating the network has learned semantically meaningful latent representations.
4. **Consistency with RG-Flow**: The qualitative evolution of receptive fields through hierarchical levels aligns closely with the original findings by Hu et al. [1], validating the accuracy of the model reproduction.

These findings confirm that our model has successfully captured the intended hierarchical structure described by the renormalisation group framework. Each latent variable operates within a well-defined spatial and semantic context, supporting targeted manipulation of image attributes at multiple scales. Minor artefacts, such as faint halos surrounding fine-scale perturbations, suggest slight residual coupling between adjacent latent variables, possibly due to the chosen stride configuration during training. Nonetheless, the clear hierarchical organisation demonstrates that our RG-Flow model has learned to disentangle image representations effectively across multiple scales.

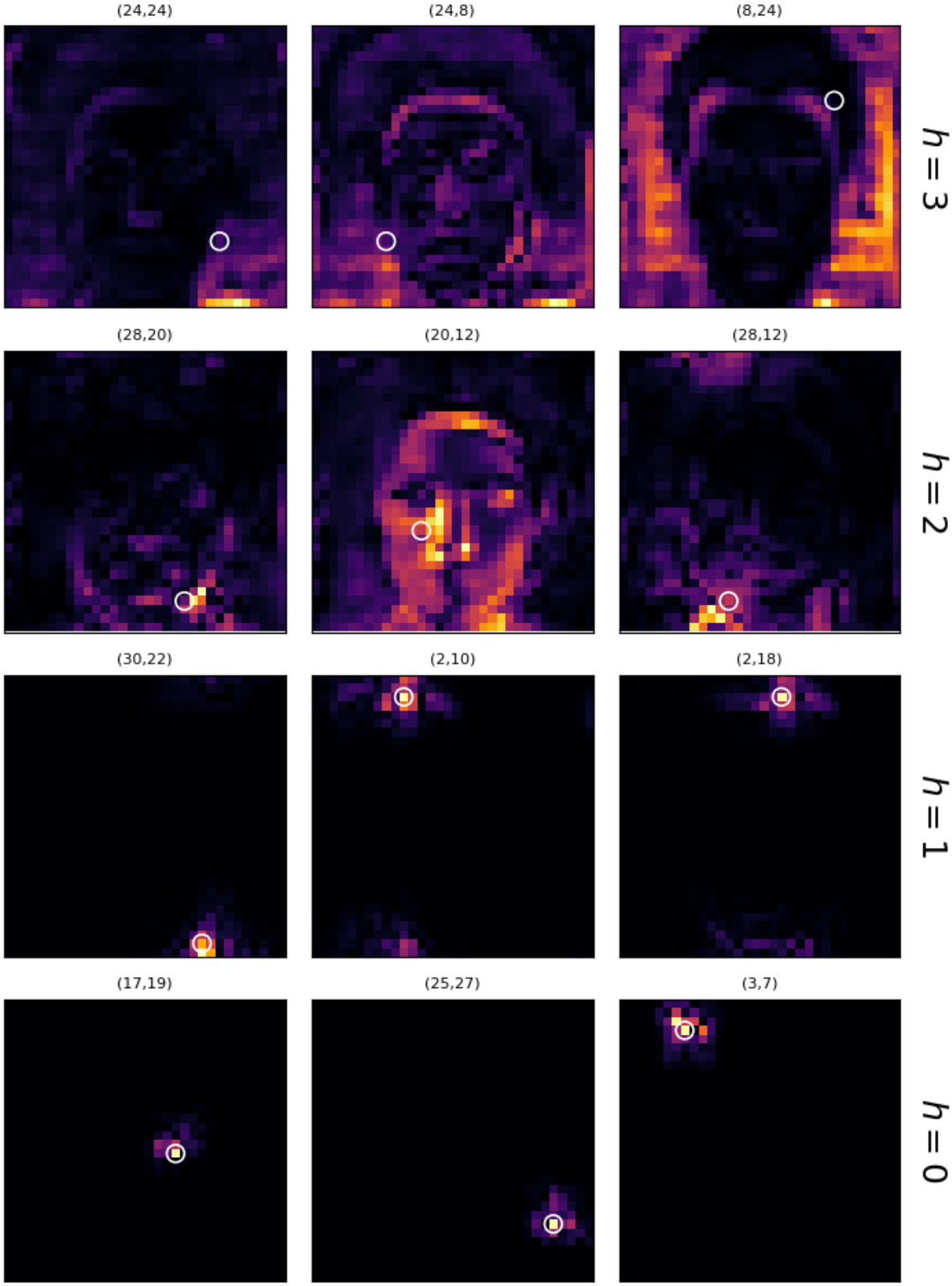

Figure 7: Receptive Fields for Gaussian RG-Flow trained on CelebA.

5 Conclusion

We have successfully implemented the RG-Flow model and demonstrated that it effectively learns hierarchical representations at multiple scales. However, the visual quality of the generated images remains limited. This is likely attributable to our decision to substantially increase the batch size from the original implementation, which has resulted in only partial convergence of the model. The two prior distributions preform approximately the same. We hypothesis that this is due to noise dominating the loss surface and overcoming any regularisation effects. Future work should address this by exploring more extensive training or reverting to a smaller batch size to improve model performance and image quality.

References

[1] Hu HY, Wu D, You YZ, Olshausen B, Chen Y. RG-Flow: a hierarchical and explainable flow model based on renormalization group and sparse prior. Machine Learning: Science and Technology. 2022 aug;3(3):035009. Available from: https://dx.doi.org/10.1088/2632-2153/ac8393.

[2] Kadanoff LP. Scaling laws for ising models near T_c. Physics Physique Fizika. 1966 Jun;2:263-72. Available from: https://link.aps.org/doi/10.1103/PhysicsPhysiqueFizika.2.263.

[3] Wilson KG. Renormalization Group and Critical Phenomena. I. Renormalization Group and the Kadanoff Scaling Picture. Phys Rev B. 1971 Nov;4:3174-83. Available from: https://link.aps.org/doi/10.1103/PhysRevB.4.3174.

[4] Goldenfeld N. Lectures on Phase Transitions and the Renormalization Group. Frontiers in Physics. Addison-Wesley; 1992. Chapter 5.6.

[5] Pagani C, Sonoda H. Geometry of the theory space in the exact renormalization group formalism. Phys Rev D. 2018 Jan;97:025015. Available from: https://link.aps.org/doi/10.1103/PhysRevD.97.025015.

[6] Cardy JL. Scaling and renormalization in statistical physics. Cambridge lecture notes in physics. Cambridge: Cambridge Univ. Press; 1996. Available from: https://cds.cern.ch/record/318508.

[7] Bell JS. On the Einstein Podolsky Rosen paradox. Physics Physique Fizika. 1964 Nov;1:195-200. Available from: https://link.aps.org/doi/10.1103/PhysicsPhysiqueFizika.1.195.

[8] Evenbly G, Vidal G. Real-Space Decoupling Transformation for Quantum Many-Body Systems. Phys Rev Lett. 2014 Jun;112:220502. Available from: https://link.aps.org/doi/10.1103/PhysRevLett.112.220502.

[9] Vidal G. Class of Quantum Many-Body States That Can Be Efficiently Simulated. Physical Review Letters. 2008 Sep;101(11). Available from: http://dx.doi.org/10.1103/PhysRevLett.101.110501.

[10] Marzlin KP, Bartlett SD, Sanders BC. Entanglement gauge and the non-Abelian geometric phase with two photonic qubits. Phys Rev A. 2003 Feb;67:022316. Available from: https://link.aps.org/doi/10.1103/PhysRevA.67.022316.

[11] Molina-Vilaplana J. Connecting Entanglement Renormalization and Gauge/Gravity dualities; 2011. Available from: https://arxiv.org/abs/1109.5592.

[12] Rançon A, Rançon U, Ivek T, Balog I. Dreaming up scale invariance via inverse renormalization group; 2025. Available from: https://arxiv.org/abs/2506.04016.

[13] Sheshmani A, You YZ, Fu W, Azizi A. Categorical representation learning and RG flow operators for algorithmic classifiers. Machine Learning: Science and Technology. 2023 feb;4(1):015012. Available from: https://dx.doi.org/10.1088/2632-2153/acb488.

[14] Prince SJD. Understanding Deep Learning, Chapter 20. The MIT Press; 2023. Available from: http://udlbook.com.

[15] Lee J, Bahri Y, Novak R, Schoenholz SS, Pennington J, Sohl-Dickstein J. Deep Neural Networks as Gaussian Processes; 2018. Available from: https://arxiv.org/abs/1711.00165.

[16] Schoenholz SS, Gilmer J, Ganguli S, Sohl-Dickstein J. Deep Information Propagation; 2017. Available from: https://arxiv.org/abs/1611.01232.

[17] Poole B, Lahiri S, Raghu M, Sohl-Dickstein J, Ganguli S. Exponential expressivity in deep neural networks through transient chaos; 2016. Available from: https://arxiv.org/abs/1606.05340.

[18] Yaida S. Non-Gaussian processes and neural networks at finite widths; 2020. Available from: https://arxiv.org/abs/1910.00019.

[19] Roberts DA, Yaida S, Hanin B. The Principles of Deep Learning Theory: An Effective Theory Approach to Understanding Neural Networks. Cambridge University Press; 2022. Available from: http://dx.doi.org/10.1017/9781009023405.

[20] Bény C. Deep learning and the renormalization group; 2013. Available from: https://arxiv.org/abs/1301.3124.

[21] Tangpanitanon J, Mangkang C, Bhadola P, Minato Y, Angelakis DG, Chotibut T. Explainable natural language processing with matrix product states. New Journal of Physics. 2022 May;24(5):053032. Available from: http://dx.doi.org/10.1088/1367-2630/ac6232.

[22] Chen H, Barthel T. Machine Learning With Tree Tensor Networks, CP Rank Constraints, and Tensor Dropout. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2024 Dec;46(12):7825–7832. Available from: http://dx.doi.org/10.1109/TPAMI.2024.3396386.

[23] Metz F, Bukov M. Self-correcting quantum many-body control using reinforcement learning with tensor networks. Nature Machine Intelligence. 2023 Jul;5(7):780–791. Available from: http://dx.doi.org/10.1038/s42256-023-00687-5.

[24] Fjelde T, Mathieu E, Dutordoir V. An Introduction to Flow Matching; 2024. Available from: https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html.

[25] Zhang H, Dauphin YN, Ma T. Fixup Initialization: Residual Learning Without Normalization; 2019. Available from: https://arxiv.org/abs/1901.09321.

[26] Karras T, Laine S, Aittala M, Hellsten J, Lehtinen J, Aila T. Analyzing and Improving the Image Quality of StyleGAN. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020. Computer Vision Foundation / IEEE; 2020. p. 8107-16. Available from: https://openaccess.thecvf.com/content_CVPR_2020/html/Karras_Analyzing_and_Improving_the_Image_Quality_of_StyleGAN_CVPR_2020_paper.html.

[27] Verine A, Negrevergne B, Rossi F, Chevaleyre Y. On the expressivity of bi-Lipschitz normalizing flows; 2024. Available from: https://arxiv.org/abs/2107.07232.

[28] Liu Z, Luo P, Wang X, Tang X. Deep Learning Face Attributes in the Wild. In: Proceedings of International Conference on Computer Vision (ICCV); 2015. .

AI declaration

All code was created in VSCode with github copilot enabled. Chatgpt was used to help create and improve the quality of plots as well as to proofread this report. Some initial training was done on Google Colab with Gemini enabled.