
Rethinking Conventional Interpretations of Deep Neural Networks

Anonymous Author(s)

Affiliation

Address

email

Abstract

Deep Neural networks (DNNs) despite their tremendous success are still considered as black boxes due to the presence of non-linear activations. In this work, we attempt to disentangle the non-linearity in DNNs so as to achieve feature interpretability. Recent work by Lakshminarayanan and Singh [2020] propose Deep Gated Networks (DGNs) which decompose the DNN into simpler parts, namely a gating network and a weight network. Such a network was used to argue that the most critical information is learnt in the gates. We use DGNs as a starting point to propose a feature interpretable counterpart to DNNs, namely Deep Linearly Gated Network (DLGN). DLGNs learn simple linear mappings of features (which are amenable to interpretations) which are then used to lift the input to a high dimensional representation where a linear function is learnt using weights. Using novel theoretical insights into the kernels associated with DLGNs, we argue that DLGNs completely disentangle the weight learning from the unlifted input, thus making the weights unnecessary for feature interpretability. Extensive experimental results on CIFAR-10 and CIFAR-100 data-sets comparing DLGN version of state of the art architectures with DGNs and DNNs show that DLGNs manage the interpretability-accuracy trade-off in a much better fashion.

1 Introduction

Despite their success deep neural networks (DNNs) are still largely considered as black boxes. The main issue is that in each layer of a DNN, the linear computation, i.e., multiplication by the weight matrix and the non-linear activations are entangled. The conventional view is that such entanglement is the key to success of DNNs, in that (a) it allows DNNs to learn sophisticated features in a layer-by-layer manner and (b) a linear function of the learnt features in the final layer predicts the output. The major shortcoming of this view is that it does not lend to simple interpretations of the learnt features due to the entangled non-linear nature of the feature map. We refer to this as the *lack of feature interpretability* of DNNs. Note that in this view the weights in the last layer are unnecessary for feature interpretability as their role is to simply linearly combine the learnt features.

In this work, we attempt to understand how much of this non linear entanglement can be disentangled without affecting the performance. The reason for this exercise is to see if the DNN architecture can naturally be separated into simple components - a part that contributes to *feature interpretability* and a part that is unnecessary for feature interpretability. Naively, one can completely disentangle the non-linearity by making the entire network linear (Deep/Shallow Linear Networks). Such networks will have completely feature interpretability. However, it is common wisdom that doing this adversely affects the performance and leads to a loss of the essential power of DNNs that comes due to non-linearity. Thus, a natural question is the following:

- *How can one disentangle the non-linearity in DNN so as to achieve feature interpretability?*

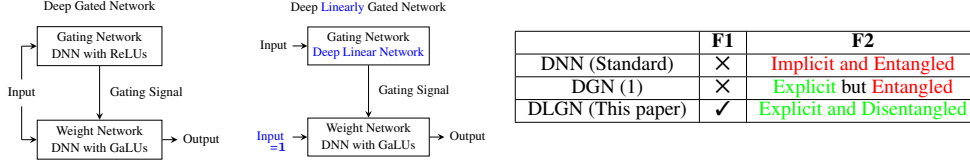


Figure 1: The figures on the left show the architectural difference between DGN of Lakshminarayanan and Singh [2020] and DLGN (this paper). Table on the right compares the functional differences among DNN, DGN, DLGN in terms of the (i) **F1**- linear feature learning and (ii) **F2**- linear weight learning in the lifted space. Refer text for an explanation of implicit/explicit and disentangled/entangled.

As we will see, answering this question requires a fundamental rethinking of how we view DNNs. Specifically, one must move away from the layer-by-layer non-linear feature learning + final layer linear weight learning viewpoint. To appreciate this, notice that to achieve feature interpretability, one needs to make the features linear functions of the input. The role of the weights is as usual to linearly combine the learnt features. However, naively doing this will lead to learning a overall linear function which is not so useful in practice. Thus, a *lifting* operation is needed that lifts the learnt linear features into a high dimensional space where the weights then act linearly. In DNNs, this lifting is entangled and happens implicitly thus leading to difficulty in interpreting the learnt feature maps (see next paragraph). Our goal in this paper to develop a novel architecture which performs the following two functions in an explicit and disentangled manner: (i) **F1**- linear feature learning and (ii) **F2**- linear weight learning in the lifted space. Such an architecture would then enjoy both feature interpretability and the power of non-linearity.

Dual View of DNN: To achieve our goal we turn towards the recently developed dual view of DNNs with rectified linear units (ReLUs) Lakshminarayanan and Singh [2020]. The dual view looks at DNNs via the lens of *paths, gates and weights*. As per the dual view, the DNN is seen to first generate the gates ('on/off' states of the ReLUs) in a layer-by-layer fashion. The DNN then lifts the information in the input and the gates to a higher dimensional feature (called as neural path feature) and the weights (encoded as neural path values) can be seen to linearly combine the higher dimensional neural path features¹. As the same set of weights is used to generate the gates as well as the linear combination in the lifted space, we can see that the DNN (i) does not perform **F1** and (ii) performs **F2** in an implicit and entangled manner.

Deep Gated Networks: Lakshminarayanan and Singh [2020] aimed to understand the role of gates by separating them from the weights. Towards this they introduced a Deep Gated Network (DGN) in which a gating network generates the gates which are then used as external masks in a different network called the weight network which produces the output. Through experiments with the DGN, it was shown that learning the gates is critical for performance and disentangling the gates and weights does not hurt performance.

We use DGN as a starting point to achieve our desired goal. In a DGN, while the weight network explicitly performs linear operation in the lifted space (i.e., **F2**), it is nevertheless entangled with the input (Figure 1). To achieve disentanglement, we provide a constant 1 (i.e., tensor of all ones) as input to the weight network (the actual input is fed only to the gating network). We verify experimentally that providing constant 1 input to the weight network does not degrade performance. Thus DGN with the constant 1 input to weight network performs **F2** in an explicit and disentangled manner as desired. However, the gating network in the DGN is a DNN with ReLUs which violates **F1**.

Deep Linearly Gated Networks: In this paper, we propose a Deep Linearly Gated Network (DLGN), which is a DGN (i) whose gating network is a deep linear network, (ii) whose weight network is fed a constant 1 input. DLGN therefore performs both **F1** and **F2** in an explicit and disentangled manner. DLGNs are the feature interpretable counterparts of DGNs/DNN wherein the gating network being linear is feature interpretable and the weight network (with constant 1 as input) is unnecessary for feature interpretability. We show experimentally that DLGNs of state-of-the-art architectures achieve competitive performance on standard datasets. A summary of ideas in the paper is in Figure 1.

¹Neural path feature and value are vectors whose dimension is equal to the total number of paths in the network

2 Preliminaries

In this section, we discuss relevant ideas from Lakshminarayanan and Singh [2020].

2.1 Dual View for DNNs with ReLUs: Implicit Linear Learning in Lifted Space

Let us consider a fully connected DNN with ‘ d ’ layers and ‘ w ’ hidden units in each layer, let $x = (x(1), \dots, x(d_{\text{in}})) \in \mathbb{R}^{d_{\text{in}}}$ be the input and let $\Theta \in \mathbb{R}^{d_{\text{net}}}$ be the weights of the DNN. In the dual view, the computations are broken down path-by-path. A path starts at an input node, passes through a weight and a ReLU in each layer and ends at the output node. For each path, the input and the gates (‘0/1’ state of the ReLUs) are encoded in a *lifted high dimensional feature called the neural path feature vector* - $\phi_{\Theta}(x) \in \mathbb{R}^P$ and the weights are encoded in a *neural path value vector* - $v_{\Theta}(x) \in \mathbb{R}^P$, where P is the total number of paths. For a path p that starts at input node i , the NPF $\phi_{\Theta}(x, p)$ is the product of $x(i)$ and the gates in the path, and the NPV $v_{\Theta}(p)$ is the product of the weights in the path. The output $\hat{y}_{\Theta}(x)$ of the DNN is then

$$\hat{y}_{\Theta}(x) = \langle \phi_{\Theta}(x), v_{\Theta} \rangle = \sum_{p=1}^P \phi_{\Theta}(x, p) v_{\Theta}(p) \quad (\text{F2: Linear in Lifted Space}) \quad (1)$$

Remark: The lifted high dimensional space is the space of paths, and ‘implicit and entangled’ denotes that the same set of weights compute the gates as well as the inner product in the lifted space.

2.2 Deep Gated Network (DGN): Explicit Linear Learning in Lifted Space

Lakshminarayanan and Singh [2020] proposed the Deep Gated Network (DGN) which is a setup to separate the gates from the weights. Consider a DNN with ReLUs with weights $\Theta \in \mathbb{R}^{d_{\text{net}}}$. The DGN *corresponding* to this DNN (Figure 1) has two networks of *identical architecture* (to the DNN) namely the gating network and the weight network with distinct weights $\Theta^f \in \mathbb{R}^{d_{\text{net}}}$ and $\Theta^v \in \mathbb{R}^{d_{\text{net}}}$. The gating network has ReLUs which turn ‘on/off’ based on their pre-activation signals, and the weight network has gated linear units (GaLUs) [Fiat et al., 2019, Lakshminarayanan and Singh, 2020], which multiply their respective pre-activation inputs by the external gating signals provided by the gating network. Since both the networks have identical architecture, the ReLUs and GaLUs in the respective networks have a one-to-one correspondence. Gating network realises $\phi_{\Theta^f}(x)$ by turning ‘on/off’ the corresponding GaLUs in the weight network. The weight network realises v_{Θ^v} and computes the output $\hat{y}_{\text{DGN}}(x) = \langle \phi_{\Theta^f}(x), v_{\Theta^v} \rangle$. Using extensive experiments in the DGN, Lakshminarayanan and Singh [2020] demonstrated that the learning the gates is critical for performance and disentangling the gates and weights does not hurt performance.

Remark: In a DGN, the weight network explicitly performs the linear learning in the lifted space of paths, i.e., inner product $\hat{y}_{\text{DGN}}(x) = \langle \phi_{\Theta^f}(x), v_{\Theta^v} \rangle$. However, the gating and the weight network are still entangled by the common *unlifted* input (please refer to the DGN in Figure 1).

2.3 Linear Learning in Lifted Space = Neural Path Kernel

Neural Tangent Kernel: An important kernel associated with a DNN is its *neural tangent kernel* (NTK), which, for a pair of input examples $x, x' \in \mathbb{R}^{d_{\text{in}}}$, and network weights $\Theta \in \mathbb{R}^{d_{\text{net}}}$, is given by:

$$\text{NTK}(x, x') = \langle \nabla_{\Theta} \hat{y}(x), \nabla_{\Theta} \hat{y}(x') \rangle, \quad \text{where}$$

$\hat{y}_{\Theta}(\cdot) \in \mathbb{R}$ is the DNN output. Prior works [Jacot et al., 2018, Arora et al., 2019, Cao and Gu, 2019] showed that as the DNN’s width goes to infinity, the NTK matrix converges to a limiting deterministic matrix NTK_{∞} , and training an infinitely wide DNN is equivalent to a kernel method with NTK_{∞} .

Neural Path Kernel (NPK): Lakshminarayanan and Singh [2020] show that under mild conditions, for a fully connected DGN, learning the weight network with a fixed gating network is equivalent to a kernel method with the neural path kernel (NPK), which is the Gram matrix of the neural path features given by $\text{NPK}(x, x) = \langle \phi_{\Theta}(x), \phi_{\Theta}(x') \rangle$. Specifically, in the limit of infinite width, assuming all the weights are initialized from the set $\{\sigma, -\sigma\}$ with equal probability, they show the following:

$$\text{NTK}(x, x') \rightarrow d \cdot \sigma^{2(d-1)} \cdot \text{NPK}(x, x'), \quad \text{as } w \rightarrow \infty$$

3 Neural Path Kernel : Unnoticed Structural Properties

As remarked in the previous section, the weight network of the DGN performs **F2** in an explicit manner. However, the issue is that the weight network of the DGN is not disentangled from the input. Hence, our first goal will be to disentangle the input from the explicit linear learning in lifted space. We achieve this goal in Section 4. To build up towards this goal, we list down previously unnoticed the structural properties of the NPK (since it characterises the learning in the weight network albeit in the infinite width regime). We list down and discuss these properties in this section, and explain how they help us in the overall goal in Section 4.

We consider the NPK for (i) fully connected, (ii) convolutional and (iii) residual architectures. Lakshminarayanan and Singh [2020] developed the dual view only for the full connected case. In this paper, we extend the definitions (see appendix) of the neural path feature and value to convolutional and residual architectures – expression for NPK^{CONV} and NPK^{RES} in Proposition 3.1 are novel and have not appeared before in literature.

Convolutional Architecture: We consider circular convolution with global average pooling (see appendix). In what follows, let the circular rotation of vector $x \in \mathbb{R}^{d_{\text{in}}}$ by ‘ r ’ co-ordinates be defined as $\text{rot}(x, r)(i) = x(i + r)$, if $i + r \leq d_{\text{in}}$ and $\text{rot}(x, r)(i) = x(i + r - d_{\text{in}})$ if $i + r > d_{\text{in}}$.

Residual Architecture: We next consider a residual network (ResNet) with ‘ $(b + 2)$ ’ blocks and ‘ b ’ skip connections between the blocks. Each block is a fully connected (FC) network of depth ‘ d_{blk} ’ and width ‘ w ’. There are 2^b many sub-FCNs within this ResNet. Note that the blocks being fully connected is for expository purposes, and the result can be extended for convolutional blocks as well. Let $2^{[b]}$ denote the power set of $[b]$ and let $\mathcal{J} \in 2^{[b]}$ denote any subset of $[b]$. Define the ‘ \mathcal{J}^{th} ’ sub-FCN of the ResNet to be the fully connected network obtained by (i) including block $_j, \forall j \in \mathcal{J}$ and (ii) ignoring block $_j, \forall j \notin \mathcal{J}$. In Proposition 3.1 below, $\text{NPK}_{\mathcal{J}}^{\text{FC}}$ is the NPK of the \mathcal{J}^{th} FCN.

Proposition 3.1. (Structural Properties of NPK) Let $G_l(x)$ be the w -dimensional vector encoding the gating of the w hidden ReLUs in layer l . Let $\Lambda = \text{overlap}(\cdot, x, \text{rot}(x', r))$ be a diagonal matrix whose diagonal entries correspond to the total number of common paths starting from input node i active for both inputs x and x' .

$$\begin{array}{llll}
 & \text{NPK}^{\text{FC}}(x, x') & = & \langle x, x' \rangle \cdot \text{overlap}_{\Theta}(x, x') \quad (\text{Prior Work}) \\
 \text{Permutation Invariance:} & \text{NPK}^{\text{FC}}(x, x') & = & \langle x, x' \rangle \cdot \prod_{l=1}^{(d-1)} \langle G_l(x), G_l(x') \rangle \quad (\text{This Paper}) \\
 \text{Rotational Invariance:} & \text{NPK}^{\text{CONV}}(x, x') & = & \sum_{r=0}^{d_{\text{in}}-1} \langle x, \Lambda \text{rot}(x', r) \rangle \quad (\text{This Paper}) \\
 \text{Ensemble Kernel:} & \text{NPK}^{\text{RES}}(x, x') & = & \sum_{\mathcal{J} \in 2^{[b]}} \text{NPK}_{\mathcal{J}}^{\text{FC}} \quad (\text{This Paper})
 \end{array}$$

Remark: In the fully connected case, $\text{overlap}(i, x, x')$ is identical for all $i = 1, \dots, d_{\text{in}}$, and hence $\langle x, \Lambda x' \rangle$ becomes $\langle x, x' \rangle \cdot \text{overlap}(x, x')$. In convolutional architectures $\text{overlap}(i, x, x')$ may not be identical for all $i = 1, \dots, d_{\text{in}}$.

• **Fully Connected:** The first expression for $\text{NPK}^{\text{FC}}(x, x')$ in Proposition 3.1 is from Lakshminarayanan and Singh [2020]. The second expression is a *straightforward* rewriting of the first expression in terms of the gates. The reason for this rewriting is that the second expression evidently reveals that the NPK is a product kernel with the property of being invariant to permutation of the gating masks. As we will see in Section 4 this property is key to achieve our overall goal.

• **Convolutional:** The right hand side of the expression for NPK^{CONV} can further be written as $\sum_{r=0}^{d_{\text{in}}-1} \sum_{i=1}^{d_{\text{in}}} x(i) \text{rot}(x', r)(i) \text{overlap}(i, x, \text{rot}(x', r))$, where the inner ‘ Σ ’ is the inner product between x and $\text{rot}(x', r)$ weighted by overlap and the outer ‘ Σ ’ covers all possible rotations, which in addition to the fact that all the variables internal to the network rotate as the input rotates, results in the rotational invariance. That said, rotational invariance is not a new observation; it was shown by Li et al. [2019] the NTK for ConvNets with global-average-pooling are rotationally invariant. Importance of NPK^{CONV} in Proposition 3.1 is that it gives a fine grained expression for the kernel by explicitising the gates, a fact which will be used critically in the experiments.

• **Residual:** To the best of our knowledge, NPK^{RES} in Proposition 3.1 is the first expression to explicitise that ResNets have an ensemble structure, where each kernel in the ensemble, i.e., $\text{NPK}_{\mathcal{J}}^{\text{FC}}$ corresponds to one of the 2^b sub-architectures.

4 Achieving F2 in a disentangled and explicit manner

In this section, we take up the sub-goal of ensuring the weight network performs **F2** in an explicit and disentangled manner, and in Section 5 we take up the problem of **F1**, i.e., linear feature learning to achieve our overall goal. To achieve the former sub-goal, we only supply the input to the gating network and a constant all ones vector as input to the weight network. We refer to the resulting network architecture as DGN-ALLONES. From a conventional viewpoint of layer-by-layer learning of sophisticated features, one would expect a significant drop in the performance when the input is made constant. However, we show in Figure 2, this is not the case.

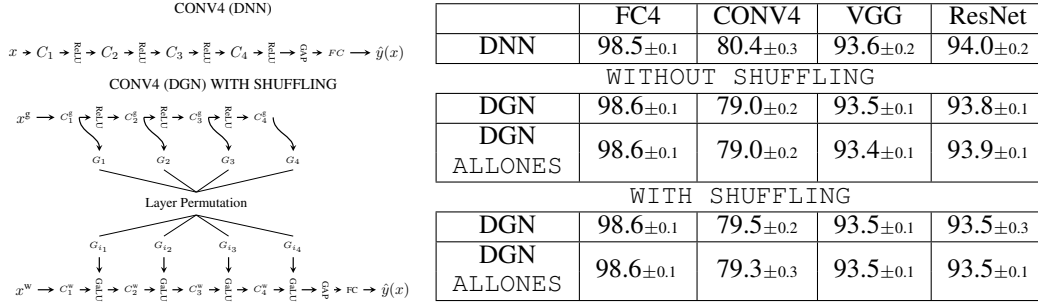


Figure 2: **Left Top:** CONV4 (DNN). **Left Bottom:** CONV4 (DGN) along with mechanism to shuffle the gating masks. Here x^g and x^w denote the input to the gating and weight network respectively. Here the gates G_1, G_2, G_3, G_4 are generated by the gating network and are permuted as $G_{i_1}, G_{i_2}, G_{i_3}, G_{i_4}$ before applying to the weight network. Shuffling of FC4 (DGN) is similar to CONV4 (DGN). Since we have 4 layers, there are 24 possible permutations of the gating masks. For FC4 and CONV4, the results of the models with identity permutation of layers are in two rows below by **WITHOUT SHUFFLING** in the table on the right and the results of the 23 other models are in the two rows below **WITH SHUFFLING** in the table on the right. In the case of VGG and ResNet, we permute the layers in the reverse order in each of the blocks (see appendix for details). In the table on the right, for DGN we set $x^g = x^w = x$ and for DGN-ALLONES we set $x^g = x$, and $x^w = \mathbf{1}$. **Right:** Entries in the first 3 rows and the 3rd column for VGG are averaged over 5 independent runs. The entries in the last column for ResNet are averaged over 3 independent runs. The entries in the first two columns and last two rows are averaged over the 23 models obtained via permuting the layers (here each model is run once). The results in this table are in the setting wherein the gating network of the DGNs are pre-trained. However, the same results also hold even when the gating network is trained from scratch simultaneously alongside the weight network, and this is shown in Table 1. In short, pre-training the gating network is not critical for these results with the constant **1** input and layer shufflings.

Theoretical Justification: We will now explain how the results in Figure 2 follows from the properties of the NPK listed in Proposition 3.1. In the fully connected case, for a constant **1** input, $\langle x, x' \rangle = \langle \mathbf{1}, \mathbf{1} \rangle = d_{\text{in}}$ and the expression on right hand side of the NPK expression in Proposition 3.1 becomes $d_{\text{in}} \cdot \prod_{l=1}^{d-1} \langle G_l(x), G_l(x') \rangle$, i.e., the kernel still has information about the input encoded via the gates. In the case of convolutional networks, the expression in Proposition 3.1 becomes $\sum_{r=0}^{d_{\text{in}}-1} \sum_{i=1}^{d_{\text{in}}} \text{overlap}(i, x, \text{rot}(x', r))$. The key novel insight is that the rotational invariance is not lost and **overlap** matrix measures the correlation of the paths which in turn depends on the correlation of the gates. In the case of ResNets, as the NPK^{RES} of the residual networks is an ensemble, the property of the block level kernel (i.e., of $\text{NPK}^{\text{FC}}/\text{NPK}^{\text{CONV}}$) translates to NPK^{RES} .

4.1 Layer-By-Layer View Fails: Weight network has no value for feature interpretability

We now devise a novel experiment which further shows that the conventional layer-by-layer viewpoint fails for the weight network of DGNs, eventually leading to the conclusion that the weight network does not contribute to feature interpretability and its sole purpose is to explicitly perform **F2**, i.e., linear learning in the lifted space. Towards this, in addition to the all ones input, we now also randomly shuffle the hidden layers i.e., the gating masks are applied post layer shuffling (See Figure 2). If the conventional wisdom about layer-by-layer learning were true, then the performance after layer shuffling must take a big hit. We show in Figure 2 that this is not the case.



































		Input	Layer 1 Filter 1	Layer 1 Filter 2	Layer 2 Filter 1	Layer 2 Filter 2	Layer 3 Filter 1	Layer 3 Filter 2	Layer 4 Filter 1	Layer 4 Filter 2	Test Acc.	Layer-By-Layer Interpretation
	Gating Network										80.4 ± 0.3	✓
DGN-1	Weight Network										79.0 ± 0.2	✗
DGN-2	Weight Network 1 input										79.5 ± 0.2	✗
DGN-3	Weight Network 1 input + Layer Shuffling										79.3 ± 0.3	✗

Figure 3: Shows the hidden layer outputs of 3 different convolutional DGNs namely DGN-1/2/3 (each with 4 hidden layers and 128 filters per layer). Each row has the input image to the network followed by the output of first 2 filters in each of the 4 hidden layers. All 3 DGNs have the same gating network which is shown in the top row. The 3 DGNs differ in the weight network as follows: (i) in DGN-1 in the second row, the weight network is provided with the input image and the layers are not shuffled, (ii) in DGN-2 in the third row, the weight network is provided with 1 as input and the layers are not shuffled, and (iii) in DGN-3 in the final row, the weight network is provided with 1 as input and the layers are shuffled. In the last two rows, the entry in the input image column ‘appears’ blank because it is the image of the 1 input to the weight network.

Theoretical Justification: Recall that in the NPK expression for the fully connected case in Proposition 3.1, $\Pi_{l=1}^{d-1} \langle G_l(x), G_l(x') \rangle$ is permutation invariant, and hence permuting the layers has no effect, in that, it leaves the NPK expression unchanged. Similarly, permuting the layers does not destroy the rotational invariance in Proposition 3.1. This is because, due to circular convolutions all the internal variables of the network rotate as the input rotates. Permuting the layers only affects the ordering of the layers, and does not affect the fact that the gates rotate if the input rotates. As explained previously, ResNet being an ensemble of block level kernels, inherits the permutation invariance property from the blocks (be it fully connected or convolutional).

Remark: In the case of ResNets, even removing layers does not hurt performance. The ensemble behaviour of ResNet and presence of 2^b architectures was observed by Veit et al. [2016], however without any concrete theoretical formalism. Veit et al. [2016] showed empirically that removing single layers from ResNets at test time does not noticeably affect their performance, and yet removing a layer from architecture such as VGG leads to a dramatic loss in performance. ?? can be seen to provide a theoretical justification for this empirical result. The ResNet inherits the invariances of the block level kernel. In addition, the ensemble structure allows to even remove layers. In other words, due to the ensemble structure a ResNet is robust to failures, in particular, the insight is that even if one or many of the kernels in the ensemble are corrupt, the good ones can compensate.

Remark: A qualitative justification of the above results can be found in Figure 3 (see caption for details).

We conclude this section by pointing out that we achieved the sub-goal **F2** namely explicit and disentangled linear learning in lifted space by setting the input to the weight network of the DGN to be a constant 1. We also showed via experimental results that the weight network of the DGN has no value for feature interpretability. However, DGN-ALLONES still violates **F1** due to the non-linearity in the gating network.

5 Achieving F1 and F2 via Deep Linearly Gated Networks

In this section, we achieve our goal of proposing an architecture, which, in an explicit and disentangled manner performs functions (i) **F1**- linear feature learning and (ii) **F2**- linear weight learning in the lifted space. For this, we propose the Deep Linearly Gated Network (DLGN) (see Figure 1) wherein we completely eliminate the non-linearity of the gating network by removing the ReLU activations thereby performing function **F1** and we provide constant 1 input to the weight network thereby performing function **F2** in a disentangled and explicit manner.

Before we explain the results that we obtain for DLGNs, we take a step back and discuss feature interpretability in general. This will help us contextualise our results better. The simplest *interpretable* model is the shallow linear model. In this model, while retaining linearity, one can increase it’s complexity by introducing deep layers. Instead of adding deep layers, if one adds ReLU to a shallow linear model, one gets the well studied ReLU networks with single hidden layer Chizat and Bach

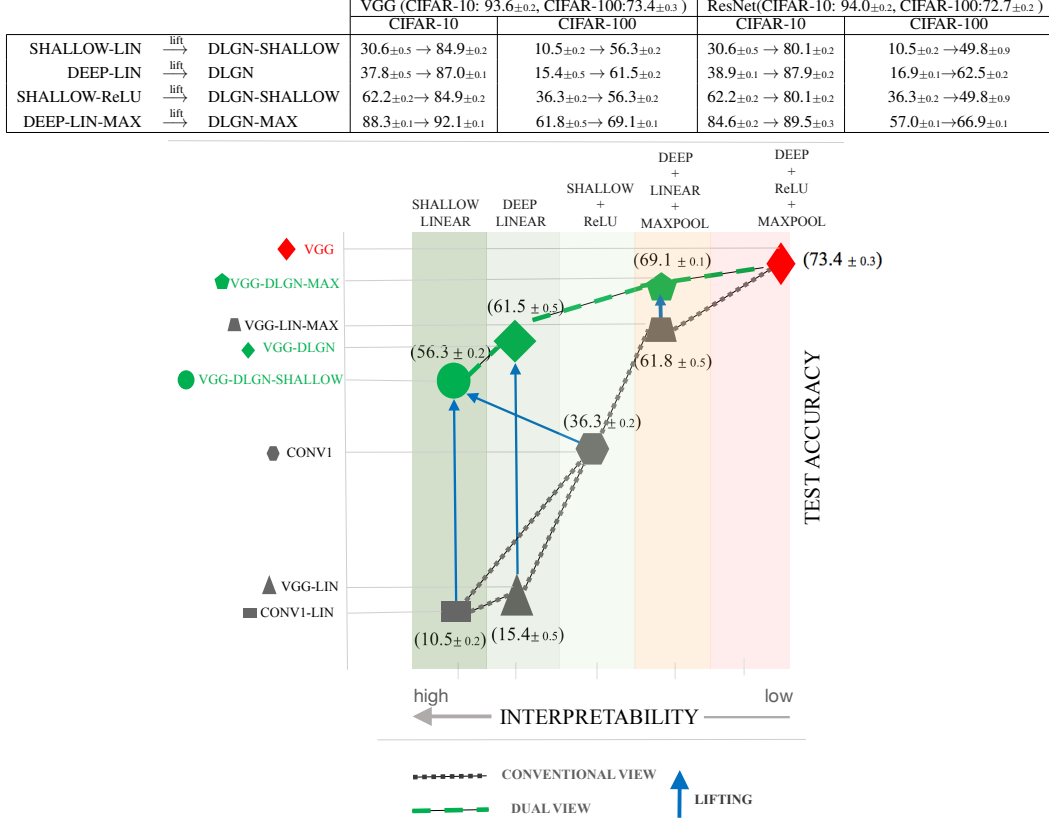


Figure 4: **Bottom:** Shows the interpretability-accuracy tradeoff plot for the various models considered. The gray path shows models that are obtained via DLGN (proposed in this paper) by lifting (see ??). **Top:** Shows the tabular column of test accuracies of all the models. Here, the results for ResNet variants are averaged over 3 independent runs and the results for other models are averaged over 5 independent runs.

[2018], Zhang et al. [2019], Li and Liang [2018], Zhong et al. [2017], Li and Yuan [2017], Ge et al. [2017], Du et al. [2018]. However, the non-linearity in the ReLU makes it less interpretable compared to a deep linear model. As a next step, one might want to add the standard deep learning *tricks* such as striding, average/max-pooling and batch normalisation to a deep linear model. Out of these tricks, only max-pooling is non-linear. Yet, interpretability wise it may be considered a benign form of non-linearity in comparison to ReLU, the reason being that, max-pooling acts directly on the input as opposed to ReLU which acts on the input transformed by a weight vector. As max-pooling is essentially a sub-sampling operation a deep linear model with max-pooling could potentially be interpreted by understanding the linear part and max-pooling separately. Once non-linearity is introduced in the form of say ReLU in deep networks, one typically loses interpretability. Thus one can arrange the increasing levels of interpretable models as follows:

$$\begin{aligned} \text{Shallow Linear} &> \text{Deep Linear} > \text{Shallow} + \text{ReLU} > \\ \text{Deep Linear} + \text{MaxPool} &> \text{Deep} + \text{ReLU} > \text{Deep} + \text{MaxPool} + \text{ReLU} \end{aligned} \quad (2)$$

The conventional viewpoint is that as we gain interpretability, we lose accuracy in proportional measure. In practice then one resorts to choosing accuracy over interpretability and then attempts to interpret/explain the decisions made by the model in a post-hoc fashion Ribeiro et al. [2016], Selvaraju et al. [2017], Chattopadhyay et al. [2018]. However, several recent works have pointed out the need for developing built-in interpretable models i.e., making deep models more interpretable and avoiding taking a post-hoc view of the same Rudin [2019], Rudin et al. [2021].

We describe our results in Figure 4 (the table in the top and the plot in the bottom right). The figure lays out the landscape of interpretable models starting from the shallow linear model all the way upto

the standard Deep + Maxpool + ReLU models. We first describe the models considered in detail below.

Models Considered: Our aim is to create one conventional model for each of the cases in the inequality in Equation (2) and the corresponding DLGN counterpart. **CONV1** has 1 convolutional layer with 512 filters, followed by ReLU, global-average-pooling and output layer (this is the shallow + ReLU model). **CONV1-LIN** is same as **CONV1** with the ReLU activations replaced by identity activation (this is the shallow linear model). **VGG-LIN** is same as VGG with average pooling instead of max-pooling (to ensure linearity) and ReLUs replaced by identity activation. **VGG-LIN-MAX** is VGG (i.e., with max-pooling) and ReLUs replaced by identity activation. **VGG-DLGN** is the DLGN counterpart of VGG (the architecture is in Figure 7 in the appendix). **VGG-DLGN-SHALLOW** is obtained by modifying the gating network of VGG-DLGN as follows: the pre-activations to the gates are generated by d shallow linear convolutional networks (the architecture is in Figure 8 in the appendix). **VGG-DLGN-MAX** is same as VGG-DLGN, however with max-pooling instead of average pooling.

Conventional thinking to understand interpretability-accuracy tradeoff would suggest increasing model complexity one step at a time starting right from shallow linear networks i.e., moving along one of the gray paths in Figure 4. The proposed model, DLGN and it’s corresponding variants for each of the cases, by exploiting *lifting*, outperforms the conventional models at every stage. This corresponds to the green path in Figure 4. By taking the green path, one retains as much interpretability as their counterpart models in the conventional way of thinking while still gaining accuracy. This is the primary advantage of DLGNs.

Explicit Lifting: It is interesting to note that there is a significant gain due to lifting from the Shallow Linear Model to the DLGN-Shallow model. The gain decreases as one adds deep layers followed by MaxPool. This is expected as the effect of the non-linear, non-interpretable weights network is most pronounced when the original model that it is compared against is linear (and shallow). As one introduces some form of non-linearity (MaxPool or ReLU), the lifting helps lesser.

Deep Linear + MaxPool: We observe that adding max-pool to deep linear models results in significant boost in performance (VGG/ResNet-LIN-MAX performs better than VGG/ResNet-LIN). It is worth noting that dual lifting of VGG-LIN-MAX to VGG-DLGN-MAX further boosts the performance achieving accuracies of 69.1% and 92.1% on CIFAR-10 and CIFAR-100 respectively, which are within 4.3% and 1.5% of the performance of VGG on the respective datasets. As discussed before, VGG-DLGN-MAX is worthy of further interpretation by separately understanding the linear part and max-pooling. Also, it would be an interesting future study to understand the reason for the boost in performance when max-pooling is added to a deep linear model.

Deep Linear and Shallow Linear: While the representation power of the deep and shallow linear networks is the same, the difference between them is that of optimisation. We observe that both these models perform poorly in both CIFAR-10 as well as CIFAR-100. Since our main focus has been in interpretable models that also achieve reasonable test accuracy, we did not probe further into these two models. Nevertheless, deep linear networks have been studied well in the literature, especially in connection to the optimisation of deep networks Shamir [2019], Du and Hu [2019], Saxe et al. [2013], Ji and Telgarsky [2018].

SHALLOW-ReLU vs DLGN-SHALLOW: Note that the gating network of the DLGN-SHALLOW comprises of several shallow linear networks in parallel, each which in turn trigger the gates of a layer in the corresponding weight network. Thus if one dual lifts a SHALLOW-ReLU model we obtain a DLGN-SHALLOW model (as seen in Figure 4). We observe that VGG-DLGN-SHALLOW performs better than CONV1. Note that VGG-DLGN-SHALLOW is better than CONV1 both in terms of interpretability and accuracy.

Remark: The trend for the ResNet variants also follows in a similar manner as VGG variants (see table on top of Figure 4). We also observe that the performance gap between the various models is less on CIFAR-10 and more pronounced in CIFAR-100, which is expected since CIFAR-100 (with 100 classes) is a harder dataset than CIFAR-10 (with 10 classes).

6 Related Works

We now discuss prior works most relevant to our work below.

Kernels: Several works have examined theoretically as well as empirically two important kernels associated with a DNN namely its Neural Tangent Kernel (NTK) based on the correlation of the gradients and the conjugate kernel based on the correlation of the outputs [Fan and Wang, 2020, Geifman et al., 2020, Liu et al., 2020, Chen et al., 2020, Xiao et al., 2020, Jacot et al., 2018, Arora et al., 2019, Novak et al., 2018, Lee et al., 2018, 2020]. In contrast, the NPK is based on the correlation of the gates. We do not build pure-kernel method with NPK, but use it as an aid to rethinking the conventional ‘hidden layer’ interpretation of finite width DNNs.

Capacity: Our experiments on destruction of layers, and providing constant 1 input are direct consequences of the insights from dual view theory. These are not explained by mere capacity based studies showing DNNs are powerful to fit even random labelling of datasets [Zhang et al., 2016].

Interpretability: Rudin [2019] provides several compelling arguments for why it is important to build models that are interpretable by design as opposed to seeking *post-hoc* explanations for decisions of non-interpretable models. Rudin et al. [2021] discusses the grand challenges in interpretability. The DLGN in our paper resonates with the following aspect discussed in Rudin [2019], Rudin et al. [2021], i.e., most times the simpler models that are used to explain the decision of the more complicated non-interpretable models do not mimic the computations, thereby the explanations are not faithful which is an issue. The DLGN in our paper does not suffer from this issue because, the DLGN is obtained by rearranging the computations of a DNN with ReLUs in a mathematically principled manner.

7 Conclusion

In this work, we set out to explicitly disentangle DNNs into simpler parts with the goal of achieving feature interpretability. We proposed Deep Linearly Gated Network - a novel architecture that builds on recently proposed Deep Gated Networks. With novel theoretical insights backed by strong experimental evidence, we conclude that DLGNs are perhaps the simplest architecture that enjoys feature interpretability without compromising on accuracy. We believe and hope this work would be a starting point for the interpretable deep learning community to further explore DLGNs. Several future directions are possible including taking a closer domain specific look at the learnt features in the gating network, understanding DLGNs from an adversarial attack point of view, understanding more complicated architectures such as BERT, transformers, etc using the DLGN lens among others.

References

- Chandrashekar Lakshminarayanan and Amit Vikram Singh. Neural path features and neural path kernel: Understanding the role of gates in deep learning. *Advances in Neural Information Processing Systems*, 33, 2020.
- Jonathan Fiat, Eran Malach, and Shai Shalev-Shwartz. Decoupling gating from linearity. *CoRR*, abs/1906.05032, 2019. URL <http://arxiv.org/abs/1906.05032>.
- Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.
- Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8139–8148, 2019.
- Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 10835–10845, 2019.
- Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019.
- Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *arXiv preprint arXiv:1605.06431*, 2016.

348 Lenaic Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized
349 models using optimal transport. *arXiv preprint arXiv:1805.09545*, 2018.

350 Xiao Zhang, Yaodong Yu, Lingxiao Wang, and Quanquan Gu. Learning one-hidden-layer relu
351 networks via gradient descent. In *The 22nd international conference on artificial intelligence and*
352 *statistics*, pages 1524–1534. PMLR, 2019.

353 Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient
354 descent on structured data. *arXiv preprint arXiv:1808.01204*, 2018.

355 Kai Zhong, Zhao Song, Prateek Jain, Peter L Bartlett, and Inderjit S Dhillon. Recovery guarantees
356 for one-hidden-layer neural networks. In *International conference on machine learning*, pages
357 4140–4149. PMLR, 2017.

358 Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation.
359 *arXiv preprint arXiv:1705.09886*, 2017.

360 Rong Ge, Jason D Lee, and Tengyu Ma. Learning one-hidden-layer neural networks with landscape
361 design. *arXiv preprint arXiv:1711.00501*, 2017.

362 Simon Du, Jason Lee, Yuandong Tian, Aarti Singh, and Barnabas Poczos. Gradient descent learns
363 one-hidden-layer cnn: Don’t be afraid of spurious local minima. In *International Conference on*
364 *Machine Learning*, pages 1339–1348. PMLR, 2018.

365 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the
366 predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference*
367 *on knowledge discovery and data mining*, pages 1135–1144, 2016.

368 Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh,
369 and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localiza-
370 tion. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626,
371 2017.

372 Aditya Chattopadhyay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-
373 cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018*
374 *IEEE winter conference on applications of computer vision (WACV)*, pages 839–847. IEEE, 2018.

375 Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use
376 interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

377 Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong.
378 Interpretable machine learning: Fundamental principles and 10 grand challenges. *arXiv preprint*
379 *arXiv:2103.11251*, 2021.

380 Ohad Shamir. Exponential convergence time of gradient descent for one-dimensional deep linear
381 neural networks. In *Conference on Learning Theory*, pages 2691–2713, 2019.

382 Simon S Du and Wei Hu. Width provably matters in optimization for deep linear neural networks.
383 *arXiv preprint arXiv:1901.08572*, 2019.

384 Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics
385 of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

386 Ziwei Ji and Matus Telgarsky. Gradient descent aligns the layers of deep linear networks. *arXiv*
387 *preprint arXiv:1810.02032*, 2018.

388 Zhou Fan and Zhichao Wang. Spectra of the conjugate kernel and neural tangent kernel for linear-
389 width neural networks. *arXiv preprint arXiv:2005.11879*, 2020.

390 Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Ronen Basri. On the
391 similarity between the laplace and neural tangent kernels. *arXiv preprint arXiv:2007.01580*, 2020.

392 Chaoyue Liu, Libin Zhu, and Mikhail Belkin. On the linearity of large non-linear models: when and
393 why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33, 2020.

- 394 Zixiang Chen, Yuan Cao, Quanquan Gu, and Tong Zhang. A generalized neural tangent kernel
395 analysis for two-layer neural networks. *Advances in Neural Information Processing Systems*, 33,
396 2020.
- 397 Lechao Xiao, Jeffrey Pennington, and Samuel Schoenholz. Disentangling trainability and gener-
398 alization in deep neural networks. In *International Conference on Machine Learning*, pages
399 10462–10472. PMLR, 2020.
- 400 Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A Abolafia,
401 Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional networks with many
402 channels are gaussian processes. In *International Conference on Learning Representations*, 2018.
- 403 Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha
404 Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on*
405 *Learning Representations*, 2018.
- 406 Jaehoon Lee, Samuel S Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak,
407 and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *arXiv*
408 *preprint arXiv:2007.15801*, 2020.
- 409 Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding
410 deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.
- 411 Randall Balestriero et al. A spline theory of deep learning. In *International Conference on Machine*
412 *Learning*, pages 374–383, 2018.
- 413 Randall Balestriero and Richard G Baraniuk. From hard to soft: Understanding deep network
414 nonlinearities via vector quantization and statistical inference. *arXiv preprint arXiv:1810.09274*,
415 2018.
- 416 Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint*
417 *arXiv:1505.00387*, 2015.
- 418 Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2018. URL
419 <https://openreview.net/forum?id=SkBYyZRZ>.
- 420 Joel Veness, Tor Lattimore, Avishkar Bhoopchand, David Budden, Christopher Mattern, Agnieszka
421 Grabska-Barwinska, Peter Toth, Simon Schmitt, and Marcus Hutter. Gated linear networks. *arXiv*
422 *preprint arXiv:1910.01526*, 2019.
- 423 David G Clark, LF Abbott, and SueYeon Chung. Credit assignment through broadcasting a global
424 error vector. *arXiv preprint arXiv:2106.04089*, 2021.
- 425 Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
- 426 Szymon Mikler. Resnets in tensorflow2. [https://github.com/gahaalt/](https://github.com/gahaalt/resnets-in-tensorflow2)
427 [resnets-in-tensorflow2](https://github.com/gahaalt/resnets-in-tensorflow2), 2019.

428 A Shuffling in VGG and ResNet

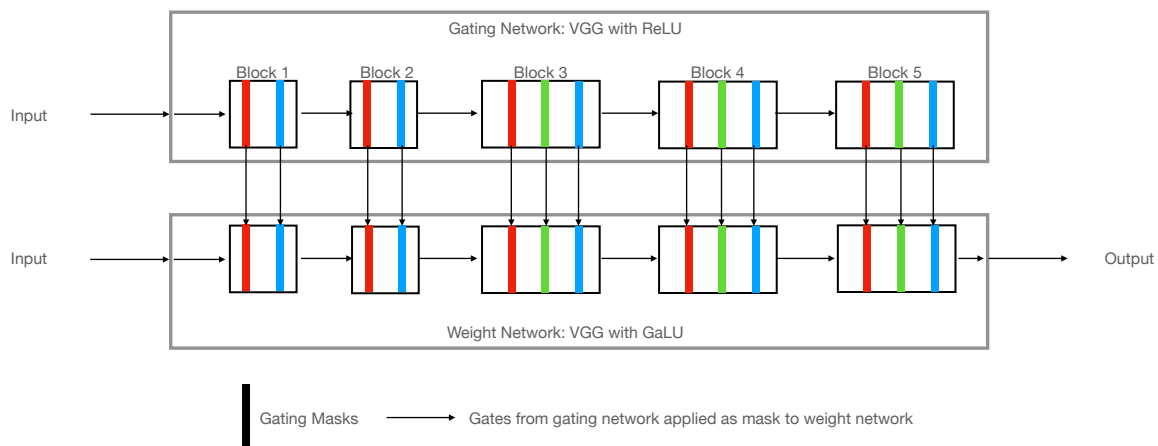


Figure 5: Shows VGG (DGN).

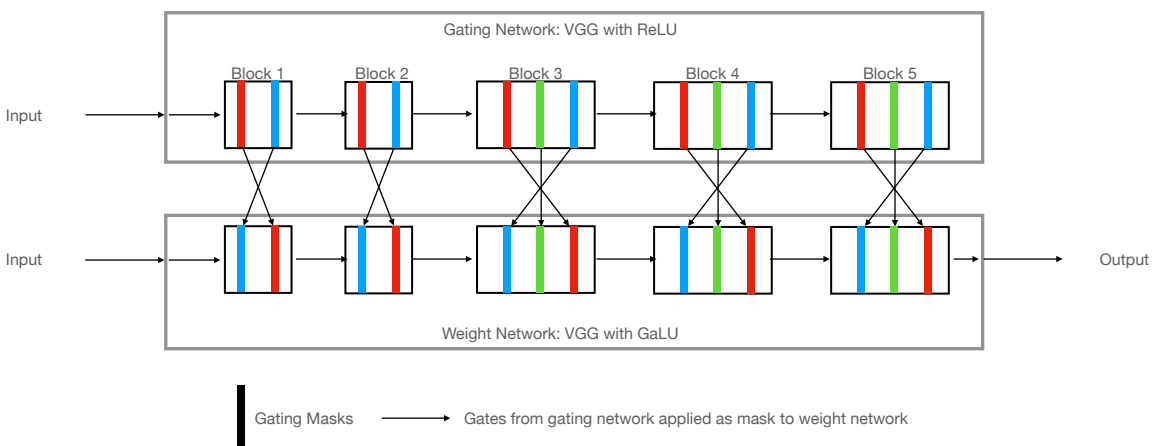


Figure 6: Shows shuffling of gating masks in VGG (DGN) in the last two columns, 3rd row in ???. The shuffling of ResNet (DGN) is similar. VGG contains 5 blocks each with filter sizes (64,128,256,512,512) and number of layers within a block (2,2,3,3,3) respectively. Since the number of filters differ across the blocks we shuffle the gating of the layers within each block in the reverse order. ResNet contains 3 blocks each with filter sizes (16,32,64) and number of layers within a block (36,36,36) respectively. The shuffling is done in a similar manner as VGG, i.e., the gating masks are reversed with each block.

429 A.1 VGG-DLGN

430 A.2 VGG-DLGN-SHALLOW

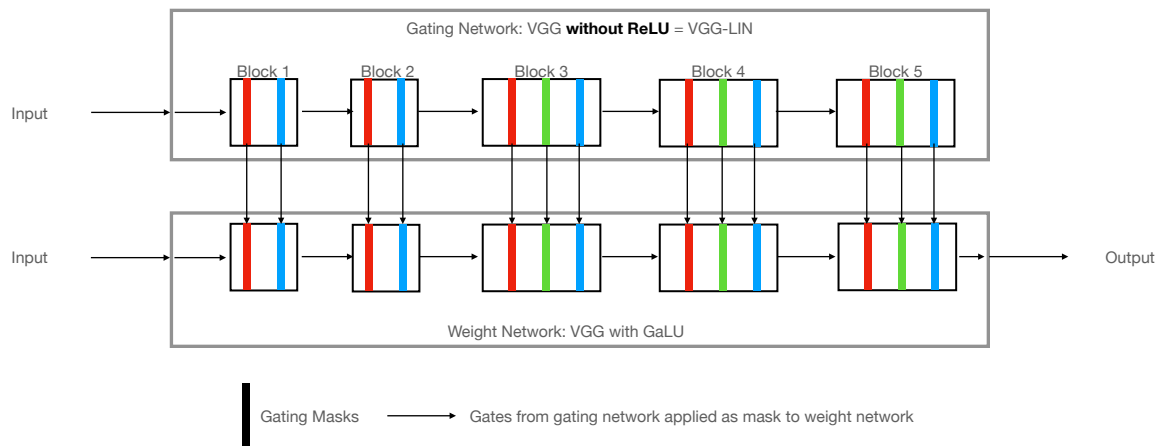


Figure 7: Shows VGG-DLGN

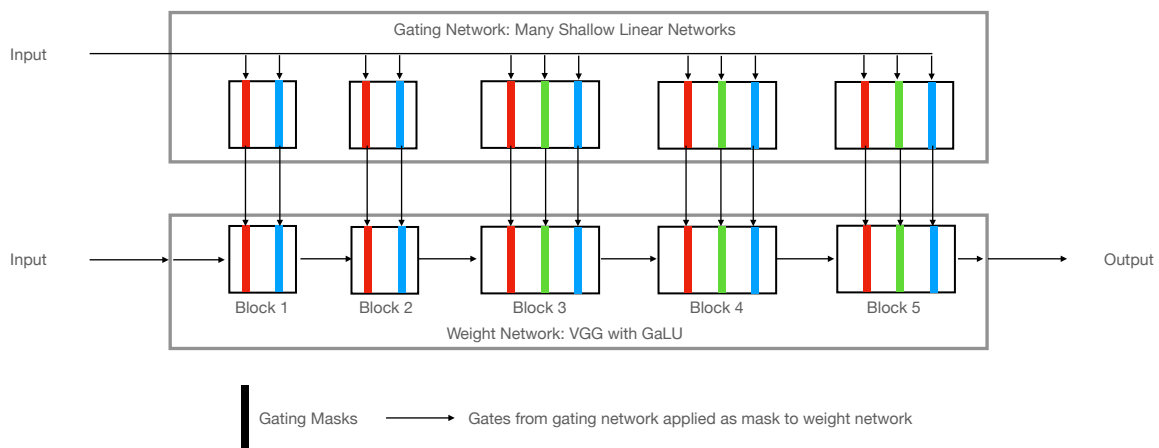


Figure 8: Shows VGG-DLGN-SHALLOW.

B DGN Results without pre-training of the gating network

Architecture	Dataset	DNN	WITHOUT SHUFFLING		WITH SHUFFLING	
			DGN	DGN-ALLONES	DGN	DGN-ALLONES
FC4	MNIST	98.5 \pm 0.1	98.2 \pm 0.1	98.2 \pm 0.1	98.1 \pm 0.1	98.2 \pm 0.1
CONV4	CIFAR-10	80.4 \pm 0.3	77.2 \pm 0.4	77.7 \pm 0.3	77.3 \pm 0.5	77.8 \pm 0.6
VGG	CIFAR-10	93.6 \pm 0.2	93.0 \pm 0.1	93.0 \pm 0.1	92.9 \pm 0.1	92.9 \pm 0.3
ResNet	CIFAR-10	94.0	93.3 \pm 0.2	93.3 \pm 0.1	92.9 \pm 0.1	92.9 \pm 0.2

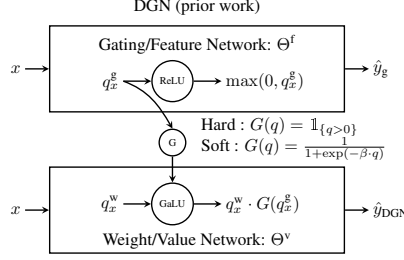
Table 1: ?? showed the results when the gating network was pre-trained. This table shows the results when the gating network of the DGN is trained from scratch simultaneously alongside the weight network. The drop in 2 – 3% of the test accuracy when the gating network of the DGNs are trained from scratch as opposed to a DGN with pre-trained gates has been documented by Lakshminarayanan and Singh [2020] and we observe that the same holds here as well. However, the most critical claim of this paper that providing a constant input as well as shuffling the gating masks does not **further** degrade the performance continues to hold even in the case when the gating network is trained from scratch is what is shown in this table. In other words, evidence provided by ?? to support our critical claims is not dependent on the fact that the gating network was pre-trained.

C Other Related Works

ReLU, Gating, Dual Linearity: A spline theory based on max-affine linearity was proposed in [Balestriero et al., 2018, Balestriero and Baraniuk, 2018] to show that a DNN with ReLUs performs hierarchical, greedy template matching. In contrast, the dual view exploits the gating property to simplify the NTK into the NPK. Gated linearity was studied in [Fiat et al., 2019] for single layered networks, along with a non-gradient algorithm to tune the gates. The main novelty in our work in contrast to the above is that in DLGN the feature generation is linear. We refer to the gating property of the ReLU itself and has no connection to [Srivastava et al., 2015] where gating is a mechanism to regulate information flow. Also, the soft-gating in our work enables gradient flow via the gating network and is different from *Swish* [Ramachandran et al., 2018], which multiplies the pre-activation and sigmoid.

Functional Role of Gating and Connection to Veness et al. [2019]: There are both comparable and non-comparable aspects between our work and that of Gated Linear Networks (GLNs) of Veness et al. [2019]. The *non-comparable aspects* are (i) training: we use backpropagation to train DLGN vs Veness et al. [2019] propose a backpropagation free algorithm, (ii) learning in the gates: gates are learnable in our paper vs gates are fixed and random in Veness et al. [2019], and (iii) gating mechanism: more importantly Veness et al. [2019] presents only a fully connected GLN and convolutional/residual equivalents of GLNs are yet to be explored vs we have presented DLGN counterparts of state-of-the-art convolutional and residual models such as VGG-16 and ResNet-110. The comparable aspects is that the DGN/DLGN and the GLN have both separate gating and both models are essentially *data dependent linear networks*. It is known that GLNs are good in continual learning tasks and given the similarity of the GLN and DLGN, an interesting future research direction would be to investigate the continual learning capabilities of the DLGN.

Connection to Clark et al. [2021]. The connection between our paper and that of Clark et al. [2021] is through the dual view. To elaborate, Clark et al. [2021] deal with Vectorized Non-Negative Networks (VNN) whose special case (for the case when vector dimension equals 1) is a DNN with ReLUs with non-negative weights past first layer. As we understand, the key aspect of VNNs is the non-negativity of the weights past the first layer, and the non-negativity of the derivative of the activation function (which holds for Heaviside step function used in their paper), which enable a *global error vector broadcasting* (GEVB) rule, a non-backpropagation rule. We would like to point out that in VNN, the gating is not separate (however the authors mention in the context of vectorisation unfolding over time that gating can be made separate). We use dual view to disentangle DNNs with ReLUs, whereas, Clark et al. [2021] use the definitions of neural path activity and neural path value to show that the GEVB rule is sign aligned with the gradient of the VNNs. In particular, it is assumed that "for all training examples, each hidden unit has at least one active path with nonzero



value connecting it to the output unit" (see page 17 of Clark et al. [2021]). Thus, the Clark et al. [2021] use the dual view to justify the GEVB rule, we use the dual view to in a much more fundamental way by disentangling the computations in a DNN with ReLUs and reorganizing them in the form of DLGN to improve mathematical interpretability.

D Deep Gated Network : Architecture and Training

The DGN is a setup to separate the gates from the weights. Consider a DNN with ReLUs with weights $\Theta \in \mathbb{R}^{d_{\text{net}}}$. The DGN *corresponding* to this DNN (left diagram in ??) has two networks of *identical architecture* (to the DNN) namely the ‘gating network’ and the ‘weight network’ with distinct weights $\Theta^f \in \mathbb{R}^{d_{\text{net}}}$ and $\Theta^v \in \mathbb{R}^{d_{\text{net}}}$. The ‘gating network’ has ReLUs which turn ‘on/off’ based on their pre-activation signals, and the ‘weight network’ has gated linear units (GaLUs) [Fiat et al., 2019, Lakshminarayanan and Singh, 2020], which multiply their respective pre-activation inputs by the external gating signals provided by the ‘gating network’. Since both the networks have identical architecture, the ReLUs and GaLUs in the respective networks have a one-to-one correspondence. Gating network realises $\phi_{\Theta^f}(x)$ by turning ‘on/off’ the corresponding GaLUs in the weight network. The weight network realises v_{Θ^v} and computes the output $\hat{y}_{\text{DGN}}(x) = \langle \phi_{\Theta^f}(x), v_{\Theta^v} \rangle$. The gating network is also called as the feature network since it realises the neural path features, and the weight network is also called as the value network since it realises the neural path value.

D.1 DGN Training

The following two modes of training have been used in the paper namely **pretrained gates** (PG) and **standalone training** (ST):

Pretrained Gates (PG): The gating network is pre-trained using \hat{y}_f as the output, and then the weights network is frozen, and the weights network is trained with \hat{y}_{DGN} as the output. Hard gating $G(q) = \mathbb{1}_{\{q>0\}}$ is used.

Standalone Training (ST): Both gating and weight network are initialised at random and trained together with \hat{y}_{DGN} as the output. Here, soft gating $G(q) = \frac{1}{1+\exp(-\beta \cdot q)}$ is used to allow gradient flow through gating network. We tried several values of β in the range from 1 to 100, and found the range 4 to 10 to be suitable. We have chosen $\beta = 10$ throughout the experiments.

D.2 DLGN Training

In all the experiments the DLGN is trained in the **standalone training** mode.

E Fully Connected

Here, we present the formal definition for the neural path features and neural path values for the fully connected case in Definition E.1. The layer-by-layer way of expressing the computation in a DNN of width ‘ w ’ and depth ‘ d ’ is given below.

Notation Index maps identify the nodes through which a path p passes. The ranges of index maps \mathcal{I}_l^f , \mathcal{I}_l , $l \in [d-1]$ are $[d_{\text{in}}]$ and $[w]$ respectively. $\mathcal{I}_d(p) = 1, \forall p \in [P^{\text{fc}}]$.

Definition E.1. Let $x \in \mathbb{R}^{d_{\text{in}}}$ be the input to the DNN. For this input,

Input Layer	:	$z_{x,\Theta}(\cdot, 0)$	=	x
Pre-Activation	:	$q_{x,\Theta}(i_{\text{out}}, l)$	=	$\sum_{i_{\text{in}}} \Theta(i_{\text{in}}, i_{\text{out}}, l) \cdot z_{x,\Theta}(i_{\text{in}}, l-1)$
Gating	:	$G_{x,\Theta}(i_{\text{out}}, l)$	=	$\mathbf{1}_{\{q_{x,\Theta}(i_{\text{out}}, l) > 0\}}$
Hidden Layer Output	:	$z_{x,\Theta}(i_{\text{out}}, l)$	=	$q_{x,\Theta}(i_{\text{out}}, l) \cdot G_{x,\Theta}(i_{\text{out}}, l)$
Final Output	:	$\hat{y}_{\Theta}(x)$	=	$\sum_{i_{\text{in}}} \Theta(i_{\text{in}}, i_{\text{out}}, d) \cdot z_{x,\Theta}(i_{\text{in}}, d-1)$

Table 2: Information flow in a FC-DNN with ReLU. Here, ‘ q ’s are pre-activation inputs, ‘ z ’s are output of the hidden layers, ‘ G ’s are the gating values. $l \in [d-1]$ is the index of the layer, i_{out} and i_{in} are indices of nodes in the current and previous layer respectively.

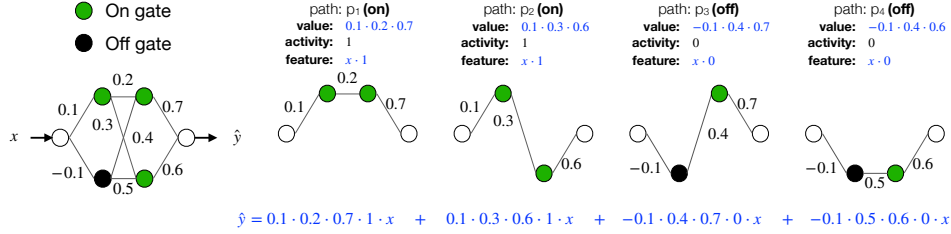


Figure 9: Illustration of dual view in a toy network with 2 layers, 2 gates per layer and 4 paths. Paths p_1 and p_2 are ‘on’ and paths p_3 and p_4 are ‘off’. The value, activity and feature of the individual paths are shown. \hat{y} is the summation of the individual path contributions.

- 504 (i) $A_{\Theta}(x, p) \triangleq \prod_{l=1}^{d-1} G_{x,\Theta}(\mathcal{I}_l(p), l)$ is the activity of a path.
- 505 (ii) $\phi_{\Theta}(x) \triangleq \left(x(\mathcal{I}_0^f(p)) A_{\Theta}(x, p), p \in [P^{fc}] \right) \in \mathbb{R}^{P^{fc}}$ is the neural path feature (NPF).
- 506 (iii) $v_{\Theta} \triangleq \left(\prod_{l=1}^d \Theta(\mathcal{I}_{l-1}(p), \mathcal{I}_l(p), l), p \in [P^{fc}] \right) \in \mathbb{R}^{P^{fc}}$ is the neural path value (NPV).

507 F Convolution With Global Average Pooling

508 In this section, we define NPFs and NPV in the presence of convolution with pooling. This requires
509 three key steps (i) treating pooling layers like gates/masks (see Definition F.2) (ii) bundling together
510 the paths that share the same path value (due to weight sharing in convolutions, see Definition F.3),
511 and (iii) re-defining the NPF and NPV for bundles (see Definition F.4). Weight sharing due to
512 convolutions and pooling makes the NPK rotationally invariant Lemma F.1. We begin by describing
513 the architecture.

514 **Architecture:** We consider (for sake of brevity) a 1-dimensional² convolutional neural network with
515 circular convolutions, with d_{cv} convolutional layers ($l = 1, \dots, d_{\text{cv}}$), followed by a *global-average-*
516 *pooling* layer ($l = d_{\text{cv}} + 1$) and d_{fc} ($l = d_{\text{cv}} + 2, \dots, d_{\text{cv}} + d_{\text{fc}} + 1$) fully connected layers. The
517 convolutional window size is $w_{\text{cv}} < d_{\text{in}}$, the number of filters per convolutional layer as well as the
518 width of the FC is w .

519 **Indexing:** Here $i_{\text{in}}/i_{\text{out}}$ are the indices (taking values in $[w]$) of the input/output filters. i_{cv} denotes
520 the indices of the convolutional window taking values in $[w_{\text{cv}}]$. i_{fout} denotes the indices (taking values
521 in $[d_{\text{in}}]$, the dimension of input features) of individual nodes in a given output filter. The weights
522 of layers $l \in [d_{\text{cv}}]$ are denoted by $\Theta(i_{\text{cv}}, i_{\text{in}}, i_{\text{out}}, l)$ and for layers $l \in [d_{\text{fc}}] + d_{\text{cv}}$ are denoted by
523 $\Theta(i_{\text{in}}, i_{\text{out}}, l)$. The pre-activations, gating and hidden unit outputs are denoted by $q_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l)$,
524 $G_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l)$, and $z_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l)$ for layers $l = 1, \dots, d_{\text{cv}}$.

525 **Definition F.1** (Circular Convolution). For $x \in \mathbb{R}^{d_{\text{in}}}$, $i \in [d_{\text{in}}]$ and $r \in \{0, \dots, d_{\text{in}} - 1\}$, define :

- 526 (i) $i \oplus r = i + r$, for $i + r \leq d_{\text{in}}$ and $i \oplus r = i + r - d_{\text{in}}$, for $i + r > d_{\text{in}}$.
- 527 (ii) $\text{rot}(x, r)(i) = x(i \oplus r)$, $i \in [d_{\text{in}}]$.
- 528 (iii) $q_{x,\Theta}(i_{\text{fout}}, i_{\text{out}}, l) = \sum_{i_{\text{cv}}, i_{\text{in}}} \Theta(i_{\text{cv}}, i_{\text{in}}, i_{\text{out}}, l) \cdot z_{x,\Theta}(i_{\text{fout}} \oplus (i_{\text{cv}} - 1), i_{\text{in}}, l-1)$.

²The results follow in a direct manner to any form of circular convolutions.

529 **Definition F.2** (Pooling). Let $G_{x,\Theta}^{pool}(i_{fout}, i_{out}, d_{cv} + 1)$ denote the pooling mask, then we have
530
$$z_{x,\Theta}(i_{out}, d_{cv} + 1) = \sum_{i_{fout}} z_{x,\Theta}(i_{fout}, i_{out}, d_{cv}) \cdot G_{x,\Theta}^{pool}(i_{fout}, i_{out}, d_{cv} + 1),$$

531 where in the case of global-average-pooling $G_{x,\Theta}^{pool}(i_{fout}, i_{out}, d_{cv} + 1) = \frac{1}{d_{in}}, \forall i_{out} \in [w], i_{fout} \in [d_{in}]$.

Input Layer	:	$z_{x,\Theta}(\cdot, 1, 0)$	=	x
-------------	---	-----------------------------	---	-----

Convolutional Layers, $l \in [d_{cv}]$				
Pre-Activation	:	$q_{x,\Theta}(i_{fout}, i_{out}, l)$	=	$\sum_{i_{cv}, i_{in}} \Theta(i_{cv}, i_{in}, i_{out}, l) \cdot z_{x,\Theta}(i_{fout} \oplus (i_{cv} - 1), i_{in}, l - 1)$
Gating Values	:	$G_{x,\Theta}(i_{fout}, i_{out}, l)$	=	$\mathbf{1}_{\{q_{x,\Theta}(i_{fout}, i_{out}, l) > 0\}}$
Hidden Unit Output	:	$z_{x,\Theta}(i_{fout}, i_{out}, l)$	=	$q_{x,\Theta}(i_{fout}, i_{out}, l) \cdot G_{x,\Theta}(i_{fout}, i_{out}, l)$

GAP Layer, $l = d_{cv} + 1$				
Hidden Unit Output	:	$z_{x,\Theta}(i_{out}, d_{cv} + 1)$	=	$\sum_{i_{fout}} z_{x,\Theta}(i_{fout}, i_{out}, d_{cv}) \cdot G_{x,\Theta}^{pool}(i_{fout}, i_{out}, d_{cv} + 1)$

Fully Connected Layers, $l \in [d_{fc}] + (d_{cv} + 1)$				
Pre-Activation	:	$q_{x,\Theta}(i_{out}, l)$	=	$\sum_{i_{in}} \Theta(i_{in}, i_{out}, l) \cdot z_{x,\Theta}(i_{in}, l - 1)$
Gating Values	:	$G_{x,\Theta}(i_{out}, l)$	=	$\mathbf{1}_{\{q_{x,\Theta}(i_{out}, l) > 0\}}$
Hidden Unit Output	:	$z_{x,\Theta}(i_{out}, l)$	=	$q_{x,\Theta}(i_{out}, l) \cdot G_{x,\Theta}(i_{out}, l)$
Final Output	:	$\hat{y}_{\Theta}(x)$	=	$\sum_{i_{in}} \Theta(i_{in}, i_{out}, d) \cdot z_{x,\Theta}(i_{in}, d - 1)$

Table 3: Shows the information flow in the convolutional architecture described at the beginning of Appendix F.

532 F.1 Neural Path Features, Neural Path Value

533 **Proposition F.1.** The total number of paths in a CNN is given by $P^{cnn} = d_{in}(w_{cv}w)^{d_{cv}}w^{(d_{fc}-1)}$.

534 **Notation**[Index Maps] The ranges of index maps $\mathcal{I}_l^f, \mathcal{I}_l^{cv}, \mathcal{I}_l$ are $[d_{in}]$, $[w_{cv}]$ and $[w]$ respectively.

535 **Definition F.3** (Bundle Paths of Sharing Weights). Let $\hat{P}^{cnn} = \frac{P^{cnn}}{d_{in}}$, and $\{B_1, \dots, B_{\hat{P}^{cnn}}\}$ be a
536 collection of sets such that $\forall i, j \in [\hat{P}^{cnn}], i \neq j$ we have $B_i \cap B_j = \emptyset$ and $\cup_{i=1}^{\hat{P}^{cnn}} B_i = [P^{cnn}]$. Further,
537 if paths $p, p' \in B_i$, then $\mathcal{I}_l^{cv}(p) = \mathcal{I}_l^{cv}(p'), \forall l = 1, \dots, d_{cv}$ and $\mathcal{I}_l(p) = \mathcal{I}_l(p'), \forall l = 0, \dots, d_{cv}$.

538 **Proposition F.2.** There are exactly d_{in} paths in a bundle.

539 **Definition F.4.** Let $x \in \mathbb{R}^{d_{in}}$ be the input to the CNN. For this input,

$$\begin{aligned}
A_{\Theta}(x, p) &\triangleq \left(\prod_{l=1}^{d_{cv}+1} G_{x,\Theta}(\mathcal{I}_l^f(p), \mathcal{I}_l(p), l) \right) \cdot \left(\prod_{l=d_{cv}+2}^{d_{cv}+d_{fc}+1} G_{x,\Theta}(\mathcal{I}_l(p), l) \right) \\
\phi_{x,\Theta}(\hat{p}) &\triangleq \sum_{\hat{p} \in B_{\hat{p}}} x(\mathcal{I}_0^f(p)) A_{\Theta}(x, p) \\
v_{\Theta}(B_{\hat{p}}) &\triangleq \left(\prod_{l=1}^{d_{cv}} \Theta(\mathcal{I}_l^{cv}(p), \mathcal{I}_{l-1}(p), \mathcal{I}_l(p), l) \right) \cdot \left(\prod_{l=d_{cv}+2}^{d_{cv}+d_{fc}+1} \Theta(\mathcal{I}_{l-1}(p), \mathcal{I}_l(p), l) \right)
\end{aligned}$$

541	NPF	$\phi_{x,\Theta} \triangleq (\phi_{x,\Theta}(B_{\hat{p}}), \hat{p} \in [\hat{P}^{cnn}]) \in \mathbb{R}^{\hat{P}^{cnn}}$
	NPV	$v_{\Theta} \triangleq (v_{\Theta}(B_{\hat{p}}), \hat{p} \in [\hat{P}^{cnn}]) \in \mathbb{R}^{\hat{P}^{cnn}}$

542 **Assumption F.1.** $\Theta_0^v \stackrel{i.i.d}{\sim} \text{Bernoulli}(\frac{1}{2})$ over $\{-\sigma, +\sigma\}$ and statistically independent of Θ_0^f .

543 F.2 Rotational Invariant Kernel

Lemma F.1.

$$\begin{aligned}
NPK_{\Theta}^{CONV}(x, x') &= \sum_{r=0}^{d_{in}-1} \langle x, \text{rot}(x', r) \rangle_{\text{overlap}_{\Theta}(\cdot, x, \text{rot}(x', r))} \\
&= \sum_{r=0}^{d_{in}-1} \langle \text{rot}(x, r), x' \rangle_{\text{overlap}_{\Theta}(\cdot, \text{rot}(x, r), x')}
\end{aligned}$$

544 *Proof.* For the CNN architecture considered in this paper, each bundle has exactly d_{in} number of
545 paths, each one corresponding to a distinct input node. For a bundle $b_{\hat{p}}$, let $b_{\hat{p}}(i), i \in [d_{in}]$ denote the

546 path starting from input node i .

$$\begin{aligned}
& \sum_{\hat{p} \in [\hat{P}]} \left(\sum_{i, i' \in [d_{\text{in}}]} x(i) x'(i') A_{\Theta}(x, b_{\hat{p}}(i)) A_{\Theta}(x', b_{\hat{p}}(i')) \right) \\
&= \sum_{\hat{p} \in [\hat{P}]} \left(\sum_{i \in [d_{\text{in}}], i' = i \oplus r, r \in \{0, \dots, d_{\text{in}} - 1\}} x(i) x'(i \oplus r) A_{\Theta}(x, b_{\hat{p}}(i)) A_{\Theta}(x', b_{\hat{p}}(i \oplus r)) \right) \\
&= \sum_{\hat{p} \in [\hat{P}]} \left(\sum_{i \in [d_{\text{in}}], r \in \{0, \dots, d_{\text{in}} - 1\}} x(i) \text{rot}(x', r)(i) A_{\Theta}(x, b_{\hat{p}}(i)) A_{\Theta}(\text{rot}(x', r), b_{\hat{p}}(i)) \right) \\
&= \sum_{r=0}^{d_{\text{in}}-1} \left(\sum_{i \in [d_{\text{in}}]} x(i) \text{rot}(x', r)(i) \sum_{\hat{p} \in [\hat{P}]} A_{\Theta}(x, b_{\hat{p}}(i)) A_{\Theta}(\text{rot}(x', r), b_{\hat{p}}(i)) \right) \\
&= \sum_{r=0}^{d_{\text{in}}-1} \left(\sum_{i \in [d_{\text{in}}]} x(i) \text{rot}(x', r)(i) \text{overlap}_{\Theta}(i, x, \text{rot}(x', r)) \right) \\
&= \sum_{r=0}^{d_{\text{in}}-1} \langle x, \text{rot}(x', r) \rangle \text{overlap}_{\Theta}(\cdot, x, \text{rot}(x', r))
\end{aligned}$$

547

□

548 **Theorem F.1.** Let $\sigma_{cv} = \frac{c_{\text{scale}}}{\sqrt{w} w_{cv}}$ for the convolutional layers and $\sigma_{fc} = \frac{c_{\text{scale}}}{\sqrt{w}}$ for FC layers. Under
549 Assumption F.1, as $w \rightarrow \infty$, with $\beta_{cv} = \left(d_{cv} \sigma_{cv}^{2(d_{cv}-1)} \sigma_{fc}^{2d_{fc}} + d_{fc} \sigma_{cv}^{2d_{cv}} \sigma_{fc}^{2(d_{fc}-1)} \right)$ we have:

$$NTK_{\Theta_0^{DGN}}^{CONV} \rightarrow \frac{\beta_{cv}}{d_{in}^2} \cdot NPK_{\Theta_0'}^{CONV}$$

550 *Proof.* Follows from Theorem 5.1 in [1].

□

551 G Residual Networks with Skip connections

552 As a consequence of the skip connections, within the ResNet architecture there are 2^b sub-FC
553 networks (see ??). The total number of paths P^{res} in the ResNet is equal to the summation of the
554 paths in these 2^b sub-FC networks (see Proposition G.1). Now, The neural path features and the
555 neural path value are P^{res} dimensional quantities, obtained as the concatenation of the NPFs and
556 NPV of the 2^b sub-FC networks.

557 **Proposition G.1.** The total number of paths in the ResNet is $P^{\text{res}} = d_{in} \cdot \sum_{i=0}^b \binom{b}{i} w^{(i+2)d_{blk}-1}$.

558 **Lemma G.1** (Sum of Product Kernel). Let $NPK_{\Theta}^{\text{RES}}$ be the NPK of the ResNet, and $NPK_{\Theta}^{\mathcal{J}}$ be the
559 NPK of the sub-FCNs within the ResNet obtained by ignoring those skip connections in the set \mathcal{J} .
560 Then,

$$NPK_{\Theta}^{\text{RES}} = \sum_{\mathcal{J} \in 2^{[b]}} NPK_{\Theta}^{\mathcal{J}}$$

561 *Proof.* Proof is complete by noting that the NPF of the ResNet is a concatenation of the NPFs of the
562 2^b distinct sub-FC-DNNs within the ResNet architecture. □

563 **Theorem G.1.** Let $\sigma = \frac{c_{\text{scale}}}{\sqrt{w}}$. Under Assumption F.1, as $w \rightarrow \infty$, for $\beta_{\text{res}}^{\mathcal{J}} = (|\mathcal{J}| + 2) \cdot d_{blk} \cdot$
564 $\sigma^2 \left((|\mathcal{J}| + 2) d_{blk} - 1 \right)$,

$$NTK_{\Theta_0^{DGN}}^{\text{RES}} \rightarrow \sum_{\mathcal{J} \in 2^{[b]}} \beta_{\text{res}}^{\mathcal{J}} NPK_{\Theta_0'}^{\mathcal{J}}$$

565 *Proof.* Follows from Theorem 5.1 in [1].

□

566 **H Numerical Experiments: Setup Details**

567 We now list the details related to the numerical experiments which have been left out in the main
568 body of the paper.

569 • **Computational Resource.** The numerical experiments were run in Nvidia-RTX 2080 TI GPUs and
570 Tesla V100 GPUs.

571 • All the models other than VGG and ResNet (and their variants) we used Adam [Kingma and Ba,
572 2014] with learning rate of 3×10^{-4} , and batch size of 32.

573 • All the VGG-16, Resnet-110 (and their DGN/DLGN) models we used *SGD* optimiser with
574 momentum 0.9 and the following learning rate schedule (as suggested in Mikler [2019]) : for
575 iterations $[0, 400)$ learning rate was 0.01, for iterations $[400, 32000)$ the learning rate was 0.1, for
576 iterations $[32000, 48000)$ the learning rate was 0.01, for iterations $[48000, 64000)$ the learning rate
577 was 0.001. The batch size was 128. The models were trained till 32 epochs.