3rd Symposium on Advances in Approximate Bayesian Inference, 2020 1–19

Correlated Weights in Infinite Limits of Deep Convolutional Neural Networks

Anonymous Authors
Anonymous Institution

Abstract

Infinite width limits of deep neural networks often have tractable forms. They have been used to analyse the behaviour of finite networks, as well as being useful methods in their own right. Currently used limits of deep convolutional networks lose correlating contributions from different spatial locations in the image, unlike their finite counterparts, even when those only use convolutions. We argue that this is undesirable, and remedy it by introducing spatial correlations in the prior over weights. This leads to correlated contributions being preserved in the wide limit. Varying the amount of correlation in the convolution weights allows interpolation between independent-weight limits and mean pooling (which is equivalent to complete correlation in the weights). Empirical evaluation of the infinitely wide network shows that optimal performance is achieved between the extremes, indicating the usefulness of considering correlations in the weights.

1. Introduction

Analysing infinitely wide limits of neural networks has long been used to provide insight into the properties of neural networks. Neal (1996) first noted such a relationship, through the correspondence between infinitely wide Bayesian neural networks and Gaussian processes (GPs). The success of the GPs raised the question of whether such a comparatively simple model could replace a complex neural network. MacKay (1998) noted that taking the infinite limit results in a fixed feature representation. Given that learnable features are a key desirable property of neural networks which is lost due to the infinite limit, MacKay inquired: "have we thrown the baby out with the bath water?"

In this work, we follow the recent interest in infinitely wide convolutional neural networks (Garriga-Alonso et al., 2019; Novak et al., 2019), to investigate another property that is lost when taking the infinite limit: correlation from patches in different parts of the image. Given that convolutions were developed to introduce these correlations and that they improve performance (Arora et al., 2019), it seems undesirable that they are lost when more filters are added. Currently, the only way of reintroducing these correlations is by changing the model architecture by introducing mean-pooling (Novak et al., 2019). This raises two questions:

1) Is the loss of patchwise correlations a necessary consequence of the infinite limit?

2) Is an architectural change the only way of reintroducing patchwise correlations?

We show that correlations between patches can also be maintained in the limit without pooling. Instead, we introduce correlations between the weights in the prior. The amount of correlation can be controlled, which allows us to interpolate between the existing approaches of full independence and mean-pooling. Our approach allows the discrete architectural choice of

© A. Authors.

mean-pooling to be replaced with a more flexible continuous amount of correlation. Empirical evaluation on CIFAR-10 shows that this additional flexibility improves performance.

Our work illustrates how choices that are made in the prior affect properties of the limit, and that good choices can improve performance. The success of this approach in the infinite limit also raises questions about whether correlated weights should be used in finite networks.

2. Spatial Correlations in Single-Layer Networks

To begin, we will analyse the infinite limit of a single hidden layer convolutional neural network (CNN). We extend Garriga-Alonso et al. (2019) and Novak et al. (2019) by considering weight priors with correlations. By adjusting the correlation we can interpolate between existing independent weight limits and mean-pooling, which previously had to be introduced as a discrete architectural choice. We also make connections to variants of convolutional GPs (van der Wilk et al., 2017; Dutordoir et al., 2020).

A single-layer Convolutional Neural Network (see fig. 1 for a graphical representation of the notation) takes in an image input $\mathbf{X} \in \mathbb{R}^{C^{(0)} \times P \cdot Q}$ with width P, height Q, and $C^{(0)}$ channels (e.g. one per colour). The image is divided up into patches $\mathbf{X}^{[p]} \in \mathbb{R}^{C^{(0)} \times p^{(1)} q^{(1)}}$, with p representing a location of one of the $P \cdot Q$ zero-padded patches. Weights are applied by taking an inner product with all patches, which we do $C^{(1)}$ times to give multiple channels in the next layer. By collecting all weights in the matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{C^{(1)} \times C^{(0)} \times p^{(1)} q^{(1)}}$ we can denote the computation of the pre- and post-nonlinearity activations as

$$Z^{(1)}_{cp}(\mathbf{X}) = \sum_{\gamma=1}^{C^{(0)}} \sum_{q=1}^{p^{(0)} q^{(0)}} X^{[p]}_{\gamma q} W^{(1)}_{c\gamma q}\,, \qquad A^{(1)}_{cp}(\mathbf{X}) = \phi\big(Z^{(1)}_{cp}(\mathbf{X})\big)\,. \tag{1}$$

In a single-layer CNN, these activations are followed by a fully-connected layer with weights $\mathbf{W}^{(2)} \in \mathbb{R}^{C^{(1)} \times PQ}$. Our final output is again given by a summation over the activations

$$f(\mathbf{X}) = \sum_{c=1}^{C^{(1)}} \sum_{p=1}^{P \cdot Q} \phi\big(Z^{(1)}_{cp}(\mathbf{X})\big) W^{(2)}_{cp} = \sum_{cp} A^{(1)}_{cp}(\mathbf{X}) W^{(2)}_{cp} = \sum_{p} A^{(f)}_{p}(\mathbf{X})\,, \tag{2}$$

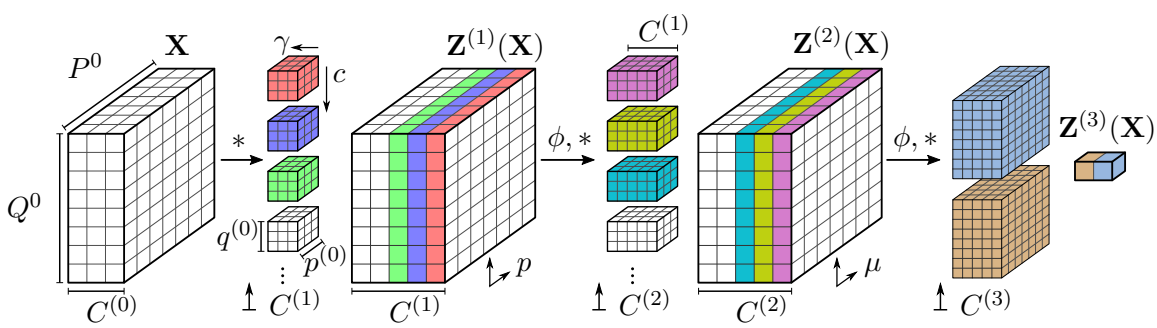

Figure 1: A deep convolutional neural network following our notation. Infinite limits are taken over the number of convolutional filters $C^{(\ell)}$. As the number of filters grow (vertical), the number of channels in the following layer (horizontal) grow as well.

where $A_p^{(f)}(\mathbf{X})$ denotes the result before the summation over the patch locations p.

We analyse the distribution on function outputs $f(\mathbf{X})$ for some Gaussian prior on the weights $p(\mathbf{W})$, where $\mathbf{W}$ is the collection of the weights at all layers. In all the cases we consider, we take the prior to be independent over layers and channels. Here we extend on earlier work by allowing spatial correlation in the final layer's weights (we will consider all layers later) through the covariance matrix $\mathbf{\Sigma}^{(1)} \in \mathbb{R}^{P \cdot Q \times P \cdot Q}$. This gives the prior

$$p(\mathbf{W}^{(1)}) = \prod_{c'=0}^{C^{(0)}} \prod_{c=1}^{C^{(1)}} \mathcal{N}\left(\mathbf{W}^{(1)}_{[cc':]}; 0, \mathbf{I}\right), \qquad p(\mathbf{W}^{(2)}) = \prod_{c=1}^{C^{(1)}} \mathcal{N}\left(\mathbf{W}^{(2)}_{[c:]}; 0, \frac{1}{C^{(1)}} \mathbf{\Sigma}^{(2)}\right), \tag{3}$$

where we use square brackets to index into a matrix or tensor, using the Numpy colon notation for the collection of all variables along an axis.

Independence between channels makes the collection of all activations[1] $\mathbf{A}^{(f)}(\mathbf{X})$ a sum of i.i.d. multivariate random variables, which allows us to apply the central limit theorem (CLT) as $C^{(1)} \to \infty$ (Neal, 1996). The covariance between the final-layer activations for two inputs $\mathbf{X}, \mathbf{X}'$ becomes

$$\begin{aligned}
\mathbb{C}_{\mathbf{W}}\left[A_p^{(f)}(\mathbf{X}), A_{p'}^{(f)}(\mathbf{X}')\right] &= \mathbb{E}_{\mathbf{W}}\left[\sum_{c=1}^{C^{(1)}} \sum_{c'=1}^{C^{(1)}} A_{cp}^{(1)}(\mathbf{X}) W_{cp}^{(2)} A_{c'p'}^{(1)}(\mathbf{X}') W_{c'p'}^{(2)}\right] \\
&= \sum_{c=1}^{C^{(1)}} \sum_{c'=1}^{C^{(1)}} \mathbb{E}_{\mathbf{W}^{(1)}}\left[A_{cp}^{(1)}(\mathbf{X}) A_{c'p'}^{(1)}(\mathbf{X}')\right] \mathbb{E}_{\mathbf{W}^{(2)}}\left[W_{cp}^{(2)} W_{c'p'}^{(2)}\right] \\
&= \sum_{c=1}^{C^{(1)}} \sum_{c'=1}^{C^{(1)}} \delta_{cc'} \mathbb{E}_{\mathbf{W}^{(1)}}\left[A_{cp}^{(1)}(\mathbf{X}) A_{c'p'}^{(1)}(\mathbf{X}')\right] \frac{\Sigma_{pp'}^{(2)}}{C^{(\ell)}} \\
&= \sum_{c=1}^{C^{(1)}} \mathbb{E}_{\mathbf{W}^{(1)}}\left[A_{cp}^{(1)}(\mathbf{X}) A_{cp'}^{(1)}(\mathbf{X}')\right] \frac{\Sigma_{pp'}^{(2)}}{C^{(\ell)}} = k^{(1)}(\mathbf{X}^{[p]}, \mathbf{X}'^{[p']}) \Sigma_{pp'}^{(2)} .
\end{aligned} \tag{4}$$

The limit of the sum of the final expectation over $\mathbf{W}^{(1)}$ can be found (see appendix D for details) in closed form and is denoted as $k^{(1)}(\mathbf{X}^{[p]}, \mathbf{X}'^{[p']})$. We find the final kernel for the GP by taking the covariance between function values $f(\mathbf{X})$ and $f(\mathbf{X}')$ and performing the final sum in eq. 2:

$$k(\mathbf{X}, \mathbf{X}') = \mathbb{C}\left[f(\mathbf{X}), f(\mathbf{X}')\right] = \sum_{pp'} k^{(1)}(\mathbf{X}^{[p]}, \mathbf{X}'^{[p']}) \Sigma_{pp'}^{(2)} . \tag{5}$$

We can now see how different choices for $\mathbf{\Sigma}^{(2)}$ give different forms of spatial correlation.

Independence Garriga-Alonso et al. (2019) and Novak et al. (2019) consider $\Sigma_{pp'}^{(2)} = \delta_{pp'}$, i.e. the case where all weights are independent. The resulting kernel simply sums components over patches, which implies an *additive model* (Stone, 1985), where a *different* function is applied to each patch, after which they are all summed together: $f(\mathbf{X}) = \sum_p f_p(\mathbf{X}^{[p]})$.

1. We use boldface variables to collect all subscripted tensors into a single matrix or tensor. This can then be indexed using square brackets, i.e. $\mathbf{A}_{[p]}^{(f)}(\mathbf{X}) = A_p^{(f)}(\mathbf{X})$.

This structure has commonly been applied to improve GP performance in high-dimensional settings (e.g. Duvenaud et al., 2011; Durrande et al., 2012). Novak et al. (2019) point out that the same kernel can be obtained by taking an infinite limit of a *locally connected network* (LCN) (LeCun, 1989) where connectivity is the same as in a CNN, but without weight sharing, indicating that a key desirable feature of CNNs is lost.

Mean-pooling By taking $\Sigma^{(2)}_{pp'} = 1$ we make the weights fully correlated over all locations, leading to identical weights for all p, i.e. $W^{(2)}_{cp} = W^{(2)}_{c}$. This is equivalent to taking the mean response over all spatial locations (see eq. 2), or mean-pooling. As Novak et al. (2019) discuss, this reintroduces the spatial correlation that is the intended result of weight sharing. This case is identical to the "translation invariant" construction in Van der Wilk et al. (2017). Their experiments show that mean-pooling is too restrictive in this single-layer case. To remedy this, they perform pooling with a learned weight α_p for each patch, which can be recovered by taking $\Sigma^{(2)}_{pp'} = \alpha_p \alpha_{p'}$. These weights need to be learned, which departs slightly from the strict Bayesian framework.

Spatially correlated weights In the pooling examples above, the spatial covariance of weights is taken to be a rank-1 matrix. We can add more flexibility to the model by varying the strength of correlation between weights based on their distance in the image. We consider an exponential decay depending on the distance between two patches: $\Sigma^{(2)}_{pp'} = \exp(-d(p, p')/l)$. We recover full independence by taking $l \to 0$, and mean-pooling with $l \to \infty$. Intermediate values of l allow the rigid assumption of complete weight sharing to be relaxed, while still retaining spatial correlations between similar patches. This construction gives exactly the same kernel as investigated by Dutordoir et al. (2020), who named this property "translation insensitivity", as opposed to the stricter invariance that mean-pooling gives. The additional flexibility improved performance without needing to add many parameters that are learned in an non-Bayesian fashion.

Our construction shows that spatial correlation can be retained in infinite limits without needing to resort to architectural changes. A simple change to the prior on the weights is all that is needed. This property is retained in wide limits of deep networks in a similar way, which we investigate next.

3. Spatial Correlations in Deep Networks

In appendix B, we provide a detailed but informal extension of the previous section's results to deep networks. We also formulate the correlated weights prior in the framework provided by by Yang (2019), which provides a formal justification for our results.

The procedure for computing the kernel has a recursive form similar to existing analyses (Garriga-Alonso et al., 2019; Novak et al., 2019). Negligable additional computational is needed to consider arbitrary correlations, compared to only considering mean-pooling (Novak et al., 2019; Arora et al., 2019). The main bottleneck is the need for computing covariances for all pairs of patches in the image, as in eq. 5.

4. Experiments

We seek to test two hypotheses. 1) Can we eliminate architectural choices, and recover their effect using continuous hyperparameters instead? 2) In the additional search space we have uncovered, can we find a kernel that performs better than the existing ones?

We evaluate various models on class-balanced subsets of CIFAR-10 of size $2^i \cdot 10$, following Arora et al. (2020). As is standard practice in the wide network literature, we reframe classification as regression to one-hot targets. We set $C = 0.1$ to make the mean over the data set zero, but we observed that this affects the results very little. The test predictions are the argmax over k of the posterior Gaussian process mean

$$f(x_*) = \mathbf{K}_{x_*\mathbf{X}}\left(\sigma^2\mathbf{I} + \mathbf{K}_{\mathbf{XX}}\right)^{-1}\mathbf{Y}_{[:k]}\,, \tag{6}$$

where σ^2 is a hyperparameter, the variance of the observation noise of the GP regression. We perform cross-validation to find a setting for σ^2. We use an eigenvalue decomposition on $\mathbf{K}_{xx}$ to avoid the need to recompute the decomposition for each value of σ^2.

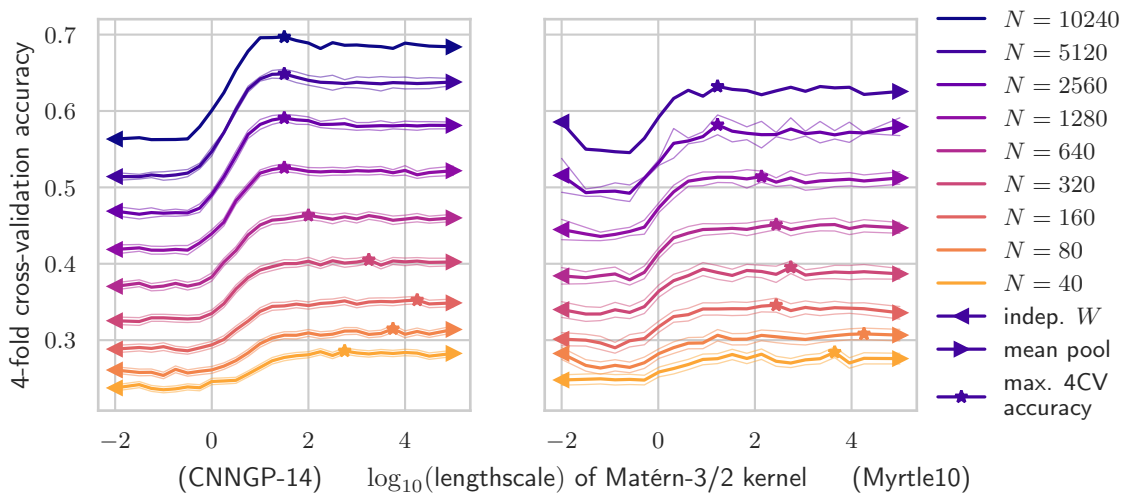

Figure 2: Cross-validation accuracy of the CNNGP-14 and Myrtle10 networks on subsets of CIFAR10, with varying lengthscale of the Matérn-3/2 kernel that determines the weight correlation in the last layer. With larger data set sizes N, the improvement is larger, and the optimal lengthscale converges to a similar value (≈ 4). For all data sets except the largest, the values are averaged over several runs, and the thin lines represent the $\pm 2\sigma_n$, the estimated standard deviation of the mean.

4.1. Correlated weights in the last layer

We start with two architectures used in the neural network kernel literature, the CNN-GP (Novak et al., 2019; Arora et al., 2019) with 14 layers, and the Myrtle network (Shankar

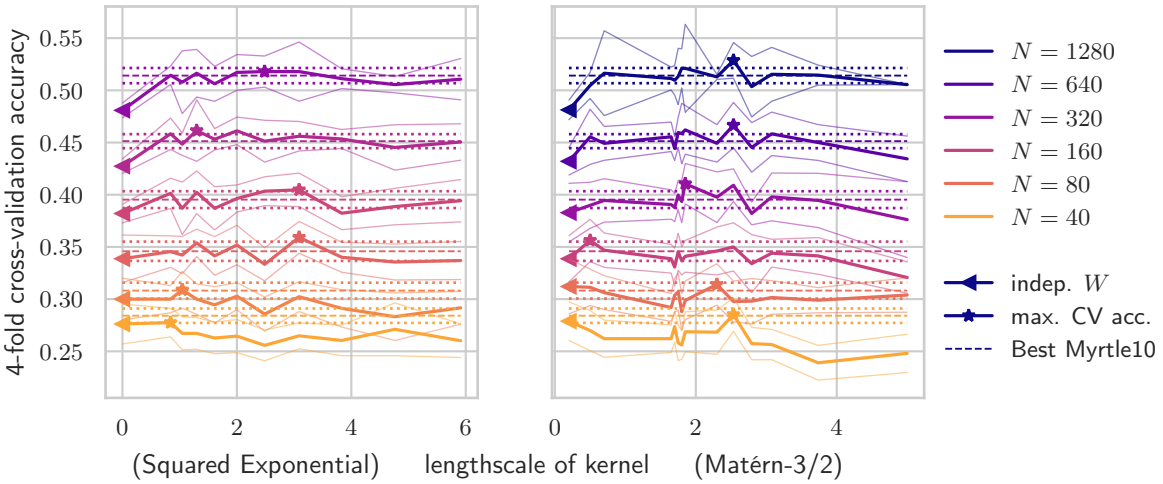

Figure 3: Correlated weights in intermediate layers. We replace pooling layers in the Myrtle10 architecture with larger convolutional filters with correlated weights. The amount of correlation is varied along the x-axis.

et al., 2020) with 10 layers. The CNNGP-14 architecture $((\texttt{conv}, \texttt{relu}) \times 14, \texttt{pool})$ has a 32×32-sized layer at the end, which is usually transformed into the 1×1 output using mean pooling. The Myrtle10 architecture $(((\texttt{conv}, \texttt{relu}) \times 2, \texttt{pool}_{2\times 2}) \times 3, \texttt{pool})$ has a 8×8 pooling layer at the end.

We replace the final pooling layers with a layer with correlated weights. Following (Dutordoir et al., 2020) covariance $\mathbf{\Sigma}_{pp'}$ of the weights is given by the Matérn-3/2 kernel (Rasmussen and Williams, 2006) with lengthscale λ:

$$\mathbf{\Sigma}_{pp'} = \left(1 + \frac{\sqrt{3}||p - p'||_2}{\lambda}\right) \exp\left(-\frac{\sqrt{3}||p - p'||_2}{\lambda}\right). \tag{7}$$

Note that p, p' are 2-d vectors representing patch locations. The "extremes" of independent weights and mean pooling are represented by setting $\mathbf{\Sigma}_{pp'} = \delta_{pp'}$ and $\mathbf{\Sigma}_{pp'} = \mathbf{1}\mathbf{1}^\intercal$, respectively.

In figure 2, we investigate how the 4-fold cross-validation accuracy on data sets of size $N = 2^i \cdot 10$ varies with the lengthscale λ of the Matérn-3/2 kernel, which controls the "amount" of spatial correlation in the weights of the last layer. For each data point in each line, we split the data set into 4 folds, and we calculate the test accuracy on 1 fold using the other 3 as training set, for each value of σ that we try. We take the maximum accuracy over σ.

We investigate how the effect above varies with data set size. As the data set grows larger, we observe that the advantage of having a structured covariance matrix in the output becomes more apparent. We can also see the optimal lengthscale λ converging to a similar value, of ≈ 4, which is evidence for the hypothesis holding with more data.

The largest data set size in each part of the plot was run only once because of computational constraints. We transform one data set of size N into two data sets of size $N/2$ by

taking block diagonals of the stored kernel matrix, so we have more runs for the smallest sizes. This is a valid Monte-Carlo estimate of the true accuracy under the data distribution, with less variance than independent data sets, because the data sets taken are anti-correlated, since they have no points in common. We have at least 1 other independent run for the second smallest data set size.

4.2. Correlated weights in intermediate layers

We take the same approach to the experiment in figure 3. This time, we replace the 2×2 intermediate mean-pooling layer, together with the next 3×3 convolution layer, in the Myrtle10 architecture with correlated weights. We change it to a 6×6 weight-correlated matrix. We observe that when the lengthscale is 0, the performance of the network is poor, suggesting that the mean-pooling layers in Myrtle10 are necessary. Additionally, we are able to recover the performance of the hand-tuned architecture by varying our lengthscale parameter.

5. Conclusion

The disappearance of spatial correlations in infinitely wide limits of deep convolutional neural networks could be seen as another example of how Gaussian processes lose favourable properties of neural networks. While other work sought to remedy this problem by making architectural changes, e.g. through mean-pooling, we showed that changing the weight prior could achieve the same effect. Our work has three main consequences. Firstly, weight correlation shows that locally connected models (without spatial correlation) and mean-pooling architectures (with spatial correlation) actually exist at ends of a spectrum. This unifies the two views in the neural network domain. We also unify to known convolutional architectures that were introduced from the Gaussian process community. Secondly, we show empirically that modest performance improvements can be gained by using weight correlations *between* the extremes of locally connected networks or mean-pooling. We also show that the performance of mean-pooling in intermediate layers can be matched by weight correlation. Finally, using weight correlation may provide advantages during hyperparameter tuning. Discrete architectural choices need to be searched through simple evaluation, while continuous parameters can use gradient-based optimisation. While we have not taken advantage of this in our current work, this may be a fruitful direction for future research.

References

Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 8141–8150. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9025-on-exact-computation-with-an-infinitely-wide-neural-net.pdf.

Sanjeev Arora, Simon S. Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In

8th International Conference on Learning Representations (ICLR 2020), 2020. URL https://openreview.net/forum?id=rkl8sJBYvH.

Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 342–350. Curran Associates, Inc., 2009. URL http://papers.nips.cc/paper/3628-kernel-methods-for-deep-learning.pdf.

Nicolas Durrande, David Ginsbourger, and Olivier Roustant. Additive covariance kernels for high-dimensional Gaussian process modeling. *Annales de la Faculté des sciences de Toulouse: Mathématiques*, Ser. 6, 21(3):481–499, 2012. URL http://www.numdam.org/item/AFST_2012_6_21_3_481_0.

Vincent Dutordoir, Mark van der Wilk, Artem Artemev, and James Hensman. Bayesian image classification with deep convolutional Gaussian processes. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS 2020)*. 2020.

David K. Duvenaud, Hannes Nickisch, and Carl Edward Rasmussen. Additive Gaussian processes. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24 (NeurIPS 2011)*, pages 226–234. Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4221-additive-gaussian-processes.pdf.

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow Gaussian processes. In *7th International Conference on Learning Representations (ICLR 2019)*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bklfsi0cKm.

Tamir Hazan and Tommi Jaakkola. Steps toward deep kernel methods from infinite neural networks, 2015.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 8571–8580. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8076-neural-tangent-kernel-convergence-and-generalization-in-neural-networks.pdf.

Yann LeCun. Generalization and network design strategies. *Connectionism in perspective*, 19:143–155, 1989.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as Gaussian processes. In *6th International Conference on Learning Representation (ICLR 2018)*, 2018. URL https://openreview.net/forum?id=B1EA-M-0Z.

David J. C. MacKay. Introduction to Gaussian processes. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, NATO ASI Series, pages 133–166. Kluwer Academic Press, 1998.

Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27 (NeurIPS 2014)*, pages 2627–2635. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5348-convolutional-kernel-networks.pdf.

Alexander G. de G. Matthews, Jiri Hron, Mark Rowland, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. In *6th International Conference on Learning Representations (ICLR 2018)*, 2018. URL https://openreview.net/forum?id=H1-nGgWC-.

Radford M. Neal. *Bayesian learning for neural networks*, volume 118. Springer, 1996.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are Gaussian processes. In *7th International Conference on Learning Representations (ICLR 2019)*, 2019. URL https://openreview.net/forum?id=B1g30j0qF7.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in Python. In *9th International Conference on Learning Representations (ICLR 2020)*, 2020. URL https://github.com/google/neural-tangents.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning. 2006*. The MIT Press, Cambridge, MA, USA, 2006.

Vaishaal Shankar, Alex Fang, Wenshuo Guo, Sara Fridovich-Keil, Ludwig Schmidt, Jonathan Ragan-Kelley, and Benjamin Recht. Neural kernels without tangents, 2020.

Charles J. Stone. Additive regression and other nonparametric models. *Ann. Statist.*, 13(2): 689–705, 06 1985. doi: 10.1214/aos/1176349548. URL https://doi.org/10.1214/aos/1176349548.

Mark van der Wilk. *Sparse Gaussian process approximations and applications*. PhD thesis, University of Cambridge, 2019.

Mark van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional Gaussian processes. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pages 2849–2858. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6877-convolutional-gaussian-processes.pdf.

Christopher K. I. Williams. Computing with infinite networks. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*

(NIPS 1996), pages 295–301. MIT Press, 1997. URL http://papers.nips.cc/paper/1197-computing-with-infinite-networks.pdf.

Christopher K. I. Williams and Carl Edward Rasmussen. Gaussian processes for regression. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8 (NeurIPS 1995)*, pages 514–520. MIT Press, 1996. URL http://papers.nips.cc/paper/1048-gaussian-processes-for-regression.pdf.

Greg Yang. Wide feedforward or recurrent neural networks of any architecture are Gaussian processes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 9951–9960. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9186-wide-feedforward-or-recurrent-neural-networks-of-any-architecture-are-gaussian-proces pdf.

Appendix A. Related work

Infinitely wide limits of neural networks are currently an important tool for creating approximations and analyses. Here we provide a background on the different infinite limits that have been developed, together with a brief overview of where they have been applied.

Interest in infinite limits first started with research into properties of Bayesian priors on the weights of neural networks. Neal (1996) noted that prior function draws from a single hidden layer neural network with appropriate Gaussian priors on the weights tended to a Gaussian process as the width grew to infinity. The simplicity of performing Bayesian inference in Gaussian process models led to their widespread adoption soon after (Williams and Rasmussen, 1996; Rasmussen and Williams, 2006). Over the years, the wide limits of networks with different weight priors and activation functions have been analysed, leading to various *kernels* which specify the properties of the limiting Gaussian processes (Williams, 1997; Cho and Saul, 2009).

With the increasing prominence of deep learning, recursive kernels were introduced in an attempt to obtain similar properties. Cho and Saul (2009); Mairal et al. (2014) investigated such methods for fully-connected and convolutional architectures respectively. Despite similarities between recursive kernels and neural networks, the derivation did not provide clear relationships, or any equivalence in a limit. Hazan and Jaakkola (2015) took initial steps to showing the wide limit equivalence of a neural network beyond the single layer case. Recently, Matthews et al. (2018); Lee et al. (2018) simultaneously provided general results for the convergence of the prior of deep fully-connected networks to a GP.[2] A different class of limiting kernels, the Neural Tangent Kernel (NTK), originated from analysis of the function implied by a neural network during optimisation (Jacot et al., 2018), rather than the prior implied by the weight initialisation. Just like the Bayesian prior limit, this kernel sheds light on certain properties of neural networks, as well as providing a method with predictive

2. The derivation of the limiting kernel differs between the two papers, with the results being consistent. Matthews et al. (2018) carefully take limits of realisable networks, while Lee et al. (2018) take the infinite limit of each layer sequentially.

capabilities of its own. The two approaches end up with subtly different kernels, which both can be computed as a recursive kernel.

With the general tools in place, Garriga-Alonso et al. (2019); Novak et al. (2019) derived limits of the prior of convolutional neural networks with infinite filters. These two papers directly motivated this work by noting that spatial correlations disappeared in the infinite limit. Spatial mean pooling at the last layer was suggested as one way to recover correlations, with Novak et al. (2019) providing initial evidence of its importance. Due to computational constraints, they were limited to using a Monte Carlo approximation to the limiting kernel, while Arora et al. (2019) performed the computation with the exact NTK. Very recent preprints provide follow-on work that pushes the performance of limit kernels (Shankar et al., 2020) and demonstrated the utility of limit kernels for small data tasks (Arora et al., 2020). Extending on the results for convolutional architectures, Yang (2019) showed how infinite limits could be derived for a much wider range of network architectures.

In the kernel and Gaussian process community, kernels with convolutional structure have also been proposed. Notably, these retained spatial correlation in either a fixed (van der Wilk et al., 2017) or adjustable (Mairal et al., 2014; Dutordoir et al., 2020) way. While these methods were not derived using an infinite limit, Van der Wilk (2019) provided an initial construction from an infinitely wide neural network limit. Inspired by these results, we propose limits of deep convolutional neural networks which retain spatial correlation in a similar way.

Appendix B. Spatial Correlations in Infinitely Wide Deep Convolutional Networks

The setup for the case of a deep neural network follows that of section 2, but with weights applied to each patch of the activations in the previous layer as

$$Z^{(\ell)}_{cp}(\mathbf{X}) = \sum_{\gamma=1}^{C^{(\ell)}} \sum_{q=1}^{p^{(\ell)}q^{(\ell)}} A^{(\ell-1)[p]}_{\gamma q}(\mathbf{X}) W^{(\ell)}_{c\gamma q}\,, \qquad A^{(\ell)}_{cp}(\mathbf{X}) = \phi\big(Z^{(\ell)}_{cp}(\mathbf{X})\big)\,. \tag{8}$$

We use two ways to index into activations. We either index into the pth location as $A^{(\ell)}_{cp}(\mathbf{X})$, or into the qth location in the pth patch as $A^{(\ell)[p]}_{cq}(\mathbf{X})$. For regular convolutions, the number of patches is equal to the number of spatial positions in the layer before due to zero-padding, regardless of filter size. The only operation that changes the spatial size of the activations is a strided convolution. The one exception is the final layer, where we reduce all activations with their own weight. To unify notation, we see this as just another convolutional layer, but with a patch size equal to the activation size, and without zero padding. The final layer can have multiple output channels to allow e.g. classification with multiple classes.

As pointed out by Matthews et al. (2018), a straightforward application of the central limit theorem is not strictly possible for deep networks. Fortunately, Yang (2019) developed a general framework for expressing neural network architectures and finding their corresponding Gaussian process infinite limits. The resulting kernel is given by the recursion that can be derived from a more informal argument which takes the infinite width limit in a sequential layer-by-layer fashion, as was used in Garriga-Alonso et al. (2019). We follow this informal

derivation, as this more naturally illustrates the procedure for computing the kernel. A formal justification can be found in appendix A.

B.1. Recursive Computation of the Kernel

To derive the limiting kernel for the output of the neural network, we will derive the distribution of the activations for each layer. In our weight prior, we correlate weights *within* a convolutional filter:

$$\mathcal{N}\left(\mathbf{W}^{(\ell)}_{[c\gamma:]}; 0, \frac{1}{C^{(\ell-1)}}\mathbf{\Sigma}^{(\ell)}\right). \tag{9}$$

Our derivation is general for any covariance matrix, so layers with correlated weights can be interspersed with the usual layers.

A Gaussian process is determined by the covariance of function values for pairs of inputs $\mathbf{X}, \mathbf{X}'$. Since the activations at the top layer are the function values, we will compute covariances between activations from the bottom of the network up. Starting from the recursion in eq. 8, we can find the covariance between any two pre-nonlinearity activations from a pair of inputs $\mathbf{X}, \mathbf{X}'$:

$$\begin{aligned}
\mathbb{C}_{\mathbf{W}}\left[Z^{(\ell)}_{cp}(\mathbf{X}), Z^{(\ell)}_{c'p'}(\mathbf{X}')\right] &= \sum_{\gamma=1}^{C^{(\ell-1)}}\sum_{\gamma'=1}^{C^{(\ell-1)}}\sum_{q=1}\sum_{q'=1}\mathbb{E}_{\mathbf{W}}\left[A^{(\ell-1)[p]}_{\gamma q}(\mathbf{X})W^{(\ell)}_{c\gamma q}A^{(\ell-1)[p']}_{\gamma q'}(\mathbf{X}')W^{(\ell)}_{c\gamma' q'}\right] \\
&= \delta_{cc'}\frac{1}{C^{(\ell-1)}}\sum_{\gamma q q'}\mathbb{E}_{\mathbf{W}}\left[A^{(\ell-1)[p]}_{\gamma q}(\mathbf{X})A^{(\ell-1)[p']}_{\gamma q'}(\mathbf{X}')\right]\Sigma^{(\ell)}_{qq'} \\
&= \delta_{cc'}K^{(\ell)}_{pp'}(\mathbf{X}, \mathbf{X}')\,.
\end{aligned} \tag{10}$$

For $\ell = 1$, the activations in the previous layer are the image inputs, i.e. $\mathbf{A}^{(0)}(\mathbf{X}) = \mathbf{X}$, making the expectation a simple product between image patches. The pre-nonlinearity activations are Gaussian, because of the linear relationship with the weights. This allows us to find the covariance of the post-nonlinearity activations. Since $\mathbf{Z}^{(\ell)}(\mathbf{X}), \mathbf{Z}^{(\ell)}(\mathbf{X}')$ are jointly Gaussian, the expectation will only depend on pairwise covariances. Here we represent this dependence through the function $F(\cdot,\cdot,\cdot)$ (see appendix B for details on the computation):

$$\begin{aligned}
\mathbb{E}_{\mathbf{W}}\left[A^{\ell}_{cp}(\mathbf{X})A^{\ell}_{cp'}(\mathbf{X}')\right] &= \mathbb{E}_{\mathbf{Z}^{(\ell)}(\mathbf{X}),\mathbf{Z}^{(\ell)}(\mathbf{X}')}\left[\phi\big(Z^{(\ell)}_{cp}(\mathbf{X})\big)\phi\big(Z^{(\ell)}_{cp}(\mathbf{X}')\big)\right] \\
&= F(K^{(\ell)}_{pp}(\mathbf{X},\mathbf{X}), K^{(\ell)}_{pp'}(\mathbf{X},\mathbf{X}'), K^{(\ell)}_{p'p'}(\mathbf{X}',\mathbf{X}')) \\
&= V^{(\ell)}_{pp'}(\mathbf{X},\mathbf{X}')\,.
\end{aligned} \tag{11}$$

The pre- and post-nonlinearity activations are independent between different channels, and identical over all channels, so we omit denoting the channel indices.

To compute the covariance of the pre-nonlinearity for $\ell \geq 2$, we can again apply eq. 10. Equation 11 shows that the post-nonlinearity covariances are constant over channels, so we

can simplify eq. 10 further:

$$K^{(\ell)}_{pp'}(\mathbf{X},\mathbf{X}') = \frac{1}{C^{(\ell-1)}} \sum_{\gamma qq'} \mathbb{E}_{\mathbf{W}}\left[A^{(\ell-1)[p]}_{\gamma q}(\mathbf{X}) A^{(\ell-1)[p']}_{\gamma q'}(\mathbf{X}')\right] \Sigma^{(\ell)}_{qq'}$$
$$= \sum_{q \in p\text{th patch}} \sum_{q' \in p'\text{th patch}} V^{(\ell-1)[p]}_{qq'}(\mathbf{X},\mathbf{X}') \Sigma^{(\ell)}_{qq'} \,. \tag{12}$$

We next want to compute the post-nonlinearity activations for layer ℓ. For finite $C^{(1)}$, $\mathbf{Z}^{(2)}(\mathbf{X})$ will not be Gaussian, which is required by eq. 11. However, if we take $C^{(\ell-1)} \to \infty$, $\mathbf{Z}^{(\ell)}(\mathbf{X})$ will converge to a Gaussian by the central limit theorem, all while keeping the covariance constant. After taking the limit, we can then apply eq. 11. This provides us with a recursive procedure to compute the covariances all the way up to the final layer, by sequentially taking limits of $C^{(\ell)} \to \infty$.

B.2. Computational Properties: convolutions double in dimensions

The core of the kernel computation for convolutional networks, whether or not they have spatial correlations, is the sum over pairs of elements of input patches qq', for each pair of output locations pp' in eq. 10. For a network that is built with convolutions of 2-dimensional inputs with 2-dimensional *weights*, the sum in 10 is exactly a 4-dimensional convolution of the full second moment of the input distribution (for inputs $\mathbf{X}$, the outer product), with the 4-dimensional *covariance tensor* of the weights. In general, a D-dimensional convolution in weight space corresponds to a $2D$-dimensional convolution in covariance space, with the same strides, dilation and padding.

With this framework, the expression for the covariance of the next layer when using independent weights becomes a 4-d convolution of the activations' second moment with a diagonal 4-d covariance tensor $\mathbf{\Sigma}^{(\ell)}$. This is conceptually simpler, but computationally more complex, than the convolution-like sums over diagonals pp' of Arora et al. (2019).

B.3. Implementation

We extend the `neural-tangents` (Novak et al., 2020) library with a convolution layer and a read-out layer, that admit a 4-dimensional covariance tensor for the weights. This allows interoperation with existing operators.

4d convolutions are uncommon in deep learning, so our implementation uses a sum over d 3-d convolutions. While this enables GPU acceleration, computing the kernel is a computationally costly operation. Reproducing our results takes around 10 days using an nVidia RTX 2070 GPU. Access to computational resources limited our experiments to subsets of data on CIFAR-10.

Appendix C. Proof that a CNN with correlations in the weights converges to a GP

In this section, we formally prove that a CNN with correlated weights converges to a Gaussian process in the limit of infinite width. Using the Netsor programming language due to Yang

(2019), most of the work in the proof is done by one step: describe a CNN with correlated weights in Netsor .

For the reader's convenience, we informally recall the Netsor programming language (Yang, 2019) and the key property of its programs (Corollary 6). The outline of our presentation here also closely follows Yang (2019). Readers familiar with Netsor should skip to appendix C.3, where we show the program that proves Theorem 9.

We write $[n]$ to mean the set $\{1, \ldots, n\}$.

C.1. The Netsor programming language

There are three types of variables: $\mathsf{G}(n)$-vars, $\mathsf{A}(n_1, n_2)$-vars, and $\mathsf{H}(n)$-vars. Each of these have one or two parameters, which are the widths we will take to infinity. For a given index in $[n]$ (or $[n_1], [n_2]$), each of these variables is a *scalar*. To represent vectors that do not grow to infinity, we need to use collections of variables.

G-vars (Gaussian-vars) are n-wise *approximately* i.i.d. and Gaussian. By "n-wise (approximately) independent" we mean that there can be correlations between G-vars, but only within a single index $i \in 1, \ldots, n$. G-vars will converge in distribution to an n-wise independent, identically distributed Gaussian in the limit of $n \to \infty$, if all widths are n.

A-vars represent matrices, like the weight matrices of a dense neural network. Their entries are always i.i.d. Gaussian with with zero mean, even for finite instantiations of the program (finite n). There are no correlations between different A-vars, or elements of the same A-var.

H-vars represent variables that become n-wise i.i.d. (not necessarily Gaussian) in the infinite limit. G is a subtype of H, so all G-vars are also H-vars.

We indicate the type of a variable, or each variable in a collection, using "var : Type".

Definition 1 (Netsor program) *A* Netsor *program consists of:*

Input: *A set of* G*-vars or* A*-vars.*

Body: *New variables can be defined using the following rules:*

- `MatMul`*:* $\mathsf{A}(n_1, n_2) \times \mathsf{H}(n_2) \to \mathsf{G}(n_1)$*. Multiply an an i.i.d. Gaussian matrix times an i.i.d. vector, which becomes a Gaussian vector in the limit* $n_2 \to \infty$*.*
- `LinComb`*: Given constants* $\alpha_1, \ldots, \alpha_K$*, and* G*-vars* $x_1, \ldots, x_K$ *of type* $\mathsf{G}(n_1)$*, their linear combination* $\sum_{k=1}^{K} \alpha_k x_k$ *is a* G*-var.*
- `Nonlin`*: applying an elementwise nonlinear function* $\phi : \mathbb{R}^k \to \mathbb{R}$*, we map several* G*-vars* $x_1, \ldots, x_K$ *to one* H*-var.*

Output: *A tuple of scalars* $(v_1^\intercal x_1/\sqrt{n_1}, \ \ldots, \ v_K^\intercal x_K/\sqrt{n_K})$*. The variables* $v_k : \mathsf{G}(n)$ *are input* G*-vars used only in the output tuple of the program. It may be the case that* $v_i = v_j$ *for different* i, j*. Each* $x_k : \mathsf{H}(n_k)$ *is an* H*-var.*

C.2. The output of a Netsor program converges to a Gaussian process

Definition 2 (Controlled function) *A function $\phi : \mathbb{R}^k \to \mathbb{R}$ is* controlled *if*

$$|\phi(\mathbf{x})| \leq C\, exp\Big(\|\mathbf{x}\|_2^{(2-\epsilon)}\Big) + c$$

for $C, c, \epsilon > 0$, where $\|\cdot\|_2$ is the L2 norm.

Intuitively, this means that the function ϕ grows more slowly than the rate at which the tail of a Gaussian decays. Recall that the tail of a mean zero, identity covariance Gaussian decays as $\mathcal{N}(\mathbf{x}\,|\,\mathbf{0}, \mathbf{I}) \propto \exp\Big(-\|\mathbf{x}\|_2^2\Big)$.

Assumption 3 *All nonlinear functions $\phi(\cdot)$ in the* Netsor *program are controlled.*

Assumption 4 (Distribution of A-var inputs) *Each element $W_{c,\gamma} \in A^i(n,n)$ in each input* A*-var is sampled from the zero-mean, i.i.d. Gaussian, $W_{c,\gamma} \sim \mathcal{N}\big(0, \sigma_w^2/n\big)$.*

Assumption 5 (Distribution of G-var inputs) *Consider the input vector of all* G*-vars for each channel $c \in [n]$, that is the vector $\mathbf{z}_c := [x_c : x$ is input* G*-var$]$. It is drawn from a Gaussian, $\mathbf{z}_c \sim \mathcal{N}\big(\boldsymbol{\mu}^{in}, \boldsymbol{\Sigma}^{in}\big)$. The covariance $\boldsymbol{\Sigma}^{in}$ may be singular. The* G*-vars v that correspond to the output are sampled independently from all other* G*-vars, so they are excluded from each $\mathbf{z}_c$*

Yang (2019) goes on to prove the Netsor master theorem, from which the corollary of interest follows.

Corollary 6 (Corollary 5.5, abridged, Yang (2019)) *Fix a* Netsor *program with controlled nonlinearities, and draw its inputs according to assumptions 4 and 5. For simplicity, fix the widths of all the variables to n. The program outputs are $(v_1^T x_1/\sqrt{n},\ \ldots,\ v_K^T x_K/\sqrt{n})$, where each x_k is an* H*-var, and each v_k is a* G*-var independent from all others with variance $\sigma_{v_k}^2$ (there can be some repeated indices, $v_i = v_j$). Then, as $n \to \infty$, the output tuple converges in distribution to a Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{K})$. The value of $\mathbf{K}$ is given by the recursive rules in equation 2 of Yang (2019).*

Informally, the rules consist of recursively calculating the covariances of the program's G-vars and output, assuming at every step that the G-vars are n-wise i.i.d. and Gaussian. This is the approach we employ in Section 4 of the main paper.

C.3. Description in Netsor of a CNN with correlations in the weights

The canonical way to represent convolutional filters in Netsor (Yang, 2019, Netsor program 4) is to use one A-var for every spatial location of the weights. That is, if our convolutional patches have size $p^{(\ell)} \times q^{(\ell)}$, we define the input $\mathbf{W}_q^{(\ell)} : \mathsf{A}(C^{(\ell+1)}, C^{(\ell)})$ for all $q \in [p^{(\ell)} q^{(\ell)}]$. But A-vars have to be independent of each other, so how can we add correlation in the weights? We apply the correlation separately, using a `LinComb` operation. For this, we will use the following well-known lemma, which is the $\mathbf{L}\epsilon + \boldsymbol{\mu}$ expression of a Gaussian random variable.

Lemma 7 *Let $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ be an arbitrary mean vector and covariance matrix, respectively. Let $u_1, \ldots, u_K \sim \mathcal{N}(0,1)$ be a collection of i.i.d. Gaussian random variables. Then, there exists a lower-triangular square matrix $\mathbf{L}$ such that $\mathbf{L}\mathbf{L}^\intercal = \boldsymbol{\Sigma}$. Furthermore, the random vector $\mathbf{w} \in \mathbb{R}^K$, $\mathbf{w} := \mathbf{L}\mathbf{u} + \boldsymbol{\mu}$ (equivalently, $w_k = \sum_{j=1}^k L_{kj} u_j + \mu_k$) has a Gaussian distribution, $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.*

Proof $\boldsymbol{\Sigma}$ is a covariance matrix so it is positive semi-definite, thus a lower-triangular square matrix $\mathbf{L}$ s.t. $\mathbf{L}\mathbf{L}^\intercal = \boldsymbol{\Sigma}$ always exists. (If $\boldsymbol{\Sigma}$ is singular, $\mathbf{L}$ might have zeros in the diagonal.) The vector $\mathbf{w}$ is Gaussian because it is a linear transformation of $\mathbf{u}$. Calculating its moments finishes the proof. ■

Thus, to express convolution in Netsor with correlated weights $\mathbf{w}$, we can use the following strategy. First, express several convolutions with uncorrelated weights $\mathbf{u}$. Then, combine the output of the convolutions using `LinComb` and coefficients of the matrix $\mathbf{L}$.

If the correlated weights have a non-zero mean, we can add an input G-var with mean μ and variance 0, and use it in the `LinComb` as well. Because we only use $\mu = 0$ in the main text, we omit this step here.

Lemma 8 (The convolution with L-trick is correct) *Consider the definitions in algorithm 1. Define the correlated convolution*

$$Y^{(2)}_{cp}(\mathbf{X}_m) := \sum_{\gamma=1}^{C^{(2)}} \sum_{q=1}^{p^{(2)}q^{(2)}} A^{(1)[p]}_{\gamma q}(\mathbf{X}_m) W^{(2)}_{c\gamma q}$$

where $\mathbf{W}^{(2)}_{c\gamma} \sim \mathcal{N}\Big(\mathbf{0}, \frac{1}{C^{(1)}}\boldsymbol{\Sigma}^{(2)}\Big)$, for $\gamma \in [C^{(1)}]$ and $c \in [C^{(2)}]$, mirroring eqs. (6) and (7) in the main text. Then, conditioning on the value of $A^{(1)}_{\gamma p}(\mathbf{X}_m)$ for all $\gamma \in C^{(1)}$, $p \in [P^{(1)}Q^{(1)}]$ and $m \in [M]$, and for any widths $C^{(1)}, C^{(2)}$, the random variables $Y^{(2)}_{cp}(\mathbf{X}_m)$ and $Z^{(2)}_{cp}(\mathbf{X}_m)$ have the same distribution for all $c \in [C^{(2)}]$, $p \in [P^{(2)}Q^{(2)}]$ and $m \in [M]$.

(Note, we abused notation and used c to index into the G-var $Z^{(2)}_p(\mathbf{X}_m)$, and γ for $A^{(2)}_q(\mathbf{X}_m)$.)
Proof Conditioned on $\mathbf{A}^{(1)}(\mathbf{X})$, both $\mathbf{Z}^{(2)}(\mathbf{X})$ and $\mathbf{Y}^{(2)}(\mathbf{X})$ are Gaussian, because they are linear combinations of Gaussians. Thus, we just have to show their first two moments are equal. First, the mean. $\mathbb{E}\Big[Z^{(2)}_{cp}(\mathbf{X}_m)\Big] = 0$ because each $\mathbb{E}\Big[\mathbf{U}^{(2)}_{c\gamma}\Big] = \mathbf{0}$, and $\mathbb{E}\Big[Y^{(2)}_{cp}(\mathbf{X}_m)\Big] = 0$ because $\mathbb{E}\Big[\mathbf{W}^{(1)}_{c\gamma}\Big] = \mathbf{0}$.

The covariance is more involved. First we rewrite $Z^{(2)}_{cp}(\mathbf{X}_m)$ as a function of $\mathbf{A}^{(1)}(\mathbf{X}_m)$, by substituting the definition of $B^{(2)[p]}_{qi}(\mathbf{X}_m)$ into it and making the indices of the `MatMul` explicit

$$Z^{(2)}_{cp}(\mathbf{X}_m) = \sum_{q=1}^{p^{(1)}q^{(1)}} \sum_{i=1}^{q} L^{(2)}_{qi} \sum_{\gamma=1}^{C^{(1)}} U^{(2)}_{c\gamma i} A^{(1)[p]}_{\gamma q}(\mathbf{X}_m) \tag{13}$$

Then we can write out the second moment:

$$\mathbb{E}\Big[Z^{(2)}_{cp}(\mathbf{X}_m) Z^{(2)}_{c'p'}(\mathbf{X}_{m'})\Big] = \sum_{q=1}^{p^{(1)}q^{(1)}} \sum_{q'=1}^{p^{(1)}q^{(1)}} \sum_{i=1}^{q} \sum_{i'=1}^{q'} \sum_{\gamma=1}^{C^{(1)}} \sum_{\gamma'=1}^{C^{(1)}} L^{(2)}_{qi} L^{(2)}_{q'i'} \mathbb{E}\Big[U^{(2)}_{c\gamma i} U^{(2)}_{c'\gamma' i'}\Big] A^{(1)[p]}_{\gamma q}(\mathbf{X}_m) A^{(1)[p']}_{\gamma' q'}(\mathbf{X}_{m'}) \tag{14}$$

Algorithm 1 Netsor description of the CNN in Figure 1, with correlated weights

// Program for M training + test points, $\mathbf{X}_1, \ldots, \mathbf{X}_M$. The activation nonlinearity is ϕ.

// G-vars for the 1st layer pre-activations, for all spatial locations p and input points $\mathbf{X}_m$.
Input $Z_p^{(1)}(\mathbf{X}_m) : \mathsf{G}(C^{(1)})$ for $p \in \left[P^{(1)}Q^{(1)}\right]$ and $m \in [M]$.

// A-vars for the independent convolutional patches $\mathbf{U}$, for every location i in a patch and layer ℓ.
Input $\mathbf{U}_i^{(\ell)} : \mathsf{A}\left(C^{(\ell)}, C^{(\ell-1)}\right)$ for $i \in [p^{(\ell)}q^{(\ell)}]$ and $\ell \in \{2\}$.

// G-vars for the output, for every location i
Input $\mathbf{U}_i^{(3)} : \mathsf{G}\left(C^{(2)}\right)$ for $i \in [P^{(2)}Q^{(2)}]$

// Construct the second layer's independent activations $B_q^{(2)[p]}$, for each spatial location p, location q in a patch and location i in a patch. **for** $m \in [M]$, $p \in \left[P^{(1)}Q^{(1)}\right]$, $q \in \left[p^{(1)}q^{(1)}\right]$, $i \in \left[p^{(1)}q^{(1)}\right]$ **do**

end

$A_p^{(1)}(\mathbf{X}_m) := \phi(Z_p^{(1)}(\mathbf{X}_m)) : \mathsf{H}(C^{(1)})$

`MatMul`: $B_{qi}^{(2)[p]}(\mathbf{X}_m) := \mathbf{U}_i^{(2)} A_q^{(1)[p]}(\mathbf{X}_m) : \mathsf{G}(C^{(2)})$

// Correlate the activations according to $\mathbf{\Sigma}^{(2)} = \mathbf{L}^{(2)}\left(\mathbf{L}^{(2)}\right)^{\intercal}$ **for** $m \in [M]$, $p \in \left[P^{(2)}Q^{(2)}\right]$ **do**

end

Convolution (sum in a patch, index q) with weights made dependent with index i (c.f. Lemma 7)

`LinComb`: $Z_p^{(2)}(\mathbf{X}_m) := \sum_{q=1}^{p^{(1)}q^{(1)}} \sum_{i=1}^{q} L_{qi}^{(2)} B_{qi}^{(2)[p]}(\mathbf{X}_m) : \mathsf{G}(C^{(2)})$

// Repeat the last two for-loops as needed to create more layers **for** $m \in [M]$, $p \in \left[P^{(2)}Q^{(2)}\right]$ **do**

end

$A_p^{(2)}(\mathbf{X}_m) := \phi(Z_p^{(2)}(\mathbf{X}_m)) : \mathsf{H}(C^{(2)})$

// One output for every spatial location p, spatial location i and data point m
Output $\left\{\left(U_i^{(3)}\right)^{\intercal} A_p^{(2)}(\mathbf{X}_m)/\sqrt{C^{(2)}} : \text{for } p \in \left[P^{(2)}Q^{(2)}\right], i \in \left[P^{(2)}Q^{(2)}\right] \text{ and } m \in [M]\right\}$
Output postprocessing: correlate the outputs (not strictly part of Netsor , c.f. Lemma 7)
$\left\{Z^{(3)}(\mathbf{X}_m) := \sum_{p=1}^{P^{(2)}Q^{(2)}} \sum_{i=1}^{p} \left(U_i^{(3)}\right)^{\intercal} A_p^{(2)}(\mathbf{X}_m) : \text{ for } m \in [M]\right\}$

Because the $\mathbf{U}_i^{(2)}$ are independent, that is $\mathbb{E}\left[U_{c\gamma i}^{(2)} U_{c'\gamma' i'}^{(2)}\right] = \delta_{cc'}\delta_{\gamma\gamma'}\delta_{ii'} 1/C^{(1)}$ (assumption 4), the covariance across output channels c, c' is zero if $c \neq c'$. Furthermore, we can reduce some double sums to single sums:

$$\mathbb{E}\left[Z_{cp}^{(2)}(\mathbf{X}_m) Z_{c'p'}^{(2)}(\mathbf{X}_{m'})\right] = \delta_{cc'} \sum_{q=1}^{p^{(1)}q^{(1)}} \sum_{q'=1}^{p^{(1)}q^{(1)}} \frac{1}{C^{(1)}} \sum_{\gamma=1}^{C^{(1)}} \sum_{i=1}^{\min(q,q')} L_{qi}^{(2)} L_{q'i'}^{(2)} A_{\gamma q}^{(1)[p]}(\mathbf{X}_m) A_{\gamma q}^{(1)[p']}(\mathbf{X}_{m'}) \tag{15}$$

$$= \delta_{cc'} \sum_{q=1}^{p^{(1)}q^{(1)}} \sum_{q'=1}^{p^{(1)}q^{(1)}} \frac{1}{C^{(1)}} \sum_{\gamma=1}^{C^{(1)}} \Sigma_{qq'}^{(2)} A_{\gamma q}^{(1)[p]}(\mathbf{X}_m) A_{\gamma q}^{(1)[p']}(\mathbf{X}_{m'}), \tag{16}$$

where we recognized $\sum_{i=1}^{\min(q,q')} L_{qi}^{(2)} L_{q'i'}^{(2)}$ as lower-triangular matrix multiplication, and recall that $\mathbf{\Sigma}^{(2)} = \mathbf{L}^{(2)}\left(\mathbf{L}^{(2)}\right)^{\mathsf{T}}$.

The covariance $\mathbb{E}\left[Y_{cp}^{(2)}(\mathbf{X}_m) Y_{c'p'}^{(2)}(\mathbf{X}_{m'})\right]$ (conditioned on $\mathbf{A}^{(1)}(\mathbf{X}_m)$) has exactly the same expression. This can be derived in the same way as equation (8) in the main text. ■

Theorem 9 (Correlated CNN converges in distribution to a GP, with covariance given in Sectio
Given a set of M input points $\mathbf{X}_1, \ldots, \mathbf{X}_M$, the postprocessed output of the Netsor *program in algorithm 1 correctly implements a convolutional neural network (CNN) with correlated weights and 3 layers, as described in equations (6) and (7) and figure 1 of the main text. Fix the widths of all channels $C^{(\ell)} = n$. Under assumptions (4,5,3), as $n \to \infty$, the output of the correlated CNN applied to the training set $\{\mathbf{X}_m\}_{m=1}^M$ converges in distribution to a Gaussian process with mean 0, and covariance $K^{(3)}(\mathbf{X}_m, \mathbf{X}_{m'})$ given by equation (10) in the main text.*

Proof We proceed in order of the claims.

- **The program in algorithm 1 is correct:** the novel part of this program, compared to the CNNs with mean pooling of Yang (2019, appendix B.2), is the application of Lemma 7 to correlate the convolution weights and the postprocessed output. Applying Lemma 8 to both, we can see that Algorithm 1 implements a 3-layer CNN with correlated weights.

- **The postprocessed output of the program converges to a GP** with covariance in equation (10) of the main text. Using Corollary 6, we show the output tuple of the Netsor program in algorithm 1 converges in distribution to a GP, with mean zero and a covariance that is independent across the index i of the output G-vars $U_i^{(3)}$. Using the same technique as Lemma 8, we can show that the covariance of the postprocessed output $Z^{(3)}(\mathbf{X}_m)$ is the correct one.

  Now we need only show that the postprocessed outputs $\left\{Z^{(3)}(\mathbf{X}_m)\right\}_{m=1}^M$ converge to a GP in distribution. Convergence in distribution is convergence of the expectation of all bounded functions. Since the covariance $\mathbf{\Sigma}^{(3)}$ of the last layer weights is fixed, the set of bounded functions of $\left\{Z^{(3)}(\mathbf{X}_m)\right\}_{m=1}^M$ is the same as the set of bounded functions of $\left\{\left(U_i^{(3)}\right)^{\mathsf{T}} A_p^{(2)}(\mathbf{X}_m)/\sqrt{C^{(2)}}\right\}_{i,m}$.

■

Appendix D. Details of the expectation of the nonlinearities.

For details on the computation of the expectation for the second moment of tanhs, see the appendix of (Lee et al., 2018).

For the balanced ReLU nonlinearity ($\phi(x) = \sqrt{2}\max(0, x)$), which we use in all the experiments in this paper, we can use the expression by Cho and Saul (2009):

$$V_{pp'}^{(\ell)}(\mathbf{X}, \mathbf{X}') = \frac{\sqrt{K_{pp}^{(\ell)}(\mathbf{X}, \mathbf{X}) K_{p'p'}^{(\ell)}(\mathbf{X}', \mathbf{X}')}}{\pi} \left(\sin \theta_{pp'}^{(\ell)} + (\pi - \theta_{pp'}^{(\ell)}) \cos \theta_{pp'}^{(\ell)} \right) \tag{17}$$

where $\theta_{pp'}^{(\ell)} = \cos^{-1}\left(K_{pp'}^{(\ell)}(\mathbf{X}, \mathbf{X}') / \sqrt{K_{pp}^{(\ell)}(\mathbf{X}, \mathbf{X}) K_{p'p'}^{(\ell)}(\mathbf{X}', \mathbf{X}')} \right)$.

It also turns out that $V_{pp}^{(\ell)}(\mathbf{X}, \mathbf{X}) = K_{pp}^{(\ell)}(\mathbf{X}, \mathbf{X})$ and $V_{p'p'}^{(\ell)}(\mathbf{X}', \mathbf{X}') = K_{p'p'}^{(\ell)}(\mathbf{X}', \mathbf{X}')$.

This was adopted from the start of the GP-NN literature by Lee et al. (2018); Matthews et al. (2018). The `neural-tangents` library (Novak et al., 2020) implements a numerically stable version of it.