A Related Work

The techniques in this paper were inspired by prior research in multiple areas, including neural architecture and activation function search, as well as research on the FIM.

Neural Architecture Search In neural architecture search [NAS; 12, 53, 56], the goal is to design a neural network architecture automatically. NAS approaches typically focus on optimizing the type and location of the layers and the connections between them, but often use standard activation functions like ReLU. This work in complimentary to NAS approaches, because it uses standard architectures but optimizes the design of the activation function.

Zero-cost NAS Proxies Recently, zero-cost NAS proxies have received increased attention [39, 46, 52]. These approaches aim to accelerate neural architecture search by using cheap surrogate calculations in place of expensive full training of architectures. This paper adopts a similar approach, using FIM eigenvalues and activation function outputs to predict which activation functions are likely to be most promising before dedicating resources to evaluating them.

Activation Function Search Methods for automatically discovering activation functions include reinforcement learning [45], evolutionary computation [3, 5, 6, 33], and gradient-based methods [1, 5, 18, 40, 50]. This paper builds upon existing work, focusing on efficient search and on understanding the properties that make activation functions effective.

Other Uses of the FIM This paper used FIM eigenvalues to predict the performance of different activation functions. The FIM is an important quantity in machine learning with several uses. One important example is optimal experiment design [13], where experiments are designed to be optimal with respect some criterion. The criteria vary, but are often functions of the eigenvalues of the FIM, such as the maximum or minimum eigenvalue, or the trace of the FIM (sum of the eigenvalues) or determinant of the FIM (product of the eigenvalues). Instead of choosing one optimality criterion and only considering one summary statistic, this paper keeps all of the eigenvalues of the FIM and learns an optimal distribution experimentally.

Past work has also used the eigenvalues of the FIM to determine suitable values of the batch size or learning rate for neural networks [14, 15, 20, 27, 32]. The FIM provides insights to the learning dynamics of SGD [25] and the dynamics of signal propagation at different layers in networks with and without batch normalization layers [23]. The FIM has also been used to develop second-order optimization algorithms for neural networks [19, 35, 36]. Applying it to activation function design is thus a compelling further opportunity.

B Activation Function Search Spaces

The activation functions in this paper were implemented as computation graphs from the PANGAEA search space [5]. The space includes unary and binary operators, in addition to existing activation functions [7, 11, 28, 41, 45]. This approach allows specifying families of functions in a compact manner. It is thus possible to focus the search on a space where good functions are likely to be located, and also to search it comprehensively.

Benchmark Datasets The benchmark datasets introduced in Section 2 contain every activation function of the three-node form `binary(unary(`x`),unary(`x`))` using the operators in Table 3. The result is 5,103 activation functions, of which 2,913 are unique. This space is visualized in Figure 4.

For `Act-Bench-CNN` and `Act-Bench-ResNet`, the accuracies are the median from three runs. For `Act-Bench-ViT`, the results are from single runs due to computational costs.

New Tasks The experiments in Section 6 utilized a larger search space. Specifically, it was based on the following four-node computation graphs: `binary(unary(unary(`x`)),unary(`x`))`, `binary(unary(`x`),unary(unary(`x`)))`, `n-ary(unary(`x`),unary(`x`),unary(`x`))`, `unary(binary(unary(`x`),unary(`x`)))`, and `unary(unary(unary(unary(`x`))))`. The unary and binary nodes used the operators in Table 3, and the n-ary node used the sum, product,

Table 3: Activation function search spaces were defined through computation graphs consisting of basic unary and binary operators as well as existing activation functions [5].

Unary			**Binary**
0	erf(x)	ReLU(x)	$x_1 + x_2$
1	erfc(x)	ELU(x)	$x_1 - x_2$
x	sinh(x)	SELU(x)	$x_1 \cdot x_2$
$-x$	cosh(x)	Swish(x)	x_1 / x_2
$\|x\|$	tanh(x)	Softplus(x)	$x_1^{x_2}$
x^{-1}	arcsinh(x)	Softsign(x)	$\max\{x_1, x_2\}$
x^2	arctan(x)	HardSigmoid(x)	$\min\{x_1, x_2\}$
e^x	$e^x - 1$	bessel_i0e(x)	
$\sigma(x)$	$\log(\sigma(x))$	bessel_i1e(x)	

maximum, and minimum operators. Together, these computation graphs create a search space with 1,023,516 functions, of which 425,896 are unique. This space is visualized in Figure 9.

C Fisher Information Matrix Details

In order to calculate the FIM, this paper uses the K-FAC approach [19, 35, 36]. This technique is summarized in this Appendix, with notation similar to that of Grosse and Martens [19].

Preliminaries A feedforward neural network maps an input $\mathbf{a}_0 = \mathbf{x}$ to an output $\mathbf{a}_L = f(\mathbf{x}; \boldsymbol{\theta})$ through a series of L layers. Each layer $l \in \{1, \ldots, L\}$ is comprised of a weight matrix $\mathbf{W}_l$, a bias vector $\mathbf{b}_l$, and an element-wise activation function ϕ_l. With $\bar{\mathbf{W}}_l = (\mathbf{b}_l \quad \mathbf{W}_l)$ and $\bar{\mathbf{a}}_l = \begin{pmatrix} 1 & \mathbf{a}_l^\top \end{pmatrix}^\top$, each layer implements the transformation

$$\mathbf{s}_l = \bar{\mathbf{W}}_l \bar{\mathbf{a}}_{l-1}, \tag{4}$$

$$\mathbf{a}_l = \phi_l(\mathbf{s}_l). \tag{5}$$

Let $\boldsymbol{\theta} = \begin{pmatrix} \mathrm{vec}(\bar{\mathbf{W}}_1)^\top & \cdots & \mathrm{vec}(\bar{\mathbf{W}}_L)^\top \end{pmatrix}^\top$ represent the vector of all network parameters. Parameterized by $\boldsymbol{\theta}$ and given inputs $\mathbf{x}$ drawn from a training distribution $Q_{\mathbf{x}}$, the neural network defines the conditional distribution $R_{\mathbf{y}|f(\mathbf{x};\boldsymbol{\theta})}$. The Fisher information matrix associated with this model is

$$\mathbf{F} = \mathop{\mathbb{E}}_{\substack{\mathbf{x} \sim Q_{\mathbf{x}} \\ \mathbf{y} \sim R_{\mathbf{y}|f(\mathbf{x};\boldsymbol{\theta})}}} \left[\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta})) \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))^\top \right]. \tag{6}$$

As usual in deep learning, the loss function $\mathcal{L}(\mathbf{y}, \mathbf{z})$ represents the negative log-likelihood associated with $R_{\mathbf{y}|f(\mathbf{x};\boldsymbol{\theta})}$ and quantifies the discrepancy between the model's prediction $\mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta})$ and the true label $\mathbf{y}$. The network is trained to minimize the loss by updating its parameters according to the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))$.

Approximations For ease of notation, write $\mathcal{D}\mathbf{v} = \nabla_{\mathbf{v}} \mathcal{L}(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))$. Recalling that $\boldsymbol{\theta} = \begin{pmatrix} \mathrm{vec}(\bar{\mathbf{W}}_1)^\top & \cdots & \mathrm{vec}(\bar{\mathbf{W}}_L)^\top \end{pmatrix}^\top$, the FIM can be expressed as an $L \times L$ block matrix:

$$\mathbf{F} = \begin{pmatrix} \mathbb{E}\left[\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_1)\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_1)^\top\right] & \cdots & \mathbb{E}\left[\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_1)\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_L)^\top\right] \\ \vdots & \ddots & \vdots \\ \mathbb{E}\left[\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_L)\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_1)^\top\right] & \cdots & \mathbb{E}\left[\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_L)\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_L)^\top\right] \end{pmatrix}. \tag{7}$$

Note that $\mathcal{D}\bar{\mathbf{W}}_l = \mathcal{D}\mathbf{s}_l \bar{\mathbf{a}}_{l-1}^\top$, and recall that $\mathrm{vec}(\mathbf{u}\mathbf{v}^\top) = \mathbf{v} \otimes \mathbf{u}$. Each block of the FIM can be written as

$$\begin{aligned} \mathbf{F}_{i,j} &= \mathbb{E}\left[\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_i)\mathrm{vec}(\mathcal{D}\bar{\mathbf{W}}_j)^\top\right] && (8) \\ &= \mathbb{E}\left[\mathrm{vec}(\mathcal{D}\mathbf{s}_i \bar{\mathbf{a}}_{i-1}^\top)\mathrm{vec}(\mathcal{D}\mathbf{s}_j \bar{\mathbf{a}}_{j-1}^\top)^\top\right] && (9) \\ &= \mathbb{E}\left[(\bar{\mathbf{a}}_{i-1} \otimes \mathcal{D}\mathbf{s}_i)(\bar{\mathbf{a}}_{j-1} \otimes \mathcal{D}\mathbf{s}_j)^\top\right] && (10) \\ &= \mathbb{E}\left[(\bar{\mathbf{a}}_{i-1} \otimes \mathcal{D}\mathbf{s}_i)(\bar{\mathbf{a}}_{j-1}^\top \otimes \mathcal{D}\mathbf{s}_j^\top)\right] && (11) \\ &= \mathbb{E}\left[\bar{\mathbf{a}}_{i-1}\bar{\mathbf{a}}_{j-1}^\top \otimes \mathcal{D}\mathbf{s}_i \mathcal{D}\mathbf{s}_j^\top\right]. && (12) \end{aligned}$$

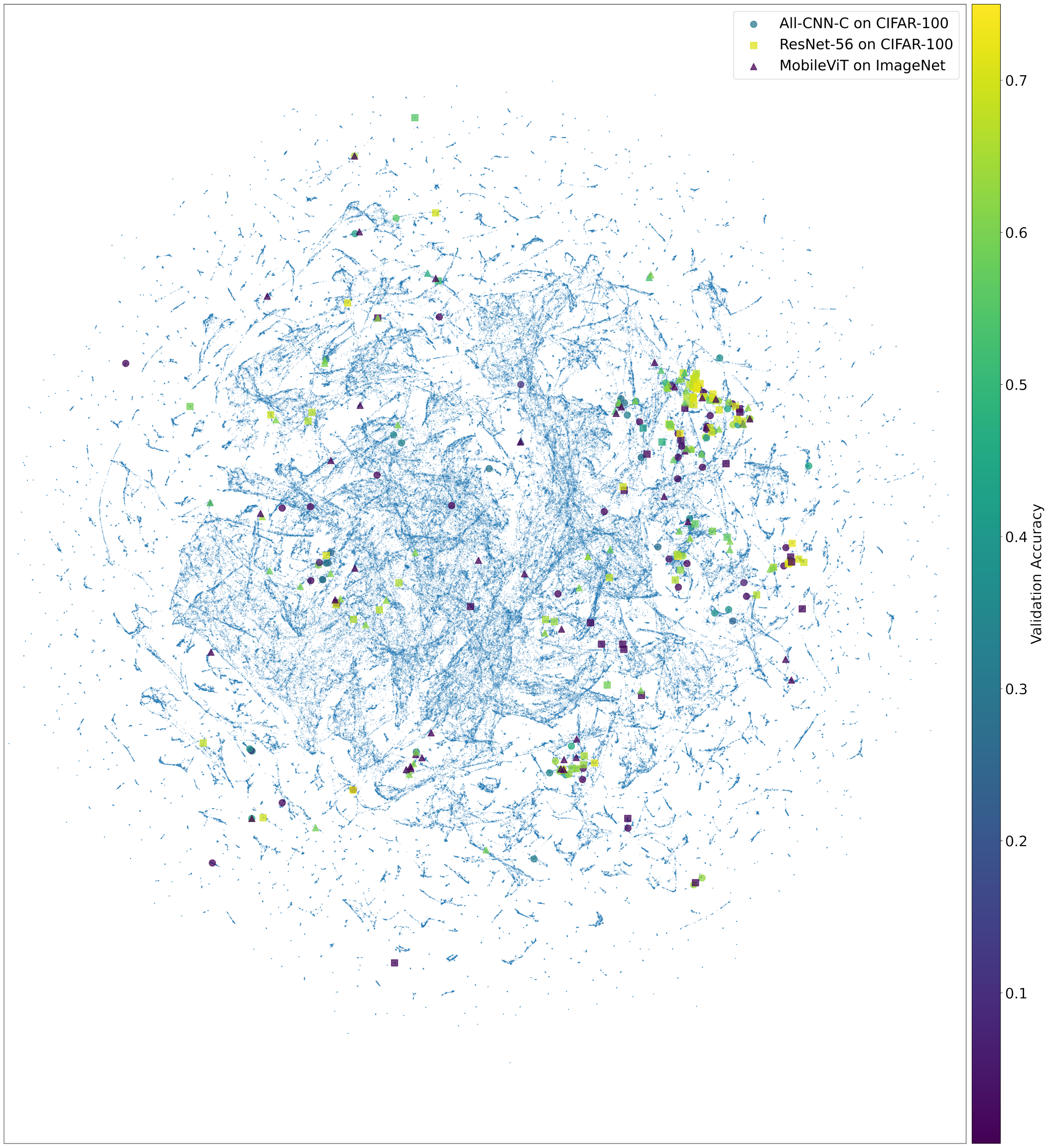

Figure 9: Low-dimensional UMAP representation of the 425,896 function search space. The activation functions are embedded according to their outputs; each point represents a unique function. The larger points represent activation functions that were evaluated during the searches; they are colored according to their validation accuracy. Although the space is vast, the searches require only tens of evaluations to discover good activation functions.

Two approximations are necessary in order to make representation of the FIM practical. First, assume that different layers have uncorrelated weight derivatives. The FIM can then be approximated as a block diagonal matrix, with $\mathbf{F}_{i,j} = \mathbf{0}$ if $i \neq j$. Second, if one approximates the pre-activation derivatives $\mathcal{D}\mathbf{s}_l$ and activations $\bar{\mathbf{a}}_{l-1}^\top$ as independent, then the diagonal blocks of the FIM can be further decomposed into the Kronecker product of two smaller matrices:

$$\mathbf{F}_{l,l} = \mathbb{E}\left[\bar{\mathbf{a}}_{l-1}\bar{\mathbf{a}}_{l-1}^\top \otimes \mathcal{D}\mathbf{s}_l\mathcal{D}\mathbf{s}_l^\top\right] \approx \mathbb{E}\left[\bar{\mathbf{a}}_{l-1}\bar{\mathbf{a}}_{l-1}^\top\right] \otimes \mathbb{E}\left[\mathcal{D}\mathbf{s}_l\mathcal{D}\mathbf{s}_l^\top\right]. \tag{13}$$

Let $\mathbf{\Omega}_l = \mathbb{E}\left[\bar{\mathbf{a}}_l\bar{\mathbf{a}}_l^\top\right]$ and $\mathbf{\Gamma}_l = \mathbb{E}\left[\mathcal{D}\mathbf{s}_l\mathcal{D}\mathbf{s}_l^\top\right]$. The approximate empirical FIM is then written as

$$\hat{\mathbf{F}} = \begin{pmatrix} \mathbf{\Omega}_0 \otimes \mathbf{\Gamma}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{\Omega}_{L-1} \otimes \mathbf{\Gamma}_L \end{pmatrix}. \tag{14}$$

Layer-Specific Implementation The above example illustrates FIM approximation for a simple feedforward network. However, most modern architectures contain several different kinds of layers. Some layers like pooling, normalization, or dropout layers do not have trainable weights, and therefore these layers are not included in the FIM [24, 48].

Each diagonal entry $\mathbf{\Omega}_{l-1} \otimes \mathbf{\Gamma}_l$ corresponds to one layer with weights. The calculation differs slightly depending on the layer type, but otherwise the example above can be straightforwardly extended to more complicated networks. Calculations for three common layer types are presented below.

Dense Layers For dense layers, the matrices $\mathbf{\Omega}_{l-1}$ and $\mathbf{\Gamma}_l$ can be readily computed with one forward and backward pass through the network using a mini-batch of data. The eigenvalues are then computed using standard techniques.

Convolutional Layers Convolutional layers require special consideration to calculate $\mathbf{\Omega}_{l-1}$ and $\mathbf{\Gamma}_l$. For a given layer, let M represent the batch size, $\mathcal{T}$ the set of spatial locations (typically a two-dimensional grid), Δ the set of spatial offsets from the center of the filter, and I and J the number of output and input maps, respectively. The activations are represented by the $M \times |\mathcal{T}| \times J$ array $\mathbf{A}_{l-1}$. The weights are represented by the $I \times |\Delta| \times J$ array $\mathbf{W}_l$ which is interpreted as an $I \times |\Delta|J$ matrix. The expansion operator $[\![\cdot]\!]$ extracts patches around each spatial location and flattens them into vectors that become the rows of a matrix: $[\![\mathbf{A}_{l-1}]\!]$ is a $M|\mathcal{T}| \times J|\Delta|$ matrix.

Similar to the feedforward networks, the bias (if used) can be prepended to the weights matrix as $\bar{\mathbf{W}}_l = (\mathbf{b}_l \quad \mathbf{W}_l)$ and a homogeneous column of ones to the expanded activations as $[\![\mathbf{A}_{l-1}]\!]_H = (\mathbf{1} \quad [\![\mathbf{A}_{l-1}]\!])$. This constructions allows the forward pass to be written as

$$\mathbf{S}_l = [\![\mathbf{A}_{l-1}]\!]_H \bar{\mathbf{W}}_l^\top, \tag{15}$$

$$\mathbf{A}_l = \phi\left(\mathbf{S}_l\right), \tag{16}$$

from which the factors are computed as

$$\mathbf{\Omega}_l = \mathbb{E}\left[[\![\mathbf{A}_l]\!]_H^\top [\![\mathbf{A}_l]\!]_H\right], \tag{17}$$

$$\mathbf{\Gamma}_l = \frac{1}{|\mathcal{T}|}\mathbb{E}\left[\mathcal{D}\mathbf{S}_l^\top \mathcal{D}\mathbf{S}_l\right]. \tag{18}$$

Depthwise Convolutional Layers Depthwise convolutional layers utilize separate kernels for each channel. In this case, $[\![\mathbf{A}_{l-1}]\!]$ is a $M|\mathcal{T}|J \times |\Delta|$ matrix. Otherwise, the factors $\mathbf{\Omega}_{l-1}$ and $\mathbf{\Gamma}_l$ are calculated in the same way as they are for standard convolutional layers.

Eigenvalue Calculation Because $\hat{\mathbf{F}}$ is a block-diagonal matrix, its eigenvalues are simply the combined eigenvalues of each block: $\lambda(\hat{\mathbf{F}}) = \{\lambda(\hat{\mathbf{F}}_l)\}_{l=1}^L$. The eigenvalue calculation for one block $\hat{\mathbf{F}}_l = \mathbf{\Omega}_{l-1} \otimes \mathbf{\Gamma}_l$ is further simplified by first computing the eigenvalues $\lambda(\mathbf{\Omega}_{l-1})$ and $\lambda(\mathbf{\Gamma}_l)$ for each Kronecker factor separately and then returning all pairwise products from the two sets of eigenvalues. For numerical stability, the eigenvalues can first be log-scaled and then all pairwise sums from the two sets are returned. Calculating the eigenvalues requires one forward and backward pass through the network with a mini-batch of data. The computational cost is therefore relatively cheap, especially compared with the cost of fully training a network from scratch.

It is possible for the FIM eigenvalues to be invalid. For example, if the forward propagated activations or backward propagated gradients explode or vanish, then the diagonal entries $\mathbf{\Omega}_{l-1} \otimes \mathbf{\Gamma}_l$ may be undefined. Such invalid values result from activation functions that are unstable. Therefore, invalid FIM eigenvalues provide a good way to filter out bad activation functions.

D Features and Surrogate Details

This section describes how the activation function features were implemented and how the surrogate was constructed.

Calculating FIM Eigenvalues The FIM eigenvalues were calculated for each activation function as discussed in Section 3. The eigenvalues were log-scaled for numerical stability. By definition, the number of eigenvalues is the same as the number of weights in the neural network. To save space, the eigenvalues were binned to histograms. For a layer l with $|\boldsymbol{\theta}_l|$ weights, $\lfloor |\boldsymbol{\theta}_l|/100 \rfloor$ equally sized bins from -100 to 100 were used. One histogram was computed for each layer in a network, and all of the histograms were concatenated together into a single feature vector for a given activation function. In this manner, the total dimensionality was 13,692 for All-CNN-C, 16,500 for ResNet-56, and 11,013 for MobileViTv2-0.5.

Calculating Activation Function Outputs The activation function outputs $y = f(x)$ were calculated for each activation function f by sampling $n =$1,000 values $x \sim \mathcal{N}(0, 1)$ and truncating to the range $[-5, 5]$. The same random inputs were used for all activation functions.

Per-Layer FIM Eigenvalues In Figure 5, the eigenvalues for the entire network are shown for completeness. However, the UMAP representations shown in Figure 4 were produced by keeping the eigenvalues at each layer separate and computing a weighted distance between them (according to Equation 2). As pointed out in the main text, FIM eigenvalues are informative but noisy features. In preliminary experiments, keeping the eigenvalues separate at each layer reduced some of this noise, resulting in a more informative Figure 4 and consequently improving the performance of the search algorithms.

FIM Eigenvalue Features Preliminary experiments aimed to predict activation function performance using common features in the literature, including maximum eigenvalue, minimum eigenvalue, sum of the eigenvalues, and product of the eigenvalues [13]. More recently proposed features, such as (second moment) / (first moment)^2, were also considered [44]. Ultimately, learning the relevant features from the entire eigenvalue distribution was found to be the most flexible and powerful approach.

UMAP Settings UMAP exposes a number of parameters that can be used to customize its behavior [37]. The `metric` parameter determines how distances are computed between points, the `n_neighbors` parameter adjusts the tradeoff between the local and global structure of the data, and the `min_dist` parameter controls the minimum distance between points in the embedding space.

The plots in Figure 4 were produced by computing the distances between FIM eigenvalues and activation function outputs. For the FIM eigenvalues `UMAP(metric='manhattan', n_neighbors=3, min_dist=0.1)` was used, and for the activation function outputs `UMAP(metric='euclidean', n_neighbors=15, min_dist=0.1)` was used. The distance metrics were chosen to implement Equations 2 and 3.

In preliminary experiments, decreasing `n_neighbors` from the default of 15 down to 3 for the FIM eigenvalues qualitatively improved the embedding for the combined features. The combined features were visualized with a union model, i.e. `umap_combined = umap_fim_eigs + umap_fn_outputs` [37].

E Experiment Details

This section specifies the details for the experiments in the main text of the paper. Several variations to the approach presented in the main text were also evaluated in preliminary experiments. The

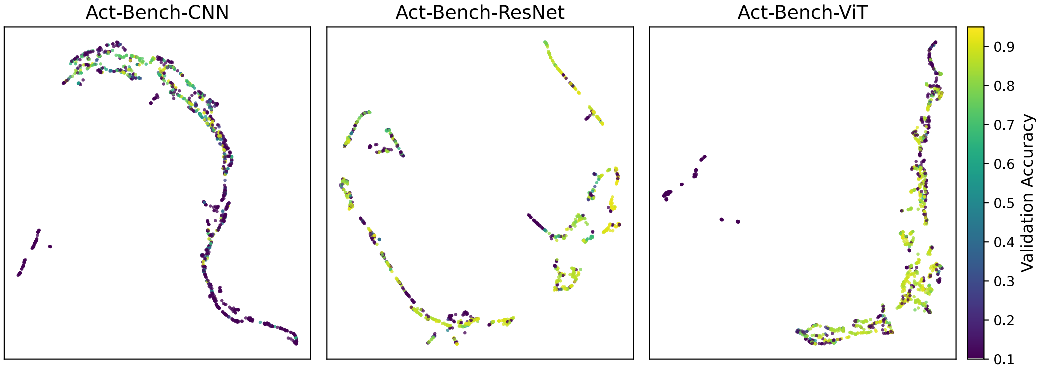

Figure 10: UMAP projections of FIM eigenvalues using the default hyperparameter of `n_neighbors=15`. The embedding is informative but also noisy. Using `n_neighbors=3`, as shown in the main text, improved performance.

approach turned out to be robust to most of them, but the results also justify the choices used for the main experiments.

Training Details For CIFAR-10 and CIFAR-100, balanced validation sets were created by sampling 5,000 images from the training set. Full training details and hyperparameters are listed in Table 4.

Search Implementation In order to predict performance for an unevaluated activation function, the function outputs and FIM eigenvalues must first be computed. Thus, the searches in Section 6 were implemented in three steps. First, activation function outputs for all 425,896 activation functions in the search space were calculated. This computation is inexpensive and easily parallelizable. Second, eight workers operated in parallel to sample activation functions uniformly at random from the search space and calculate their FIM eigenvalues. Third, once the number of activation functions with FIM eigenvalues calculated reached 5,000, seven of the workers began the search by evaluating the functions with the highest predicted performance. The eighth worker continued calculating FIM eigenvalues for new functions so that their performance could be predicted during the search. This setup allowed taking best advantage of the available compute for the regression-type search methods.

The experiments on ImageNet required substantially more compute than the experiments on CIFAR-100. For this reason, all eight workers evaluated activation functions once the number of functions with FIM eigenvalues reached 7,000.

Computing FIM eigenvalues took approximately 26 seconds, 84 seconds, and 37 seconds per activation function for All-CNN-C, ResNet-56, and MobileViTv2-0.5, respectively. This cost is not trivial, but it is well worth it, as the experiments in the main paper show.

Unique Activation Functions Different computation graphs can result in the same activation function (e.g. $\max\{x, 0\}$ and $\max\{0, x\}$). In the benchmark dataset and in the larger search space of Section 6, repeated activation functions were filtered out. 1,000 inputs were sampled $\mathcal{N}(0, 1)$ and truncated to $[-5, 5]$. Two activation functions were considered the same if their outputs were identical.

Improving the Combined UMAP Projection Figure 10 displays a projection of FIM eigenvalues using default UMAP hyperparameters. The plots show the eigenvalues organized in multiple distinct one-dimensional manifolds. Again, FIM eigenvalues are noisy features; there are some clusters of activation functions achieving similar performance, but there are also regions where performance varies widely. As mentioned in the main text, this issue was addressed by reducing the UMAP parameter `n_neighbors` to 3. This change reduced the connectivity of the low-dimensional FIM eigenvalue representation, resulting in a space with many distinct clusters (as seen in Figure 4).

On its own, this setting did not improve the search on the benchmark datasets. However, it did improve performance when the FIM eigenvalues were combined with activation function outputs (as was discussed in Section 4). The reason is that the UMAP model for the activation function outputs did not decrease `n_neighbors`, and so the combined UMAP model relied more on the

Table 4: Training details and hyperparameter values used in the experiments.

All-CNN-C on CIFAR-10 and CIFAR-100	
Batch Size	128
Dropout	0.5
Epochs	25 for `Act-Bench-CNN` and search (Figure 7), 50 for full evaluation (Table 1)
Image Size	32×32
Learning Rate	Linear warmup to 0.1 for five epochs, then linear decay
Mean/Std. Normalization	Yes
Momentum	0.9
Optimizer	SGD
Random Crops	32×32 crops of images padded with four pixels on all sides
Random Flips	Yes
Weight Decay	$1e^{-4}$
Weight Initialization	AutoInit [4]

ResNet-56 on CIFAR-10 and CIFAR-100	
Batch Size	128
Dropout	0.0
Epochs	25 for `Act-Bench-ResNet` and search (Figure 7), 50 for full evaluation (Table 1)
Image Size	32×32
Learning Rate	Linear warmup to 0.1 for five epochs, then linear decay
Mean/Std. Normalization	No
Momentum	0.9
Optimizer	SGD
Random Crops	32×32 crops of images padded with five pixels on all sides
Random Flips	Yes
Weight Decay	$1e^{-4}$
Weight Initialization	AutoInit [4]

MobileViTv2-0.5 on Imagenette and ImageNet	
Batch Size	256
CutMix Alpha [54]	1.0
Epochs	105
Evaluation Center Crop	95%
Image Size	160×160
Learning Rate	Linear warmup from $1e^{-4}$ to $4e^{-3}$ for five epochs, then cosine decay to $1e^{-6}$
Mixup Alpha [55]	0.1
Optimizer	AdamW [34]
RandAugment [9]	Magnitude six, applied twice
Random Resized Crop [49]	Minimum 8% of the original image
Weight Decay	0.02× current learning rate

ResNet-50 on ImageNet	
Batch Size	256
CutMix Alpha [54]	1.0
Epochs	105
Evaluation Center Crop	95%
Image Size	160×160
Learning Rate	Linear warmup from $1e^{-4}$ to $2e^{-3}$ for five epochs, then cosine decay to $1e^{-6}$
Mixup Alpha [55]	0.1
Optimizer	AdamW [34]
RandAugment [9]	Magnitude six, applied twice
Random Resized Crop [49]	Minimum 8% of the original image
Weight Decay	0.02× current learning rate
Weight Initialization	AutoInit [4]

activation function outputs than it did on the FIM eigenvalues. As Figure 4 shows, the activation function outputs are reliable but sometimes project good activation functions to distinct regions in the search space. Introducing extra connectivity into the fuzzy topological representation via the FIM eigenvalues was sufficient to address this issue, bringing good activation functions to common regions of the space.

Increasing the Dimension of the UMAP Projections The UMAP plots show two-dimensional projections of FIM eigenvalues and activation function outputs. Regression algorithms were also trained on five and 10-dimensional projections. These runs resulted in comparable or worse performance. Therefore, the two-dimensional projections were selected in the paper for simplicity and for consistency between the algorithm implementation and figure visualizations.

Gaussian Process Regression As an alternative search method, Gaussian process regression (GPR) was evaluated in activation function search. Several different acquisition mechanisms were used, including expected improvement, probability of improvement, maximum predicted value, and upper confidence bound. The approach worked well, but the results were inconsistent across the different acquisition mechanisms. GPR was also more expensive to run compared to the algorithms in the main text (KNR, RFR, SVR), and so those algorithms were used instead for simplicity and efficiency.

Adjusting k in KNR The initial experiments with the KNR algorithm used $k = 3$. Experimenting with $k = \{1, 5, 8\}$ did not reliably improve performance, so $k = 3$ was kept.

Uniformly Spaced Inputs for Activation Function Outputs In an alternative implementation, equally spaced inputs from -5 to 5 were given to the activation functions instead of normally distributed inputs. This variation did not noticeably change the quality of the embeddings nor the performance of the search algorithms. Therefore, normal inputs were used for consistency with Equation 3. Figure 3 is the only exception; it used 80 inputs equally spaced from -5 to 5 and increased the UMAP parameter `min_dist` to 0.5. These settings improved the quality of the reconstructed activation functions in the plot.

F Future Work

This paper demonstrated that FIM eigenvalues and activation function outputs are efficient and reliable features that can predict performance of activation functions accurately. This finding enabled discovering better activation functions for various tasks, improving the state of the art in machine learning. Because the technique is efficient, it was possible to scale it up to large datasets such as ImageNet. These discoveries inspire several avenues for future research, discussed below.

New Search Spaces The PANGAEA search space was used in this paper because it is known to work well for deep architectures [5]. In the future it will be interesting to explore search spaces with different unary, binary, and n-ary operators. Beyond computation graphs, it may also be possible to apply techniques in this paper to optimize continuous vector representations of activation functions [1, 40].

Exploration vs. Exploitation The KNR approach was utilized to search for new activation functions because it performed well on the benchmark datasets (Section 5). In the future, it will be interesting to consider other algorithms and analyze their tradeoffs between exploration and exploitation. For example, in a resource-constrained environment where improvement is needed quickly, a more exploitative approach could be used to find an improved activation function in a short time. On the other hand, if substantial compute is available, an approach that focuses on exploration could be used to discover activation functions that perform well but are maximally different from functions used in modern architectures (Figure 8b). Novelty search [30] could serve as a suitable approach, and such discoveries could further understanding of how neural networks utilize different kinds of activation functions to learn.

Optimizing Multiple Activation Functions In a typical neural network design, the same activation function is used throughout the network. However, recent work has shown that it may be beneficial to have different activation functions at different locations, and further, that it may be useful to have different activation functions in the early and late stages of training [5]. Indeed, many hybrid architectures use Swish in convolutional layers and ReLU in attention layers [38]. Unfortunately, it is difficult to design these strategies manually, and so practitioners often use a single activation function for simplicity.

The techniques proposed in this paper may provide an avenue toward optimizing multiple activation functions in tandem. For example, the features for multiple candidate activation functions could be concatenated into a single feature vector, and this vector could be projected with UMAP to a low-dimensional space where performance prediction is more straightforward.

Optimizing Parametric Activation Functions Parametric activation functions have learnable parameters that allow them to refine their shape via gradient descent. In some tasks, this extra flexibility results in better performance over fixed activation functions [5]. The techniques introduced in this paper can be readily extended to optimizing the design of parametric activation functions as well. Because the surrogate considers the state of the network and activation function at initialization, it is possible to predict the performance by treating the activation function parameters as fixed to their initial values.

However, it may be possible to extend this idea further. Because the activation function parameters are implemented as neural network weights, each parameter will have a corresponding FIM eigenvalue. These extra eigenvalues will provide the surrogate with additional information that may help predict the performance more accurately.

For simplicity, current parametric activation functions usually initialize their parameters either to be 1.0 or to approximate some existing activation function, and the initialization is usually the same throughout the network. This method is likely suboptimal; the surrogate introduced in this paper could provide a smarter approach. By adjusting the initial parameter values and observing the change in predicted performance, the surrogate can be used to find better initializations, including different ones at different layers in the network. This contribution could make parametric activation functions even more powerful.

Optimizing Other Aspects of Neural Network Design By fixing the neural network architecture and varying the activation function, this paper showed that it is possible to use FIM eigenvalues to infer future performance. As the FIM is a fundamental quantity in machine learning, it may be possible to apply a similar strategy to optimize other aspects of neural network design, such as normalization layers, loss functions, or data augmentation strategies [8, 16, 17, 33]. If a meaningful distance metric between such objects can be defined, then UMAP could be used to map them to a low-dimensional space where performance prediction is much simpler.

Similarly, one could use the FIM eigenvalues to optimize alternate objectives beyond accuracy. Robustness is a particularly interesting objective, because the FIM can be used to describe a neural network's robustness to small parameter perturbations. Other objectives, such as interpretability, fairness, or inference cost, could also be considered. For example, one could consider a multidimensional regression approach where instead of just predicting accuracy, the surrogate would predict each of these quantities separately. Such a method could present the user with a Pareto front of activation functions involving tradeoffs between these quantities.

Reverse Engineering Activation Functions UMAP was used to project activation functions to a low-dimensional space, and regression algorithms to predict the performance of activation functions in this space, i.e. to serve as a fitness function for the search. However, it is possible that there is no activation function that maps to the optimum of this fitness landscape. Indeed, because such search spaces are finite, the activation functions do not completely fill them. For example, there are empty regions in Figure 4, corresponding to activation functions outside of the predefined search space.

What should be done if an empty region of the embedding space has a higher predicted fitness than any of the candidate activation functions? In the paper, these regions were simply ignored, and the activation function with the highest predicted fitness was used. However, in the future, it may be possible to create activation functions that map to these empty spaces, an in so doing improve performance. One approach could be based on inverse transforms: Given a coordinate in the low-dimensional embedding space, UMAP can apply an inverse transform and return an object that would have mapped to those coordinates. This technique was already used for visualization in Figure

3. Using this approach, UMAP could generate a hypothetical desired FIM eigenvalue distribution, or a list of activation function outputs.

There are two challenges to this approach. First, because UMAP is a dimensionality-reduction algorithm, different activation functions can map to the same location in the embedding space. Thus, the mapping from embedding space back to activation functions is not well defined. Second, even if UMAP prescribes a FIM eigenvalue distribution that is predicted to result in good performance, it may be difficult to manually design an activation function to satisfy that distribution.

However, a generated list of prescribed activation function outputs is already a good start. From this list, it is possible to construct an activation function that interpolates through these points, either in a piecewise linear fashion, with splines, or using some other standard technique. Even without the corresponding FIM eigenvalues, such an approach could potentially improve the efficiency of novel activation function discovery, and lead to better designs for activation functions in the future.

G Compute Infrastructure

The experiments in this paper were implemented using an AWS `g5.48xlarge` instance with eight NVIDIA A10G GPUs. The total compute cost for the search experiments in Section 6 was 14.49 GPU-hours for All-CNN-C on CIFAR-100, 21.67 GPU-hours for ResNet-56 on CIFAR-100, and 196.25 GPU-days for MobileViTv2-0.5 on ImageNet. This cost includes the time to train the eight baseline activation functions and then to evaluate 100 additional functions. The instance ran in Oregon (`us-west-2`) and was powered by renewable energy, so the experiments for this paper contributed no carbon emissions.