CS 4783 Final Project

Ian Huang

May 23rd, 2022

1 Introduction

The subject of generalizability in neural networks remains mostly a mystery in current literature. In light of this, we decided to take successful complexity measures in traditional machine learning setting and investigate their correlation with actual empirical generalization gap in neural networks. Using *Fantastic Generalization Measures and Where to Find Them* [1]- henceforth referred to as the Google paper- as a springboard, we measured the Kendall correlation between theoretical complexities (VC-dimension and 3 norm-based measures) and empirical generalization gaps over two distinct convolutional neural network architectures on around 30 models each. We find that most traditional complexity measures are **poor** measurements of generalization for neural networks.

2 Background

We first describe a neural network in plain English. A neural network is a set of layers joined together by a feed-forward algorithm that takes in an input and outputs a prediction after passing through the intermediate layers. The losses are then back-propagated to refine the weights of the nodes in each layer. Between each layer, an activation function is applied to determine the "significance" of each node in that layer. Its multi-layer nature means a large number of parameters are used to produce a prediction.

More formally, define a training set $S = \{(\mathbf{X}_i, y_i)\}_{i=1}^n$ where $\mathbf{X}_i \in \mathcal{X}$ is the input data and $y_i \in \{1, \cdots, m\}$ is the corresponding multiclass label. Then, a feed-forward neural network on S of depth d is a functional map $f_{\mathbf{w}(\theta)} : \mathcal{X} \to \mathbb{R}^m$, where $\mathbf{w}(\theta) = (\mathbf{W}_1, \cdots, \mathbf{W}_d)$ is the weight parameter,

$\mathbf{W}_i$ is the weight tensor of layer $L_i \in \mathcal{L} = \{L_i\}_{i=1}^d$, the set of layers, and $\theta \in \Theta = \theta_1 \times \cdots \times \theta_q$ is a hyperparameter choice that generates $\mathbf{w}$ from the hyperparameter space Θ where each θ_i corresponds to a hyperparameter axis (say `batch-size` or `width`). Essentially, the hyperparmeter choice θ defines the network architecture, which gives a unique weight parameter $\mathbf{w}(\theta)$ for the network $f_{\mathbf{w}(\theta)}$. As with traditional machine learning, the goal is to minimize expected loss over the true, unknown distribution $\mathscr{D}$ according to some loss function ℓ:

$$\arg\min_{\theta \in \Theta} L(f_{\mathbf{w}(\theta)}) := \mathbb{E}_{(\mathbf{X},y)\sim\mathscr{D}} \left[\ell\left((\mathbf{X}, y), \mathbf{w}(\theta)\right)\right] \tag{1}$$

and as per tradition we estimate $L(f_{\mathbf{w}(\theta)})$ using the sample S:

$$\hat{L}_S(f_{\mathbf{w}(\theta)}) := \frac{1}{n}\sum_{i=1}^{n} \ell\left((\mathbf{X}_i, y_i), \mathbf{w}(\theta)\right) \tag{2}$$

Under the network architecture θ, we define $g(\theta) = L(f_{\mathbf{w}(\theta)}) - \hat{L}_S(f_{\mathbf{w}(\theta)})$ as the *generalization gap*.

Naturally, we want to know the correlation between theoretical complexity bounds and actual generalization gap. There are a plethora of theoretical metrics to choose from, and we found VC dimension and norm-based complexity as excellent introductions. They are both very successful measures in traditional machine learning setting, and we believe this very property makes them the ideal candidate to illuminate the difference bewteen neural network and classical learning. In particular, we found the Google paper [1] as an excellent starter, while *Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear activation neural networks* [2] and *Spectrally-normalized margin bounds for neural networks* [3] provide a more detailed background on various norm complexity measures necessary for our investigation.

3 Measuring Correlation

So far, we mentioned the phrase **correlation** a lot without concretely identifying what it means and how to compute it. Loosely speaking, given a set of models generated from the hyperparameter space Θ, their corresponding generalization gap $G = \{g(\theta) : \theta \in \Theta\}$, and their complexity measures $U = \{\mu(\theta) : \theta \in \Theta\}$, *correlation* determines the directional consistency of U with G. In other words, a perfect complexity measure satisfies

$\mu(\theta_1) > \mu(\theta_2) \iff g(\theta_1) > g(\theta_2)$. A popular method to capture the extent to which that statement is true is through **Kendall's rank-correlation coefficient** τ [4]. Kendall coefficient closely mirrors this idea and is mathematically defined as:

$$\tau(T) = \frac{1}{|T|(|T|-1)} \sum_{(\mu_1,g_1)\in T} \sum_{(\mu_2,g_2)\in T/\{(\mu_1,g_1)\}} sign(\mu_1 - \mu_2) sign(g_1 - g_2) \quad (3)$$

where $T := \cup_{\theta\in\Theta}(\mu(\theta), g(\theta))$. We have positive correlation when $\tau > 0$. Also, note importantly that Kendall correlation only looks at the signs (i.e. the ordinal rank of the computed values), so we can safely eliminate all order-preserving constants in our computation of complexity measures U [4]. This is a key insight that greatly simplifies our code.

4 Complexity Measure Choices

4.1 VC Dimension

The first measure we will consider is VC dimension. Recall that in binary classification, we define VC dimension of a model class $\mathcal{F}$ as

Definition 1. *$VC(\mathcal{F}) := \sup_m \{\Pi_{\mathcal{F}}(m) = 2^m\}$, where $\Pi_{\mathcal{F}}(m)$ is the growth function such that $\Pi_{\mathcal{F}}(m) := \max_{\mathbf{x}\in\mathcal{X}} |\{f(\mathbf{x}) : f \in \mathcal{F}\}|$*

There are two potential concerns. The first is that this definition only works for binary classification. The second is that this quantity seems outright impossible to compute. For the first concern, the definition could be extended to multiclass problems with some artifacts using graph dimension [5]. For the second concern, Bartlett et al. [2] was able to show a tight VC-dimension bound for the class of neural network $\mathcal{F}$ with number of weights W and number of layers L to be $\mathcal{O}(WL\log(L))$. Hence, we can simply compute the complexity measure.

$$\mu_{VC} \propto WL\log(L) \quad (4)$$

Recall that the number of weights for a convolutional network $f : \mathcal{X} \to \mathbb{R}^m$ with input channels c_i and kernel size k_i at each layer L_i is just

$$W = \sum_{i=1}^{d} k_i^2 c_{i-1}(c_i + 1) \quad (5)$$

[6]. This is actually the VC-measure used by the Google researchers [1], but we opted for μ_{VC} to stay true with Bartlett's original breakthrough.

The main thing to note with VC dimension is that it **only** depends on the explicit network architecture (i.e. the depth or width of the network), and so many have argued that VC dimension neglects the nuances of other hyperparameter choices with potential impact on regularization and thus challenged its usefulness in understanding neural network generalization [7]. Either way, VC-dimension is included as a homage to traditional machine learning.

4.2 Network Norm

The other measure we will examine is norm-based complexity measures. In classical machine learning, large overall weights tend to imply overfitting on training and thus poor generalization on testing. Norm-based measurements seek to apply the same principle in the context of neural networks: the larger the norm of the network weights, the poorer the theoretical performance on unseen data. Considering the success of norm-based arguments in traditional learning, it only makes sense to examine its viability in neural networks.

Three types of norms are considered:

(1) **Parameter norm**. This takes on the form:

$$\mu_{param} = \sum_{i=1}^{d} ||\mathbf{W}_i||_F^2 \tag{6}$$

μ_{param} is perhaps the most vanilla norm-based measure. By simpling summing up the frobenius norm of the weight matrices in each layer, we can get a crude sense of network complexity and the degree of regularization [3].

(2) **Spectral norm from the origin**. This takes on the form:

$$\mu_{spectral\text{-}orig} \propto \prod_{i=1}^{d} ||\mathbf{W}_i||_2^2 \sum_{j=1}^{d} \frac{||\mathbf{W}_j||_F^2}{||\mathbf{W}_j||_2^2} \tag{7}$$

Based on the work of Pitas et al. [8], μ_{spec} is a spectral norm measure that has been adjusted specifically for convolutional networks. The key thing to note is that μ_{spec} is positively correlated to both the two-norm

and the Frobenius norm of the weight matrices, so one should expect the measure will increase significantly with network complexity.

(3) **Sum of squared Frobenius norm**: This takes on the form:

$$\mu_{spec} = d \left(\prod_{i=1}^{d} ||\mathbf{W}_i||_F^2 \right)^{1/d} \tag{8}$$

μ_{spec} is a variation on the family of Frobenius product norm $\mu \propto \prod_{i=1}^{d} ||\mathbf{W}_i||_F^2$, and we choose μ_{spec} over a more traditional measure because it is more depth-independent with its $\frac{1}{d}$ adjustment, whereas all the previous norm-based measures clearly increase with depth. The inclusion of this measurement adds an additional flavor to our analysis.

5 Experimental Setup

5.1 Dataset Choices

We choose MNIST and CIFAR-10. They are standard dataset choices whose moderate training sizes work acceptably with our experimental capacity.

5.2 Network Architectures

We use a baseline convolutional neural network- henceforth referred to as `conv`- that is architecturally similar to the office hour tutorial **and** a Network-in-Network block implementation- henceforth referred to as `NiN`- inspired by Gao et al [9]. `NiN` block is shown to achieve competitive performance on modern image classification benchmarks, so we decided to try it out for academic curiosity.

A `NiN` block consists of three parts: one part with 3×3 convolution of stride 2 followed by two parts with 1×1 convolution of stride 1. Dropout is applied at the end of each block, and at the end there is a 1×1 convolution reducing the dimension to the class number followed by global average pooling. We can create a network f of arbitrary depth d by stacking d `NiN` blocks. To ensure homogeneity, all blocks have the same number of `out-channels`. This design of `NiN` block closely mirrors that of the Google researchers, which is shown to be quite successful.

5.3 Hyperparameter Choices

5.3.1 `conv`

We choose 4 common hyperparameter categories related to architecture design for a total of $2^2 \times 3^2 = 36$ models to test on CIFAR-10 dataset. In particular, we have `depth` $= [2, 4, 6]$, `width` $= [32, 64, 96]$, `dropout` $= [0.25, 0.5]$, and `learning-rate` $= [0.5, 1]$. We train under `epochs` $= 5$ and `batch-size` $= 64$ using Adadelta optimizer. Since we do not vary our optimizer choices, we do not need to worry about speed of convergence for different optimizers, which will require optimizer-specific `epochs`. We tried many different `epochs` and `batch-size` combinations and this one is the best because it converges decently, strikes the balance between convergence and overfitting, and fulfills our experimental capacity. For MNIST dataset, we set `epochs` $= 4$ because MNIST is 'easier' than CIFAR-10 and requires less iterations to converge. Everything else is identical.

`depth` and `width` must be chosen to see variation in μ_{VC}. `dropout` and `learning-rate` are chosen because both are essentially forms of regularization with intuitive impact on generalization and are theorized by the community to have a large impact as well [10].

5.3.2 `NiN`

For `NiN` design, we train on 3 hyperparameter categories for a total of $3^3 = 27$ models: `width` $= [32, 64, 96]$, `learning-rate` $= [0.5, 75, 1]$, and `dropout` $= [0.25, 0.5, 0.75]$ with `batch-size` $= 32$ and `epochs` $= 11$ using Adadelta optimizer. We apply the same set up for both MNIST and CIFAR-10 datasets.

Most notably, we did not vary `depth` because after many attempts with different architectures- such as increasing `epochs` and `learning-rate`-, we realized that `NiN` blocks are not converging in losses for `depth` > 2 as there are too many parameters to learn in too few passes (recall a `NiN` block of depth 1 has 3 layers). This also explained the decrease in `batch-size` and increase in `epochs` compared to `conv` design in hopes of better fitting. We suspect that with a very high number of `epochs`, the network will learn the weights more meaningfully, but that will obviously take even longer to train than the 15 hours on each dataset we already spent.

6 Summary Results

We first trained 36 models on the MNIST dataset and calculated the Kendall Tau's correlation between every complexity measure and the empirical generalization gap. As observed from the figure below, $\mu_{spectral\text{-}orig}$ and μ_{spec} show somewhat significant negative correlations with generalization gap, while the correlation magnitude for μ_{param} is slightly negative μ_{VC} is barely positive. We realized throughout the training process that some of the models might have been underfitted, as reflected by the fact that train loss is higher than test loss (refer to Appendix for details). We unfortunately did not have sufficient time to rerun the experiment, but decided to increase the number of `epochs` when training on CIFAR-10 to allow better learning.

Most models seem to be more adequately fitted for CIFAR-10. When we look at the correlation measures, we see that $\mu_{spectral\text{-}orig}$ and μ_{spec} have even more negative correlations and are consistent with our MNIST experiment. μ_{param} becomes more strongly correlated with generalization gap, while μ_{VC} now shows a rather negative correlation. Overall, we observe that all but μ_{param} are **poor** measurements for generalization. This finding broadly agrees with the analysis by the Google researchers [1]. We theorize on possible reasons in the final section. We also provide the full results in Appendix.

Kendall Correlation				
	μ_{param}	$\mu_{spectral\text{-}orig}$	μ_{spec}	μ_{VC}
MNIST	-0.0825	-0.222	-0.340	0.016
CIFAR-10	0.365	-0.235	-0.406	-0.127

Table 1: Kendall correlations between complexity measures and generalization gap obtained by training 36 models each on the MNIST and CIFAR-10 datasets using `conv` design

For `NiN` block, even when we limit `depth` = 2, we still did not obtain satisfactory training losses largely due to the reasons described in the previous part. Since the networks did not properly learn, we decided to not include the results in the main analysis to prevent further confusion. Of course, the unsuccessful results can be found in the Appendix.

7 On Negative Correlations

7.1 VC-Dimension

In a sense, a negative Kendall correlation on VC-dimension is a welcoming sight because it confirms the observed phenomenon that overparameterization (i.e. increasing depth and width) leads to better generalization in neural networks.

To really convince ourselves that VC-dimension is lackluster in neural network settings, we really want to look at the loss bound for VC dimension for a neural network $f_{\mathbf{w}(\theta)} : \mathcal{X} \to \mathbb{R}^m$ of depth d. It has been shown that for any $\delta > 0$, with probability $1 - \delta$ over $S : |S| = n$, we have

$$L(f_{\mathbf{w}(\theta)}) \leq \hat{L}(f_{\mathbf{w}(\theta)}) + 4000\sqrt{\frac{Wd\log\left(6dn\right)^3}{m}} + \sqrt{\frac{\log(1/\delta)}{m}} \tag{9}$$

[1] with $W = \sum_{i=1}^{d} k_i^2 c_{i-1}(c_i + 1)$ from above. In our largest model we have approximately $W_{max} = 3 \cdot 10^6$ weights and our smallest model we have approximately $W_{min} = 10^4$ weights, and so with moderate output dimension ($m = 10$ for CIFAR-10), clearly this does not produce particularly meaningful bound unless the output dimension is extremely complex.

But why the failure intuitively? A potential reason is that VC-dimension ignores the intricate architecture choices like `dropout` or `learning-rate` which can control the degree of regularization and hence have a tangible impact on generalizability. Another potential reason is that having more weights and parameters can in fact pick up subtle signals and create more robust snippets of pixel pattern in image recognition. These are some plausible speculations for an observed phenomenon that remains theoretically unconfirmed in current literature [2]. The point is that a negative Kendall correlation for VC-dimension in our CIFAR-10 result is expected.

7.2 Norm-Based Measures

Spectral complexity $\mu_{spectral\text{-}orig}$ is strongly negatively correlated with generalization gap and so is the Frobenius norm μ_{spec}, while the parameter norm μ_{param} is the most successful complexity measure of all. This is kind of surprising because we usually look at control on norms, traditional machine learning or not, as a direct pathway to regularization. In fact, this is pre-

cisely the intuition behind `dropout` by randomly setting weights to 0 with certain probability, which is a widely popular practice.

A sensible reason for this phenomenon is that again network norms are in general positively correlated with `depth`, and from the previous discussion on VC dimension we see that increasing `depth` tends to improve generalization due to overparameterization. Even then, we would expect the sum of squared Frobenius norm μ_{spec} to be more positively correlated as it is arguably more `depth` independent, which did not happen in our experiment. This could be explained by the Google researchers' finding that the results on Frobenius norm is especially susceptable to weight matrices initialiation [1], which we assumed to be $\mathbf{W}_i^0 = \mathbf{0}, \forall i$.

Overall, the apparent negative correlation of norm-based measures could be potentially explained by their affinity to parameterization, and we postulate that some norm-based measures would perform better than other norm-based measures due to their removed nature from the explicit network architecture. A natural extension to this investigation would be to examine more norm-based measures to test this hypothesis (we only considered 3).

References

[1] Yiding Jiang et al. *Fantastic Generalization Measures and Where to Find Them*. 2019. DOI: 10.48550/ARXIV.1912.02178. URL: https://arxiv.org/abs/1912.02178.

[2] Peter L. Bartlett et al. "Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks". In: (2017). DOI: 10.48550/ARXIV.1703.02930. URL: https://arxiv.org/abs/1703.02930.

[3] Dylan J. Foster, Peter Bartlett, and Matus Telgarsky. *Spectrally-normalized margin bounds for neural networks*. 2017. DOI: 10.48550/ARXIV.1706.08498. URL: https://arxiv.org/abs/1706.08498.

[4] Alexei Stepanov. *On the Kendall Correlation Coefficient*. 2015. DOI: 10.48550/ARXIV.1507.01427. URL: https://arxiv.org/abs/1507.01427.

[5] Yu. Maximov and D. Reshetova. "Tight risk bounds for multi-class margin classifiers". In: *Pattern Recognition and Image Analysis* 26.4 (Oct. 2016), pp. 673–680. DOI: 10.1134/s105466181604009x. URL: https://doi.org/10.1134%2Fs105466181604009x.

[6] Thomas Unterthiner et al. *Predicting Neural Network Accuracy from Weights*. 2020. DOI: 10.48550/ARXIV.2002.11448. URL: https://arxiv.org/abs/2002.11448.

[7] Linu Pinto, Sasi Gopalan, and P. Balasubramaniam. "On the stability and generalization of neural networks with VC dimension and fuzzy feature encoders". In: *Journal of the Franklin Institute* 358.16 (2021), pp. 8786–8810. ISSN: 0016-0032. DOI: https://doi.org/10.1016/j.jfranklin.2021.08.023. URL: https://www.sciencedirect.com/science/article/pii/S001600322100507X.

[8] Konstantinos Pitas, Mike Davies, and Pierre Vandergheynst. *PAC-Bayesian Margin Bounds for Convolutional Neural Networks*. 2018. DOI: 10.48550/ARXIV.1801.00171. URL: https://arxiv.org/abs/1801.00171.

[9] Jianxi Gao et al. "Robustness of a Network of Networks". In: *Physical Review Letters* 107.19 (Nov. 2011). DOI: 10.1103/physrevlett.107.195701. URL: https://doi.org/10.1103%2Fphysrevlett.107.195701.

[10] Ibrahem Kandel and Mauro Castelli. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset". In: *ICT Express* 6.4 (2020), pp. 312–315. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2020.04.010. URL: https://www.sciencedirect.com/science/article/pii/S2405959519303455.

A Appendix

	batch_size	depth	dropout	epochs	lr	width	l_train	l_test	param_norm	spectral_orig	spec	vc_dim
0	64	2	0.25	4	0.5	32	0.0237012	0.0305572	206.44044	1272739.8	6.3903906	853152.92
1	64	2	0.25	4	0.5	64	0.0184791	0.0316769	237.76079	3350128.8	6.0614783	1717467.6
2	64	2	0.25	4	0.5	96	0.0163822	0.0340887	261.81546	5688747.5	5.8530321	2594973.8
3	64	2	0.25	4	1	32	0.0183084	0.0323299	362.13449	4951426.5	7.1519215	853152.92
4	64	2	0.25	4	1	64	0.0137383	0.0309847	404.43939	10959607	6.6401707	1717467.6
5	64	2	0.25	4	1	96	0.0144758	0.029682	422.55389	18377352	6.4945956	2594973.8
6	64	2	0.5	4	0.5	32	0.0351857	0.0368964	241.31744	1415407.3	6.2036364	853152.92
7	64	2	0.5	4	0.5	64	0.0272639	0.0331103	281.49747	3585575.8	5.8083095	1717467.6
8	64	2	0.5	4	0.5	96	0.0222403	0.0310908	318.44012	7646679.5	5.7042001	2594973.8
9	64	2	0.5	4	1	32	0.0282694	0.0327129	451.5112	4839537	6.731932	853152.92
10	64	2	0.5	4	1	64	0.0247191	0.0333401	531.64557	13771515	6.5140244	1717467.6
11	64	2	0.5	4	1	96	0.0224749	0.0317581	564.60602	23751848	6.2804685	2594973.8
12	64	4	0.25	4	0.5	32	0.0291199	0.0344614	225.79228	322071096	7.5951009	1782688
13	64	4	0.25	4	0.5	64	0.0213264	0.0282155	284.91403	3.004E+09	7.3626363	3649888.6
14	64	4	0.25	4	0.5	96	0.0202752	0.0316645	336.78207	1.128E+10	7.2285596	5606557.2
15	64	4	0.25	4	1	32	0.022995	0.0296943	392.65698	2.668E+09	8.2876142	1782688
16	64	4	0.25	4	1	64	0.017778	0.0256357	467.88556	2.641E+10	8.1147482	3649888.6
17	64	4	0.25	4	1	96	0.0160413	0.0274196	526.02728	7.056E+10	7.9089539	5606557.2
18	64	4	0.5	4	0.5	32	0.0350263	0.0351384	266.36725	496410336	7.5474684	1782688
19	64	4	0.5	4	0.5	64	0.0262601	0.0292679	331.64526	4.313E+09	7.4236271	3649888.6
20	64	4	0.5	4	0.5	96	0.0260502	0.0293794	390.01245	1.661E+10	7.2866851	5606557.2
21	64	4	0.5	4	1	32	0.0297555	0.0288054	489.08261	5.91E+09	8.4831598	1782688
22	64	4	0.5	4	1	64	0.0238689	0.0244632	587.71082	3.875E+10	8.1546518	3649888.6
23	64	4	0.5	4	1	96	0.0256732	0.0335418	687.56061	1.6E+11	8.0644281	5606557.2
24	64	6	0.25	4	0.5	32	0.0272863	0.033768	253.59206	9.423E+10	9.3886142	2557528.6
25	64	6	0.25	4	0.5	64	0.0237531	0.0321888	339.13809	2.915E+12	9.3182703	5355033
26	64	6	0.25	4	0.5	96	0.0214541	0.0323369	410.55338	2.221E+13	9.2384573	8400901.6
27	64	6	0.25	4	1	32	0.0251369	0.0285592	430.74496	1.441E+12	10.150411	2557528.6
28	64	6	0.25	4	1	64	0.0210273	0.0301217	526.03064	2.92E+13	10.011789	5355033
29	64	6	0.25	4	1	96	0.0218742	0.0305216	614.96027	2.361E+14	9.9895402	8400901.6
30	64	6	0.5	4	0.5	32	0.0362975	0.0359495	292.95859	1.393E+11	9.4848136	2557528.6
31	64	6	0.5	4	0.5	64	0.0312396	0.0333146	393.02539	4.944E+12	9.3862994	5355033
32	64	6	0.5	4	0.5	96	0.0282281	0.0299567	475.7854	4.18E+13	9.2773987	8400901.6
33	64	6	0.5	4	1	32	0.0293308	0.0278848	535.9173	2.973E+12	10.286416	2557528.6
34	64	6	0.5	4	1	64	0.026719	0.0256431	678.86383	1.005E+14	10.085598	5355033
35	64	6	0.5	4	1	96	0.0236427	0.0262192	771.69318	4.874E+14	9.9563169	8400901.6

Figure 1: Training Results of 36 models under `conv` Design on MNIST with reported model parameters, losses, and various complexity measures.

	batch_size	depth	dropout	epochs	lr	width	l_train	l_test	param_norm	spectral_orig	spec	vc_dim
0	64	2	0.25	5	0.5	32	0.7298327	0.9172845	430.45184	3278735.3	6.404496	1158434.2
1	64	2	0.25	5	0.5	64	0.5657241	0.8797722	572.27869	17844732	6.255329	2328030.1
2	64	2	0.25	5	0.5	96	0.4809456	0.8813237	668.15717	41029716	6.1127339	3510817.6
3	64	2	0.25	5	1	32	0.6853351	0.939209	894.01111	13096864	6.231571	1158434.2
4	64	2	0.25	5	1	64	0.5035356	0.9107766	1158.8026	74620896	6.1869958	2328030.1
5	64	2	0.25	5	1	96	0.4806226	0.9215692	1249.6771	124406824	5.8563731	3510817.6
6	64	2	0.5	5	0.5	32	0.8278483	0.9270282	510.56296	2511448.8	5.7987891	1158434.2
7	64	2	0.5	5	0.5	64	0.689349	0.8500607	682.03546	13246584	5.6069992	2328030.1
8	64	2	0.5	5	0.5	96	0.5993809	0.8144254	828.64886	35449556	5.4713831	3510817.6
9	64	2	0.5	5	1	32	0.7877415	0.9169566	1090.6167	9736674	5.5008272	1158434.2
10	64	2	0.5	5	1	64	0.653376	0.8427216	1363.2542	47292440	5.3920867	2328030.1
11	64	2	0.5	5	1	96	0.6051705	0.8289294	1565.08	102415256	5.1961377	3510817.6
12	64	4	0.25	5	0.5	32	0.8153525	0.9272111	367.38312	1.906E+09	7.8456986	2470810
13	64	4	0.25	5	0.5	64	0.6736981	0.8603302	524.96844	3.269E+10	7.5667817	5026132.5
14	64	4	0.25	5	0.5	96	0.5925077	0.8374626	630.87354	1.447E+11	7.4751514	7670923.2
15	64	4	0.25	5	1	32	0.7454069	0.9120377	729.83124	2.385E+10	7.9311782	2470810
16	64	4	0.25	5	1	64	0.5890954	0.8680484	986.21393	4.021E+11	7.7469914	5026132.5
17	64	4	0.25	5	1	96	0.5101001	0.8411302	1163.4974	1.801E+12	7.6458227	7670923.2
18	64	4	0.5	5	0.5	32	0.8678231	0.9328799	440.78049	1.602E+09	7.5288564	2470810
19	64	4	0.5	5	0.5	64	0.7431773	0.8462488	605.42041	3.093E+10	7.3154508	5026132.5
20	64	4	0.5	5	0.5	96	0.6463235	0.7780476	739.93536	1.548E+11	7.2137967	7670923.2
21	64	4	0.5	5	1	32	0.8241315	0.902052	863.69702	2.083E+10	7.7475561	2470810
22	64	4	0.5	5	1	64	0.6930684	0.8364809	1182.572	3.377E+11	7.4782963	5026132.5
23	64	4	0.5	5	1	96	0.6060991	0.784052	1438.137	1.675E+12	7.3851765	7670923.2
24	64	6	0.25	5	0.5	32	0.9638292	1.0238298	325.49466	3.313E+11	9.2526408	3625381
25	64	6	0.25	5	0.5	64	0.8013675	0.9083961	472.23044	2.326E+13	9.2448795	7490737.8
26	64	6	0.25	5	0.5	96	0.7870002	0.8949658	553.99567	1.356E+14	9.0848171	11604459
27	64	6	0.25	5	1	32	0.9035552	1.0042758	569.90393	9.91E+12	9.7426328	3625381
28	64	6	0.25	5	1	64	0.7484513	0.8897242	816.38281	5.215E+14	9.6029147	7490737.8
29	64	6	0.25	5	1	96	0.7165771	0.8869155	977.32391	4.077E+15	9.511613	11604459
30	64	6	0.5	5	0.5	32	1.0680678	1.1079558	365.82706	2.884E+11	9.1566021	3625381
31	64	6	0.5	5	0.5	64	0.9206439	0.9803575	519.88123	1.929E+13	9.0155286	7490737.8
32	64	6	0.5	5	0.5	96	0.8157635	0.8842206	641.06812	1.722E+14	8.9728945	11604459
33	64	6	0.5	5	1	32	0.9565123	1.0182536	703.08557	1.093E+13	9.58179	3625381
34	64	6	0.5	5	1	64	0.839579	0.9252735	929.01154	4.061E+14	9.3997273	7490737.8
35	64	6	0.5	5	1	96	0.856283	0.9574112	1108.4417	3.149E+15	9.3344014	11604459

Figure 2: Training Results of 36 models under `conv` Design on CIFAR-10 with reported model parameters, losses, and various complexity measures.

	batch_size	depth	dropout	epochs	lr	width	l_train	l_test	param_norm	spectral_orig	spec	vc_dim
0	32	2	0.25	11	0.5	32	0.6638502725124359	0.6479158996582032	338.1183166503906	1310221172736.0	9.388308093620962	65934.31174750689
1	32	2	0.25	11	0.5	64	0.42600806194941204	0.41312530822753907	432.76373291015625	8337419403264.0	9.45644392340078	256126.92776127552
2	32	2	0.25	11	0.5	96	0.1621057820888857	0.1551095703125	808.564208984375	381291790860288.0	9.938582764799131	570624.537116329
3	32	2	0.25	11	0.75	32	0.517621221001943	0.5143399536132812	536.706298828125	24930405842944.0	10.127935199306236	65934.31174750689
4	32	2	0.25	11	0.75	64	0.17320009362896283	0.16649484100341796	811.1844482421875	376464251813888.0	10.396069035685134	256126.92776127552
5	32	2	0.25	11	0.75	96	0.10747653924394399	0.11419197998046875	1095.48486328125	2809184328351744.0	10.800814753374645	570624.537116329
6	32	2	0.25	11	1.0	32	2.3025851249694824	2.302585205078125	66.57447814941406	33621238.0	7.433462206612404	65934.31174750689
7	32	2	0.25	11	1.0	64	0.12600407210811973	0.12915889587402343	1074.0289306640625	2029090190131200.0	11.207691618794478	256126.92776127552
8	32	2	0.25	11	1.0	96	0.0805241202559943	0.09325560760498047	1241.4468994140625	5210460647325696.0	11.014686145598102	570624.537116329
9	32	2	0.5	11	0.5	32	0.9359950157324473	0.9118466125488282	295.33184814453125	524216483840.0	9.521302066216892	65934.31174750689
10	32	2	0.5	11	0.5	64	0.3585380607366562	0.3377916564941406	574.320068359375	47765661941760.0	10.082664278554079	256126.92776127552
11	32	2	0.5	11	0.5	96	0.2049581779157122	0.1942812515258789	783.4180908203125	347116719308800.0	10.148300452355286	570624.537116329
12	32	2	0.5	11	0.75	32	0.5302696373383204	0.5122800994873047	458.8304138183594	7971182084096.0	10.202628458726066	65934.31174750689
13	32	2	0.5	11	0.75	64	0.17782804460575183	0.16808419952392578	822.7837524414062	366890648076288.0	10.454385709561054	256126.92776127552
14	32	2	0.5	11	0.75	96	0.14195429151269298	0.1371421936035156 2	1099.9033203125	3213308639313920.0	11.204266445229893	570624.537116329
15	32	2	0.5	11	1.0	32	0.42474513353506727	0.40357452087402346	595.6392211914062	37539595419648.0	10.233473299721407	65934.31174750689
16	32	2	0.5	11	1.0	64	0.14581817890827856	0.13932832489013672	1087.6134033203125	1931609162907648.0	10.77753895145195	256126.92776127552
17	32	2	0.5	11	1.0	96	0.10189477531258016	0.09754865264892579	1356.2969970703125	8850624208699392.0	11.205457556933936	570624.537116329
18	32	2	0.75	11	0.5	32	1.1610103646914165	1.134530633544922	262.48944091796875	275798347776.0	9.287919380323164	65934.31174750689
19	32	2	0.75	11	0.5	64	0.5946336342891058	0.5714503875732422	501.14794921875	22264498880512.0	9.87943722417986	256126.92776127552
20	32	2	0.75	11	0.5	96	0.34073097597360613	0.3233444595336914	733.0216064453125	250902904569856.0	10.090483471940102	570624.537116329
21	32	2	0.75	11	0.75	32	0.9734948219458263	0.9500119384765625	373.7312316894531	2401618100224.0	9.513788898629526	65934.31174750689
22	32	2	0.75	11	0.75	64	0.4264329596837362	0.4077484405517578	718.6396484375	184301905772544.0	10.411890557649407	256126.92776127552
23	32	2	0.75	11	0.75	96	0.233291056428353	0.21401494293212892	1040.5872802734375	2073986523463680.0	10.794339803120295	570624.537116329
24	32	2	0.75	11	1.0	32	0.6848911665916443	0.6619217102050782	546.102294921875	26597040848896.0	10.59678937947566	65934.31174750689
25	32	2	0.75	11	1.0	64	0.3158234571834405	0.2966518768310547	1018.2644653320312	1367078611189760.0	10.93945271655343	256126.92776127552
26	32	2	0.75	11	1.0	96	0.16850862911815445	0.16254881439208985	1361.0465087890625	1.1285679942139904e+16	11.554022790134594	570624.537116329

Figure 3: Training Results of 27 models under `NiN` Design on MNIST with reported model parameters, losses, and various complexity measures.

	batch_size	depth	dropout	epochs	lr	width	l_train	l_test	param_norm	spectral_orig	spec	vc_dim
0	32	2	0.25	11	0.5	32	1.609464685974121	1.61523310546875	211.73451232910156	80939916288.0	9.13451283310355	68623.60246883276
1	32	2	0.25	11	0.5	64	1.478520887451172	1.4844529418945311	348.74267578125	2181138219008.0	9.226857991261085	261505.50920392727
2	32	2	0.25	11	0.5	96	1.365553132133484	1.376162109375	499.04949951171875	23757294403584.0	9.410726950602246	578692.4092803065
3	32	2	0.25	11	0.75	32	1.5024943776702882	1.50861220703125	286.4126892089844	585237323776.0	9.761623213081824	68623.60246883276
4	32	2	0.25	11	0.75	64	1.3749952617263794	1.38738037109375	481.4307556152344	19690841440256.0	9.803921417483632	261505.50920392727
5	32	2	0.25	11	0.75	96	1.2903472104263305	1.3081875610351563	671.5940551757812	176108932694016.0	9.988051165726906	578692.4092803065
6	32	2	0.25	11	1.0	32	1.4338770805358887	1.4401740844726563	382.6702880859375	36307988897152.0	9.995553697816712	68623.60246883276
7	32	2	0.25	11	1.0	64	1.2872882096099854	1.3017198608398437	614.8167114257812	104111380889600.0	10.332132641998133	261505.50920392727
8	32	2	0.25	11	1.0	96	1.1797979640579224	1.210735791015625	861.1759643554688	980501967929344.0	10.47074283608582	578692.4092803065
9	32	2	0.5	11	0.5	32	1.647818482208252	1.652047412109375	195.7499542236328	48745324544.0	9.077689639831153	68623.60246883276
10	32	2	0.5	11	0.5	64	1.5146893096923828	1.5277306640625	336.144775390625	1820130672640.0	9.297259706812927	261505.50920392727
11	32	2	0.5	11	0.5	96	1.4025040432739257	1.4133209350585938	485.2920227050781	22479308587008.0	9.611979533125876	578692.4092803065
12	32	2	0.5	11	0.75	32	1.5350758778381348	1.542181494140625	298.9728698730469	865044561920.0	9.863876434586972	68623.60246883276
13	32	2	0.5	11	0.75	64	1.4129094245147704	1.4246499755859374	442.74383544921875	12976181084160.0	9.881342971592378	261505.50920392727
14	32	2	0.5	11	0.75	96	1.3050737073135377	1.3215660400390625	629.9307250976562	134067651608576.0	10.119465151960998	578692.4092803065
15	32	2	0.5	11	1.0	32	1.47305827167511	1.4756491577148438	363.6966857910156	2859079860224.0	9.950630262853968	68623.60246883276
16	32	2	0.5	11	1.0	64	1.318439925956726	1.3339312255859375	609.2085571289062	115965943611392.0	10.603846300404228	261505.50920392727
17	32	2	0.5	11	1.0	96	1.2133391843795776	1.2385993774414064	808.0307006835938	719195595603968.0	10.60977105466707	578692.4092803065
18	32	2	0.75	11	0.5	32	1.6890360149383545	1.6904630737304687	190.304931640625	42808285184.0	9.001476670994519	68623.60246883276
19	32	2	0.75	11	0.5	64	1.511415068511963	1.5221186401367188	343.4784240722656	2618246758400.0	9.737188930962834	261505.50920392727
20	32	2	0.75	11	0.5	96	1.4447740175628663	1.4600246337890626	469.136474609375	19693327876096.0	9.760213610843358	578692.4092803065
21	32	2	0.75	11	0.75	32	1.5980530443572998	1.6011066528320312	259.4551086425781	381384708096.0	9.962372227673947	68623.60246883276
22	32	2	0.75	11	0.75	64	1.448811724319458	1.4569664916992187	435.93243408203125	13709542031360.0	10.214797803758701	261505.50920392727
23	32	2	0.75	11	0.75	96	1.3547077745819092	1.364498583984375	614.957275390625	127773324607488.0	10.315384010226081	578692.4092803065
24	32	2	0.75	11	1.0	32	1.5850845792388917	1.5870555908203126	306.98614501953125	1347497525248.0	10.393088268598351	68623.60246883276
25	32	2	0.75	11	1.0	64	1.3754798691940309	1.3874182861328126	584.2242431640625	100112927817728.0	10.67411647473484	261505.50920392727
26	32	2	0.75	11	1.0	96	1.288863236808777	1.3065276611328125	774.1082153320312	615246767063040.0	10.824647939156478	578692.4092803065

Figure 4: Training Results of 27 models under `NiN` Design on CIFAR-10 with reported model parameters, losses, and various complexity measures.

Kendall Correlation				
	μ_{param}	$\mu_{spectral\text{-}orig}$	μ_{spec}	μ_{VC}
MNIST	0.481	0.493	0.350	0.322
CIFAR-10	0.692	0.704	0.368	0.556

Table 2: Kendall correlations between complexity measures and generalization gap obtained by training 27 models each on the MNIST and CIFAR-10 datasets under `NiN` design.