A Surprising Power Law Relationship Predicts Trends in the Test Accuracy for Very Deep Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

Given two or more Deep Neural Networks (DNNs) with similar architectures, and trained on the same dataset, but trained with different solvers, hyper-parameters, regularization, etc., can we predict which DNN will have the best test accuracy, without peeking at the test data ? Solving this question of generalization would have both theoretical impact and great practical importance. In this paper, we show how to use our new theory of Implicit (Heavy Tailed) Self-Regularization for modern Deep Neural Networks to answer this. We examine over 50 different, pre-trained DNNs ranging over 15 different architectures, trained on ImagetNet, with differing test accuracies. We show that, across each architecture (VGG16, VGG19, InceptionV3/V4, ...), the reported test accuracies for each DNN are well correlated with the weighted average of the layer power law exponents. Moreover, we prove that this average complexity can be expressed simply as the average log of the Frobenius norm of the layer weight matrices. Our approach requires no changes to the underlying DNN, and does not even require access to the ImageNet data. We present and review these empirical results, and compare and contrast with recent approaches to estimate test performance of DNNs using product norms.

1 Introduction

2 Theory of Heavy Tailed Self Regularization

Let us write the Energy Landscape (or optimization function) for a typical DNN with L layers, with activation functions $h_l(\cdot)$, and with weight matrices and b iases $\mathbf{W}_l$ and $\mathbf{b}_l$, as follows:

$$E_{DNN} = h_L(\mathbf{W}_L \times h_{L-1}(\mathbf{W}_{L-1} \times h_{L-2}(\cdots) + \mathbf{b}_{L-1}) + \mathbf{b}_L). \tag{1}$$

We imagine training this model on some labeled data $\{d_i, y_i\} \in \mathcal{D}$, using Backprop, by minimizing the loss $\mathcal{L}$ For simplicity, we do not indicate the structural details of the layers (e.g., Dense or not, Convolutions or not, Residual/Skip Connections, etc.). Each layer is defined by one or more weight matrices $\mathbf{W}_L$, or tensors.

In this study, we only need to consider Linear and 2D Convolutional (Conv2D) layers. For the Linear layers, each $\mathbf{W}_L$ is a single $(N \times M)$ (real valued) 2D matrix, where $N \geq M$. These include Dense or Fully Connected (FC) layers, as well as 1D Convolutional (Conv1D) layers, Attention matrices, etc. For the Conv2D layers, with a $c\times d$ kernel, $\mathbf{W}_L$ is a 4-index Tensor, of the form $(N\times M\times c\times d)$, consisting of $c \times d$ 2D feature maps of shape $(N \times M)$. So each Linear layer l gives $n = 1$ 2D matrix $\mathbf{W}_l$, and each Conv2D layer l gives $n = c \times d$ 2D matrices $\mathbf{W}_{l,i}$. A typical modern DNN may have anywhere between 5 and 500 2D $\mathbf{W}_{l,i}$ layer matrices.

Heavy Tailed Universality For any layer weight matrix $\mathbf{W}$, we construct the associated $M \times M$ (uncentered) correlation matrix

$$\mathbf{X} = \frac{1}{N}\mathbf{W}^T\mathbf{W}, \tag{2}$$

and form the eigenvalue spectrum of $\mathbf{X}$,

$$\mathbf{X}\mathbf{v}_i = \lambda_i\mathbf{v}_i$$

(where we have dropped the l, i indices).

We call the density of eigenvalues $\rho(\lambda)$ the Empirical Spectral Density (ESD).

Small Neural Networks [charlesG: complete]

The layer matrices in all large, modern DNNs, do not have a scale cut-off, and, instead display scale-invariance. For any modern DNN, the ESD of nearly every $\mathbf{W}$ layer matrix can be fit to a power law,

$$\rho(\lambda) \sim \lambda^{-\alpha}$$

which is valid within a bounded range of eigenvalues $\lambda \in [\lambda_{min}, \lambda_{max}]$.

The power law exponent α lies, $80 - 90\%$ of the time, in a Universal range between 2 and 4, $\alpha \in [2, 4]$. We have verified this empirical Universality on empirical result on over 10,000 layer matrices $\mathbf{W}$ spanning over 50 pre-trained DNNs. Of course, there are exceptions, and in any real DNN, α may range anywhere from ~ 1.5 to 10 or higher.

The power law exponent α is a complexity metric for a weight matrix; it describes how well that matrix encodes the complex correlations in the training data. So a natural complexity metric for a DNN is to take a weighted average of the power law exponents $\alpha_{l,i}$ for each layer weight matrix $\mathbf{W}_{l,i}$.

$$\hat{\alpha} := \frac{1}{n}\sum_{l,i} b_{l,i}\alpha_{l,i}$$

The smaller $\hat{\alpha}$, the better we expect the DNN to represent the training data. And, presumably, the better the DNN will generalize. The only question is, what are good weights $b_{l,i}$?

It turns out, we can derive the weighted average $\hat{\alpha}$ directly from the more familiar Product Norm.

[charlesG: THE REST OF THIS SECTION is to justify this complexity metric, using the product norm for DNNs. I have sketeched out some of the math,. It just needs to be presented clearly AND TO JUSTIFY computing the average log Norm (which is much faster, much simpler)]

[charles: Maybe should can state a theorem and prove it.]

THEOREM: *The data dependent VC-like complexity of a Deep Neural Network can be expressed a weighted average the of power law exponents describing the empirical spectral density of the layer weight matrices*

[charles:

PROOF:...]

Product Norm Measures of Complexity Recently it has been suggested that the complexity of a DNN, $\mathcal{C}$, can be defined by something akin to a data dependent VC complexity, the product of norms of the layer weight matrices

$$\mathcal{C} \sim \|\mathbf{W}_1\| \times \|\mathbf{W}_2\| \cdots \|\mathbf{W}_L\|$$

where $\|\mathbf{W}\|$ may be the Frobenius norm or even the L1-norm. (Here we can use either $\|\mathbf{W}\|$ or $\|\mathbf{W}\|^2$ for our complexity metric, which will make more sense below.

To that end, we consider a log complexity

$$\log\mathcal{C} \sim \log\Big[\|\mathbf{W}_1\| \times \|\mathbf{W}_2\| \cdots \|\mathbf{W}_L\|\Big]$$

$$\sim \left[\log \|\mathbf{W}_1\| + \log \|\mathbf{W}_2\| \cdots \log \|\mathbf{W}_L\| \right]$$

We define the average log norm of the weight matrices as

$$\langle \log \|\mathbf{W}\|_F \rangle = \frac{1}{L} \sum_{i=1}^{L} \log_{10} \|\mathbf{W}_i\|$$

which we explicitly define in terms of the base-10 log.

Relation to Heavy Tailed Universality We note the log matrix norm squared (and dropping the l, i subscripts) is just twice the log norm

$$\log \|\mathbf{W}\|_F^2 = 2 \log \|\mathbf{W}\|$$

which makes the following analysis much simpler.

the squared Frobenius norm is just the Trace of $\mathbf{W}^T\mathbf{W}$:

$$\|\mathbf{W}\|_F^2 = Tr[\mathbf{W}^T\mathbf{W}]$$

or N times the Trace of $\mathbf{X}$

$$= Tr[N\mathbf{X}] = N \sum_{i=1}^{M} \lambda_i$$

For very large matrices, with many eigenvalues, we can approximate the Trace as an integral over the empirical density

[charles: We may be missing the N term here:

$$\|\mathbf{W}_{l,i}\|_F^2 \sim \int_{x_{min}}^{x_{max}} \lambda \rho_{l,i}(\lambda) d\lambda$$

Technically, the power only describes the tail of the ESD, for the range $\lambda \in [\lambda_{min}, \lambda_{max}]$. But for most DNN layer matrices, this range covers most the ESD. So this should also be a pretty good approximation.

Of course, we need to normalize the ESD $\rho(\lambda)$. For now, we simply take

$$\rho(\lambda) := (\alpha - 1)\lambda^{-\alpha}$$

so that

$$\int_0^\infty \rho(\lambda) d\lambda = 1$$

]

[michael: something like this ? Trying to figure out how to get rid of the - sign on alpha

$$\int_0^\infty \rho(\lambda) d\lambda = N$$

but we really want

$$C\int_{0}^{\lambda_{max}}\rho(\lambda)d\lambda = N$$

$$\frac{C}{(1-\alpha)}\lambda_{max}^{1-\alpha} = N$$

What is C ? Let $C = (\alpha-1)K$ and solve for K

$$K\lambda_{max}^{1-\alpha} = N$$

$$K = N\lambda_{max}^{\alpha-1}$$

This means we need to evaluate

$$(\alpha-1)N\lambda_{max}^{\alpha-1}\int_{0}^{\lambda_{\max}}\lambda^{1-\alpha}d\lambda$$

which gives a crazier expression for the Norm that is missing the alpha dependence ?

$$Norm^{2} = \frac{(\alpha-1)}{2-\alpha}N\lambda_{max}^{\alpha-1}\lambda_{max}^{2-\alpha}$$

$$Norm^{2} = \frac{(\alpha-1)}{2-\alpha}N\lambda_{max}$$

]

Notice we ignore that we actually want to normalize only the bounded support $\int_{\lambda_{min}}^{\lambda^{max}}\circ d\lambda$. Of course, eventually we need to consider this since we have a finite $\lambda_{max} \ll \infty$.

For now let us just evaluate the integral

$$\|\mathbf{W}\|^{2} = (\alpha-1)\int_{\lambda_{min}}^{\lambda_{max}}\lambda\rho(\lambda)d\lambda$$

$$= (\alpha-1)\int_{\lambda_{min}}^{\lambda_{max}}\lambda^{\alpha-1}d\lambda$$

$$= \frac{\alpha-1}{2-\alpha}\left[\lambda^{2-\alpha}\right]_{\lambda_{min}}^{\lambda_{max}}$$

Since $\lambda_{min} \sim 0$, we have

$$\|\mathbf{W}\|_{F}^{2} \approx \frac{\alpha-1}{2-\alpha}\lambda_{max}^{2-\alpha}$$

Taking the log of both sides now gives an expression for the log Norm (squared)

$$\log\|\mathbf{W}\|_{F}^{2} \approx (2-\alpha)\log\lambda_{max} + log\ K_{\alpha}$$

where $K_{a} = \dfrac{\alpha-1}{2-\alpha}$ is a normalization term that does not depend on λ_{max}.

Power Law - Norm Relation [michael: Why is the sign wrong ?!]

Let us first write

$$\log \|\mathbf{W}\|_F^2 \approx (-\alpha)\log \lambda_{max} + 2\log \lambda_{max} + \log\ K_\alpha$$

Notice that if we divide the log Norm by the log max eigenvalue, we get the the *Stable Rank*, but with the numerator and denominator in log units. We define this ratio as the

Log-Units Stable Rank:

$$\mathcal{S}_R^{log} := \frac{\log \|\mathbf{W}\|_F}{\log \lambda_{max}}$$

Our simple derivation suggests that this ratio is linear in α:

$$\mathcal{S}_R^{log} \sim B\alpha + 2 + \cdots$$

Notice that when $\alpha < 2$, then $\lambda_{max}^{2-\alpha}$ is large, and the relation is dominated by α. But when $\alpha < 2$, then $\lambda_{max}^{2-\alpha}$ is small, and the Log-Units Stable Rank is $\mathcal{S}_R^{log} \sim 2$.

[michael: combine into 1 equation...and why is $B > 0$?]

$$\mathcal{S}_R^{log} \approx B\alpha \ \ \alpha < 2$$
$$\mathcal{S}_R^{log} \approx 2.x \ \ \alpha > 2$$

We can test this relation between the Power Law exponent and the Log Frobenius Norm numerically. We generate a large number of random Heavy Tailed random matrices $\mathbf{W}^{rand}(\mu, M)$, with different number of eigenvalues M (with aspect ratio $Q = 1$), and drawn from a Pareto distributions with exponents $\mu \in [0.5, 5]$

$$\Pr[W_{i,j}^{rand}] \sim \frac{1}{x^{1+\mu}}$$

We fit the ESD of each $\mathbf{W}^{rand}$ to a power law using the method of Clauset et. al. to obtain the exponent α. We then examinea the empirical relation between the Log-Units Stable Rank $\mathcal{S}_R^{log}$ and the ESD Power Law exponent α

$$\mathcal{S}_R^{log} \ \ vs. \ \ \alpha$$

[michael: or is this better:

$$\frac{\log \|\mathbf{W}\|_F}{\log \lambda_{max}} \ \ vs. \ \ (\alpha)$$

]

Figure **?** plots the data for the *Power Law-Norm* relation

[charlesG: Describe here.

The numerical results for the the Power Law-Norm relation shows several interesting features:

First, as α increases, the Log-units Stable Rank increases. Then it

- is very clear for $\alpha < 2$
- saturates for $\alpha > 2$ for large M
- extends beyond $\alpha > 2$ because of finite size effects

]

[charlesG: I am not sure why the slope is not negative...the eigenvalues are VERY LARGE, and $\log \lambda_{max}\ 3 - 5$ so this just is OFF. Are we missing something in the derivation, like the fact that the norm squared has to be positive ? HELP]

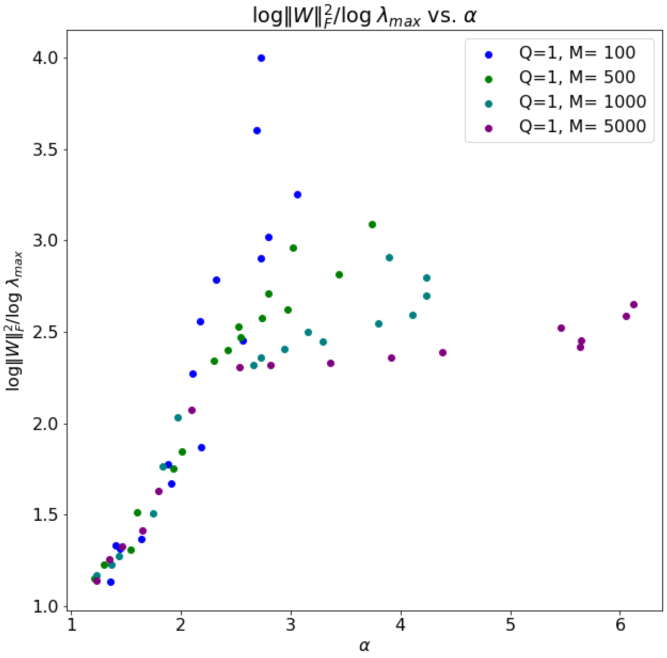


Figure 1: Numerical test of the Power Law - Norm Relation for random Heavy Tailed matrices

Defining the Layer Weight Matrices [charles: Here we describe how we exact the matrices. We have not done this yet for convolutional layers–thats *another* paper]

$$\text{Linear Layer:} \quad \log\|\mathbf{W}_l\|^2 \rightarrow b_l\alpha_l$$

For the Conv2D layers, we relate the 'Norm' of the 4-index Tensor $\mathbf{W}_l$ to the sum of the integrals of the $n = c \times d$ ESDs for each feature map, giving

$$\text{Conv2D Layer:} \quad \log\|\mathbf{W}_l\|^2 \rightarrow \sum_i b_{i,l}\alpha_{i,l}$$

So in the expression for the product norm for $\log\mathcal{C}$, let us replace each $\log\|\mathbf{W}_l\|$ for layer l with the sum of the power law exponents $\alpha_{l,i}$ for the n_l $\mathbf{W}_{l,i}$ layer matrices, and take the average over all N_α matrices. This lets us relate the product norm complexity metric to the weighted average of power law exponents

where

$$\hat{\alpha} := \frac{1}{N_\alpha}\sum_{i,l} b_{i,j}\alpha_{i,l}$$

We can now use $\hat{\alpha}$, or, equivalently, $2\langle\log\|\mathbf{W}\|\rangle$ to analyze numerous pre-trained DNNs...and the results are indeed surprising.

I HAVE PLOTS FOR BOTH ! Which should we display ?

Architecture	Model	Top1	Top5	L	N_α	$\hat{\alpha}$	$\hat{\alpha}*$
VGG11	VGG11 VGG11 BN						
VGG13	VGG13 VGG13 BN						
VGG16	VGG16 VGG16 BN						
VGG19	VGG19 VGG19 BN						

Table 1: VGG Architectures and DNN Models

Architecture	Model	Test Accuracy
ResNet (larger)		
ResNet (extended)		
ResNet (b)		

Table 2: ResNet Architectures and DNN Models

3 Empirical Results on PreTrained DNNs

VGG and VGG BN Models We start by looking at the VGG class of models, including VGG11, VGG13, VGG16, and VGG19, and their counterparts with Batch Normalization, VGG11_BN, VGG13_BN, VGG16_BN and VGG19_BN. Figure **??**

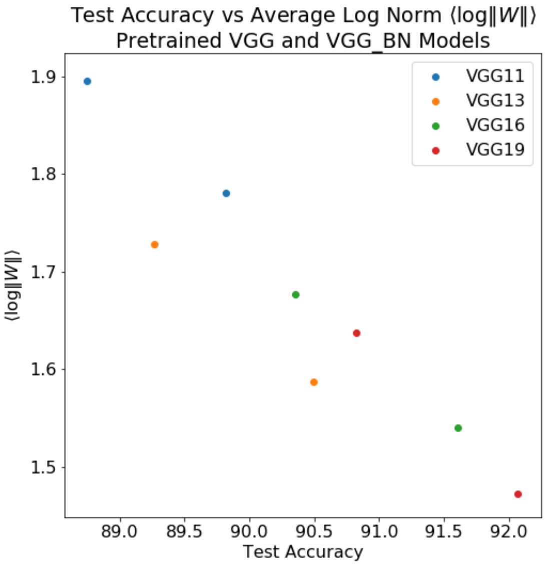


Figure 2: Pretrained VGG and VGG BN Architectures and DNNs. Test Accuracy and average log norm $\hat{\alpha}*$ for VGG11 vs VGG11_BN (blue), VGG13 vs VGG13_BN (orange), VGG16 vs VGG16_BN (green), and VGG19 vs VGG19_BN (red). Note that $\hat{\alpha}*$ does not include the last layer, connecting the model to the labels.

ResNet PyTorch Models

More PreTrained Models More Pretrained Models

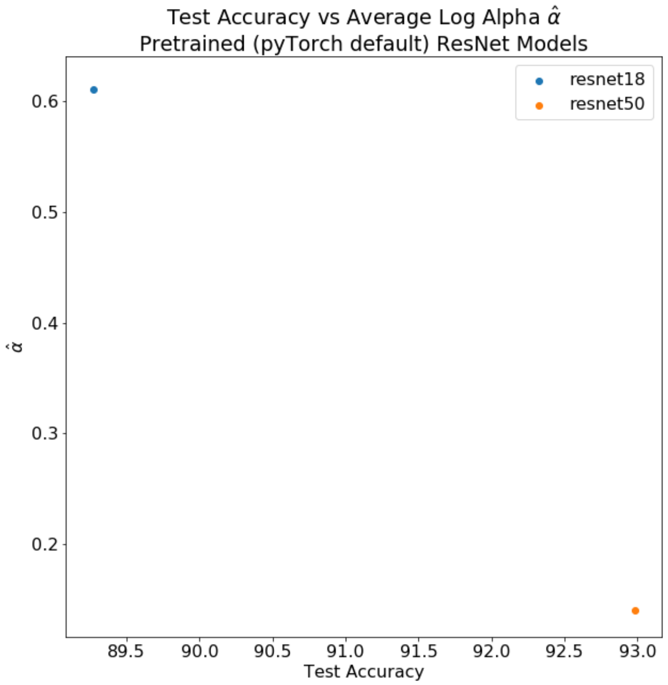
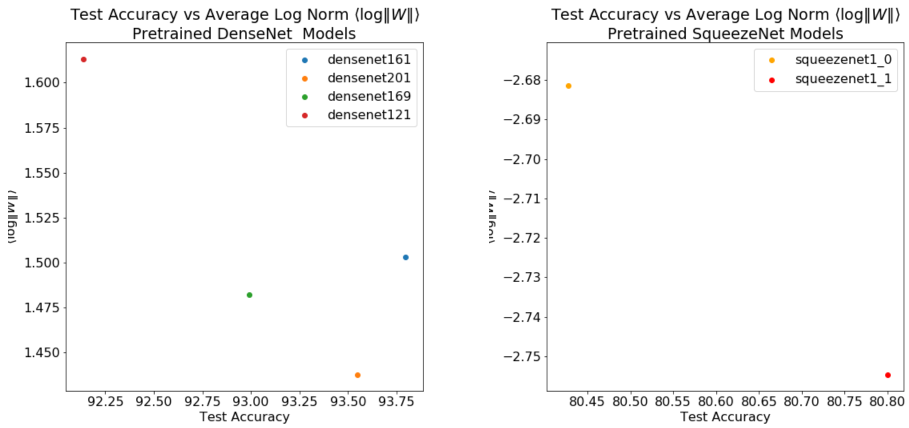


Figure 3: Pretrained ResNet models available in PyTorch

Architecture	Model	Test Accuracy
GoogLeNet		
ResNeXt		
SqueezeNet		

Table 3: Other Models



(a) DenseNet

(b) SqueezeNet

Figure 4: Densenet and SqueezeNet PyTorch Models

4 Discussion

We have presented a *Unsupervised* metric which predicts the trends in the test accuracies of a trained deep neural network–without peeking at the test data. This complexity metic $\hat{\alpha}$ is a weighted average of the power law exponents α for each layer weight matrix, where α is defined in our Theory of Heavy Tailed Implicit Regularization. We prove that this new complexity metric $\hat{\alpha}$ is equivalent to the the average log of the Frobenius norm of the layer weight matrices, $\langle \log \|\mathbf{W}\| \rangle$, which is much easier to compute.

We examine several commonly available, production quality, pretrained DNNs by plotting the average complexity metric $\langle \log \|\mathbf{W}\| \rangle$ vs the reported (Top1) test accuracies. This covers classes of DNN architectures including the VGG models, ResNet, DenseNet, etc. In nearly every class, the smaller average complexity, the better the test accuracy.

[charlesG: We can do BOTH log Norm and Weighted Alpha...which ?]

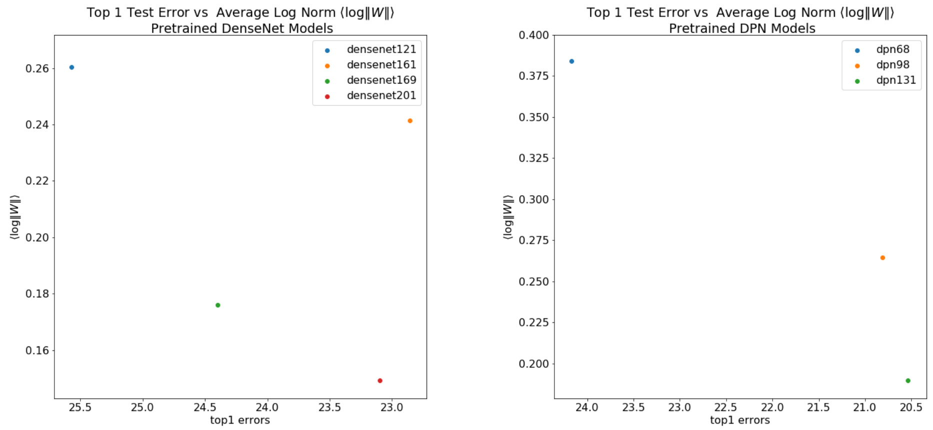
The method is consistent with both recent theoretical results by Hidary and Poggio, but the approach and the intent is a bit different. Unlike their result, our approach does not require modifying the loss function. Moreover, they seek a *worst case* complexity bound. We seek *average case* metrics that can be used in production to guide the development of better DNNs.

But most importantly, we can apply these complexity metrics *across related DNN architectures*. This is in stark contrast to the standard practice in machine learning. The equivalent notion would be to compare margins across SVMs applied to the same data, but with different Kernels. One loose interpretation is that a set of related of DNN models (i.e. VGG11, VGG13, ...) is analogous to a single, very complicated Kernel, and that the hierarchy of architectures is analogous to the hierarchy of hypothesis spaces in VC theory. [charlesG: more here ? like this ?]

We believe this result will have large applications in hyperparameter fine tuning DNNs. Because we do not need to peek at the test data, it may prevent information from leaking from the test set into the model, thereby helping to prevent overtraining and making fined tuned DNNs more robust.

This work also leads to a much harder theoretical question; is it possible to determine if a DNN is overtrained without peeking at the test data ?

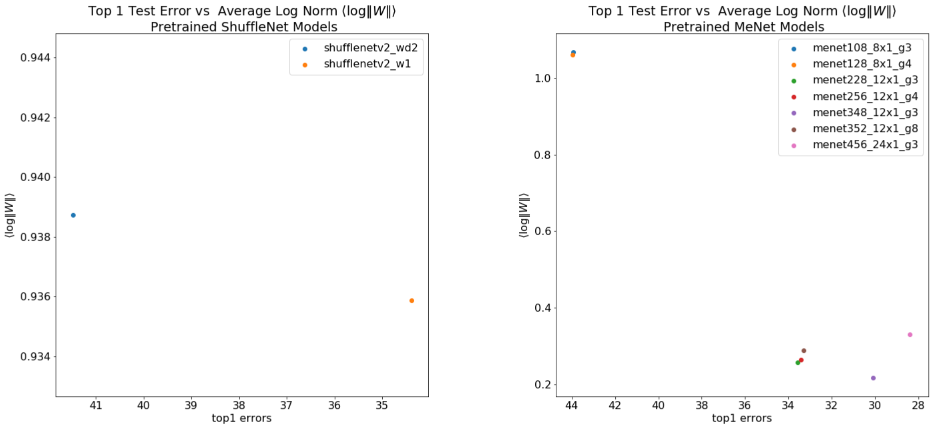
5 Appendix



(a) DenseNet

(b) DPN

Figure 5: DenseNet, DPN



(a) ShuffleNet

(b) MeNet

Figure 6: ShuffleNet, MeNet

Architecture	Model	Top 1 Error	
DenseNet	densenet121	25.57	
	densenet161	22.86	
	densenet169	24.4	
	densenet201	23.1	
DPN	dpn68	24.17	
	dpn98	20.81	
	dpn131	20.54	
MeNet	menet108_8x1_g3	43.92	
	menet128_8x1_g4	43.95	
	menet228_12x1_g3	33.57	
	menet256_12x1_g4	33.41	
	menet348_12x1_g3	30.1	
	menet352_12x1_g8	33.31	
	menet456_24x1_g3	28.4	
MobileNet	mobilenet_wd4	46.26	
	mobilenet_wd2	36.3	
	mobilenet_w3d4	33.54	
	mobilenet_w1	29.86	
MobileNetV2	mobilenetv2_wd4	49.72	
	mobilenetv2_wd2	36.54	
	mobilenetv2_w3d4	31.89	
	mobilenetv2_w1	29.31	
FDMobileNet	fdmobilenet_wd4	55.77	
	fdmobilenet_wd2	43.85	
	fdmobilenet_w1	34.7	
SE-ResNet	seresnet50	22.47	
	seresnet101	21.88	
	seresnet152	21.48	
SE-ResNeXt	seresnext50_32x4d	21.0	
	seresnext101_32x4d	19.96	
ShuffleNet	shufflenetv2_wd2	41.48	
	shufflenetv2_w1	34.39	

Table 4: Even more pretrained DNN models.

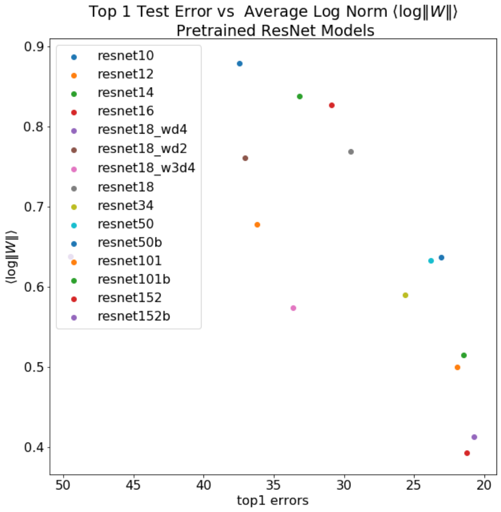
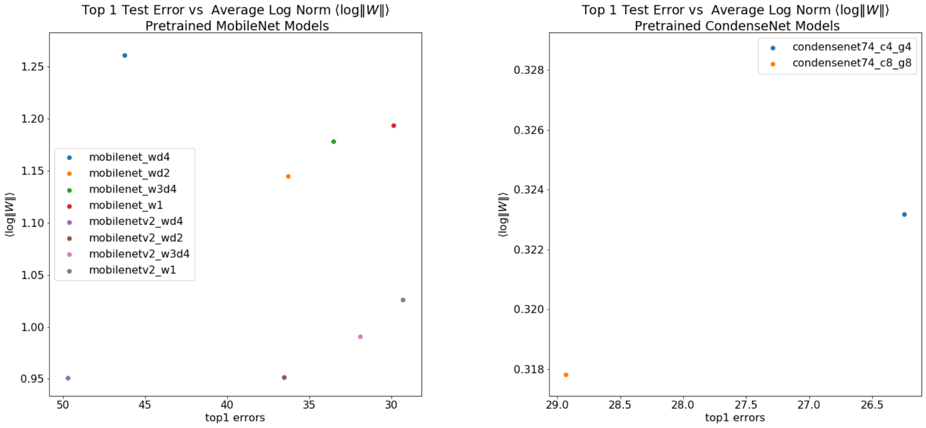


Figure 7: Pretrained ResNet models available in OSMR package

Architecture	Model	Top 1 Error	$\hat{\alpha}$
ResNet (small)	resnet10	37.46	
	resnet12	36.18	
	resnet14	33.17	
	resnet16	30.9	
	resnet18	29.52	
	resnet34	25.66	
	resnet50	23.79	
CondenseNet	condensenet74_c4_g4	26.25	
	condensenet74_c8_g8	28.93	

Table 5: Counter Examples



(a) MobileNet

(b) CondenseNet

Figure 8: CounterExamples