## A. Experiment Details

The details on model architectures, data set information and hyper-parameters used in the experiments for empirical estimation of the C-score can be found in Table 2. We implement our experiment in Tensorflow (Abadi et al., 2015). The holdout subroutine used in the empirical C-score estimation is based on the estimator proposed in Feldman & Zhang (2020), and listed in Algorithm 1. Most of the training jobs for C-score estimation are run on single NVidia® Tesla P100 GPUs. The ImageNet training jobs are run with 8 P100 GPUs using single-node multi-GPU data parallelization.

The experiments on learning speed are conducted with ResNet-18 on CIFAR-10, trained for 200 epochs while batch size is 32. For optimizer, we use the SGD with the initial learning rate 0.1, momentum 0.9 (with Nesterov momentum) and weight decay is 5e-4. The stage-wise constant learning rate scheduler decrease the learning rate at the 60th, 90th, and 120th epoch with a decay factor of 0.2.

**Algorithm 1** Estimation of $\hat{C}_{\hat{\mathcal{D}},n}$

**Require:** Data set $\hat{\mathcal{D}} = (X, Y)$ with $N$ examples
**Require:** $n$: number of instances used for training
**Require:** $k$: number of subset samples
**Ensure:** $\hat{C} \in \mathbb{R}^N$: $(\hat{C}_{\hat{\mathcal{D}},n}(x,y))_{(x,y)\in\hat{\mathcal{D}}}$
  Initialize binary mask matrix $M \leftarrow 0^{k\times N}$
  Initialize 0-1 loss matrix $L \leftarrow 0^{k\times N}$
  **for** $i \in (1, 2, \ldots, k)$ **do**
    Sample $n$ random indices $I$ from $\{1, \ldots, N\}$
    $M[i, I] \leftarrow 1$
    Train $\hat{f}$ from scratch with the subset $X[I], Y[I]$
    $L[i, :] \leftarrow \mathbf{1}[\hat{f}(X) \neq Y]$
  **end for**
  Initialize score estimation vector $\hat{C} \leftarrow 0^N$
  **for** $j \in (1, 2, \ldots, N)$ **do**
    $Q \leftarrow \neg M[:, j]$
    $\hat{C}[j] \leftarrow \text{sum}(\neg L[:, Q])/\text{sum}(Q)$
  **end for**

## B. Time and Space Complexity

The time complexity of the holdout procedure for empirical estimation of the C-score is $\mathcal{O}(S(kT + E))$. Here $S$ is the number of subset ratios, $k$ is number of holdout for each subset ratio, and $T$ is the average training time for a neural network. $E$ is the time for computing the score given the $k$-fold holdout training results, which involves elementwise computation on a matrix of size $k \times N$, and is negligible comparing to the time for training neural networks. The space complexity is the space for training a single neural network times the number of parallel training jobs. The space complexity for computing the scores is $\mathcal{O}(kN)$.

For kernel density estimation based scores, the most expensive part is forming the pairwise distance matrix (and the kernel matrix), which requires $\mathcal{O}(N^2)$ space and $\mathcal{O}(N^2 d)$ time, where $d$ is the dimension of the input or hidden representation spaces.

## C. More Visualizations of Images Ranked by C-score

Examples with high, middle and low C-scores from all the 10 classes of MNIST and CIFAR-10 are shown in Figure 11 and Figure 12, respectively. The results from the first 60 out of the 100 classes on CIFAR-100 is depicted in Figure 13. Figure 14 and Figure 15 show visualizations from ImageNet. Please see (URL anonymized) for more visualizations.

## D. C-Score Proxies based on Pairwise Distances

In the experiments of pairwise distance based C-score proxies, we use an RBF kernel $K(x, x') = \exp(-\|x-x'\|^2/h^2)$, where the bandwidth parameter $h$ is adaptively chosen as $1/2$ of the mean pairwise Euclidean distance across the data set. For the local outlier factor (LOF) algorithm (Breunig et al., 2000), we use the neighborhood size $k = 3$. See Figure 16 for the behavior of LOF across a wide range of neighborhood sizes.

### D.1. Pairwise Distance Estimation with Gradient Representations

Most modern neural networks are trained with first order gradient descent based algorithms and variants. In each iteration, the gradient of loss on a mini-batch of training examples evaluated at the current network weights is computed and used to update the current parameter. Let $\nabla_t(\cdot)$ be the function that maps an input-label training pair (the case of mini-batch size one) to the corresponding gradient evaluated at the network weights of the $t$-th iteration. Then this defines a gradient based representation on which we can compute density based ranking scores. The intuition is that in a gradient based learning algorithm, an example is consistent with others if they all compute similar gradients.

Comparing to the hidden representations defined the outputs of a neural network layer, the gradient based representations induce a more natural way of incorporating the label information. In the previous section, we reweight the neighbor examples belonging to a different class by 0 or -1. For gradient based representations, no ad hoc reweighting is needed as the gradient is computed on the loss that has already takes the label into account. Similar inputs with different labels automatically lead to dissimilar gradients. Moreover, this could seamlessly handle labels and losses with rich

*Table 2.* Details for the experiments used in the empirical estimation of the C-score.

| | MNIST | CIFAR-10 | CIFAR-100 | ImageNet |
|---|---|---|---|---|
| Architecture | MLP(512,256,10) | Inception† | Inception† | ResNet-50 (V2) |
| Optimizer | SGD | SGD | SGD | SGD |
| Momentum | 0.9 | 0.9 | 0.9 | 0.9 |
| Base Learning Rate | 0.1 | 0.4 | 0.4 | 0.1×7 |
| Learning Rate Scheduler | $\wedge(15\%)^\star$ | $\wedge(15\%)^\star$ | $\wedge(15\%)^\star$ | LinearRampupPiecewiseConstant$^{\star\star}$ |
| Batch Size | 256 | 512 | 512 | 128×7 |
| Epochs | 20 | 80 | 160 | 100 |
| Data Augmentation | ····· Random Padded Cropping$^\circledast$ + Random Left-Right Flipping ····· | | | |
| Image Size | 28×28 | 32×32 | 32×32 | 224×224 |
| Training Set Size | 60,000 | 50,000 | 50,000 | 1,281,167 |
| Number of Classes | 10 | 10 | 100 | 1000 |

† A simplified Inception model suitable for small image sizes, defined as follows:
Inception :: Conv(3×3, 96) → Stage1 → Stage2 → Stage3 → GlobalMaxPool → Linear.
Stage1 :: Block(32, 32) → Block(32, 48) → Conv(3×3, 160, Stride=2).
Stage2 :: Block(112, 48) → Block(96, 64) → Block(80, 80) → Block (48, 96) → Conv(3×3, 240, Stride=2).
Stage3 :: Block(176, 160) → Block(176, 160).
Block($C_1$, $C_2$) :: Concat(Conv(1×1, $C_1$), Conv(3×3,$C_2$)).
Conv :: Convolution → BatchNormalization → ReLU.

$\star$ $\wedge(15\%)$ learning rate scheduler linearly increase the learning rate from 0 to the *base learning rate* in the first $15\%$ training steps, and then from there linear decrease to 0 in the remaining training steps.

$\star\star$ LinearRampupPiecewiseConstant learning rate scheduler linearly increase the learning rate from 0 to the *base learning rate* in the first $15\%$ training steps. Then the learning rate remains piecewise constant with a $10\times$ decay at $30\%$, $60\%$ and $90\%$ of the training steps, respectively.

$\circledast$ Random Padded Cropping pad 4 pixels of zeros to all the four sides of MNIST, CIFAR-10, CIFAR-100 images and (randomly) crop back to the original image size. For ImageNet, a padding of 32 pixels is used for all four sides of the images.

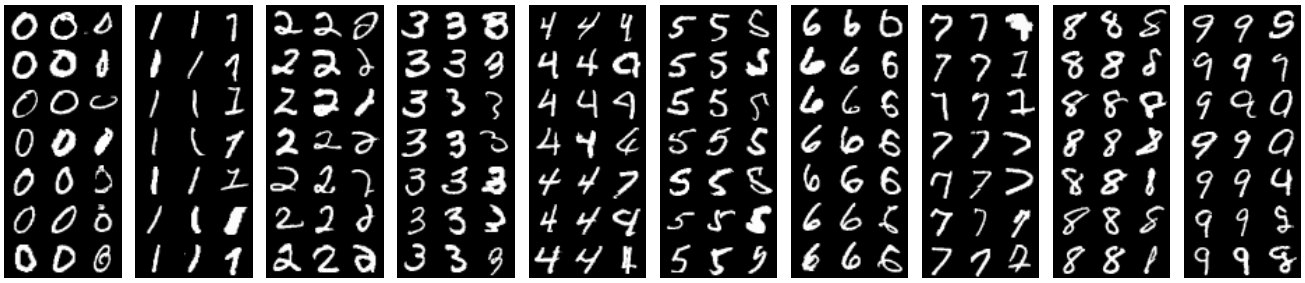


*Figure 11.* Examples from MNIST. Each block shows a single class; the left, middle, and right columns of a block depict instances with high, intermediate, and low C-scores, respectively.

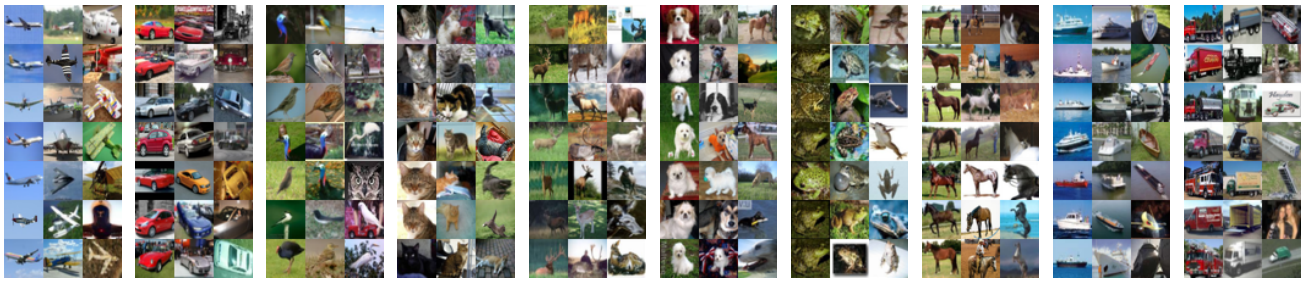


*Figure 12.* Examples from CIFAR-10. Each block shows a single class; the left, middle, and right columns of a block depict instances with high, intermediate, and low C-scores, respectively.

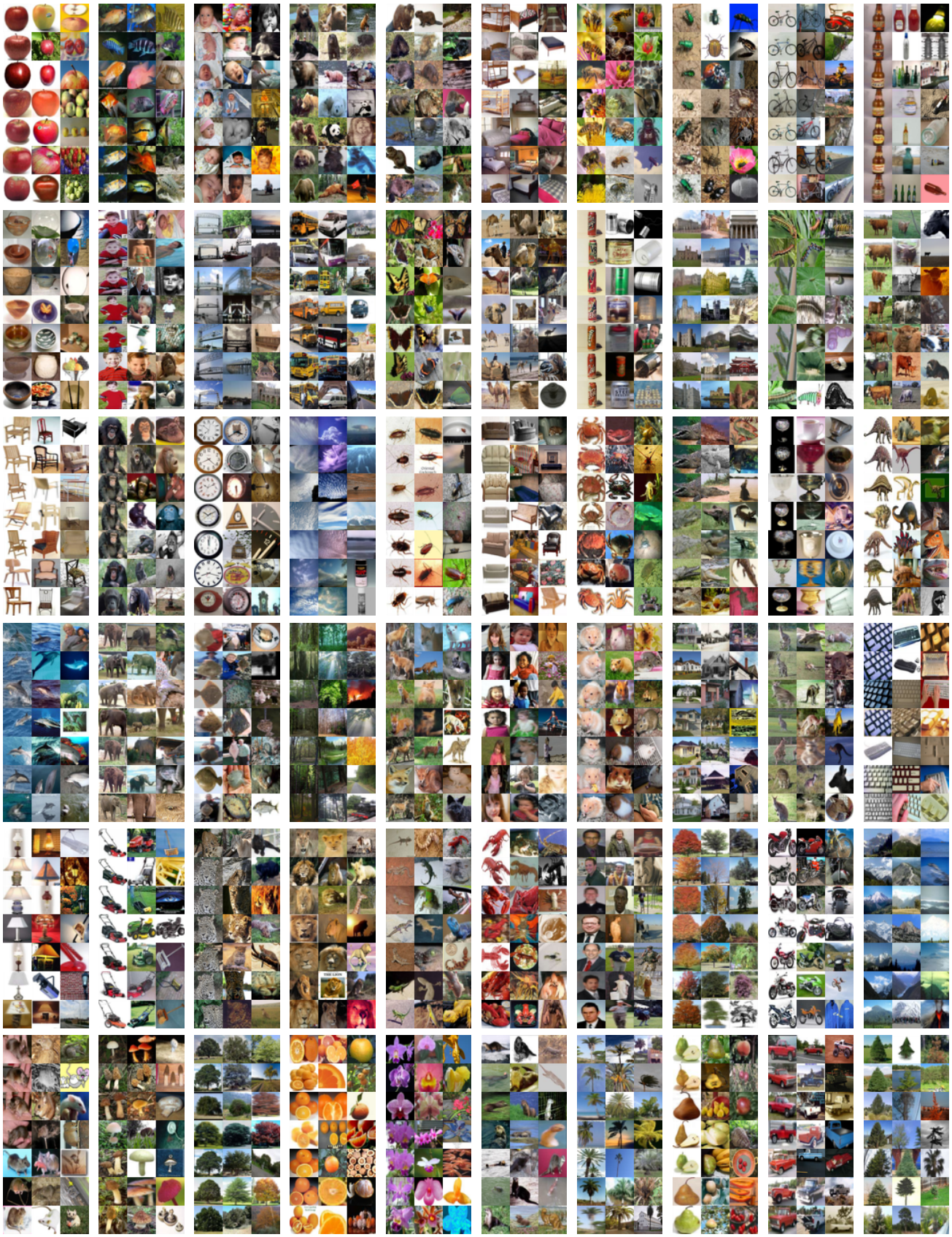


*Figure 13.* Examples from CIFAR-100. Each block shows a single class; the left, middle, and right columns of a block depict instances with high, intermediate, and low C-scores, respectively. The first 60 (out of the 100) classes are shown.

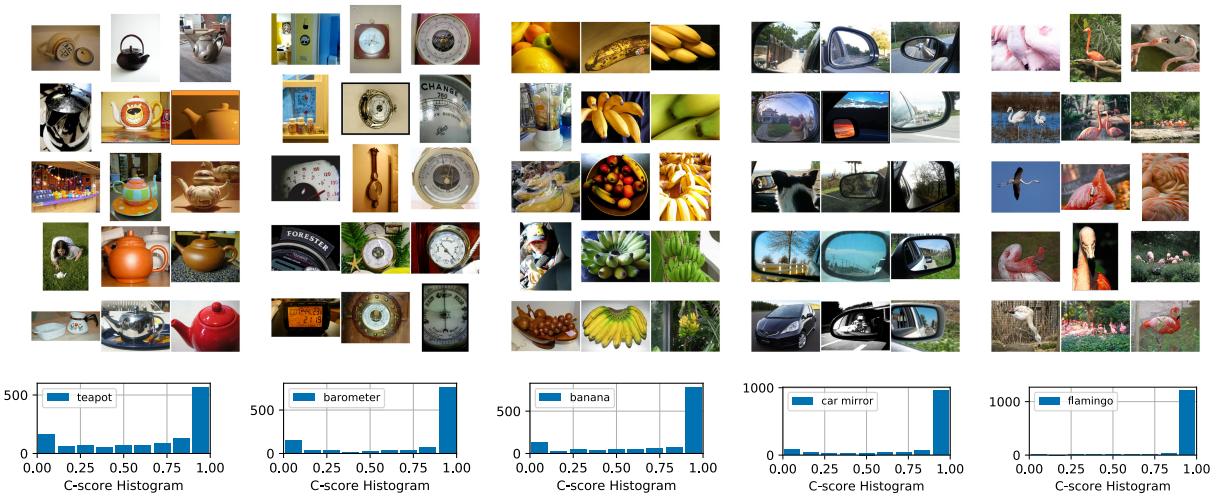



*Figure 14.* Example images from ImageNet. For each class, the three columns show sampled images from the (C-score ranked) top 99%, 35%, and 1% percentiles, respectively. The bottom pane shows the histograms of the C-scores in each of the 5 classes.

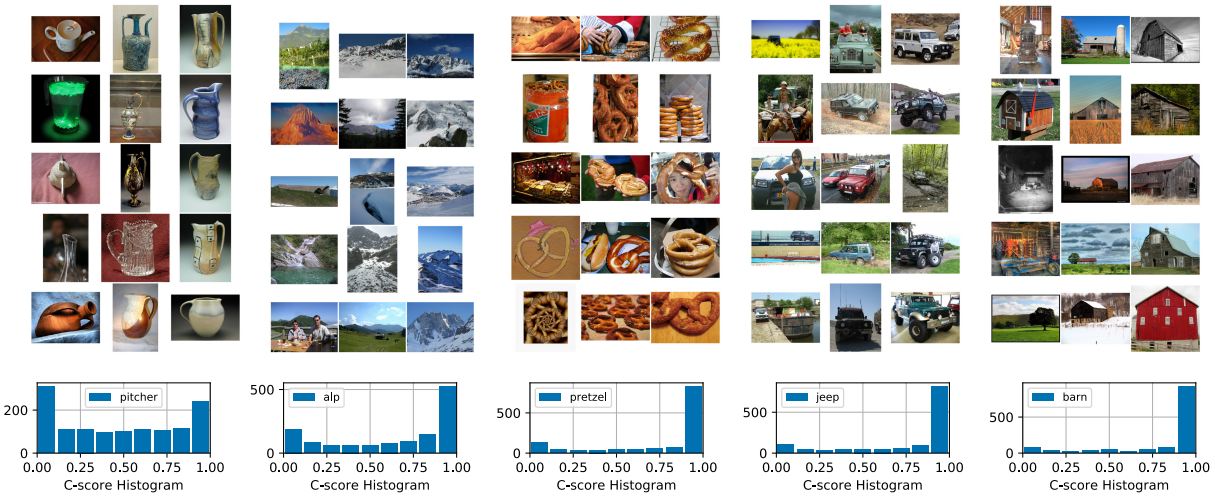
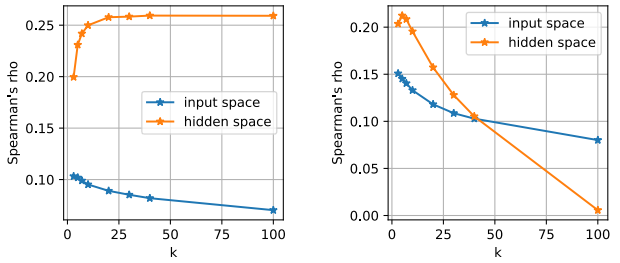



*Figure 15.* Example images from ImageNet. For each class, the three columns show sampled images from the (C-score ranked) top 99%, 35%, and 1% percentiles, respectively. The bottom pane shows the histograms of the C-scores in each of the 5 classes.




(a) CIFAR-10 (b) CIFAR-100

*Figure 16.* The Spearman's $\rho$ correlation between the C-score and the score based on LOF with different neighborhood sizes.

structures (e.g. image segmentation, machine translation) where an effective reweighting scheme is hard to find. The gradient based representation is closely related to recent developments on Neural Tagent Kernels (NTK) (Jacot et al., 2018). It is shown that when the network width goes to infinity, the neural network training dynamics can be effectively approximately via Taylor expansion at the initial network weights. In other words, the algorithm is effectively learning a *linear* model on the *nonlinear* representations defined by $\nabla_0(\cdot)$. This feature map induces the NTK, and connects deep learning to the literature of kernel machines.

Although NTK enjoys nice theoretical properties, it is challenging to perform density estimation on it. Even for the more practical case of *finite width* neural networks, the gradient representations are of extremely high dimensions as modern neural networks general have parameters ranging from millions to billions (e.g. Tan & Le, 2019; Radford et al., 2019). As a result, both computation and memory requirements are prohibitive if a naive density estimation is to be computed on the gradient representations. We leave as future work to explore efficient algorithms to practically compute this score.

## E. What Makes an Item Regular or Irregular?

The notion of regularity is primarily coming from the statistical consistency of the example with the rest of the population, but less from the intrinsic structure of the example's contents. To illustrate this, we refer back to Figure 4b in the main text, the distribution is uneven between high and low C-score values. As a result, the high C-score groups will have more examples than the low C-score groups. This agrees with the intuition that regularity arises from high probability masses.

To test whether an example with top-ranking C-score is still highly regular after the density of its neighborhood is reduced, we group the training examples according equal sized bins on the value range of their C-score values. We then subsample each group to contain an equal number ($\sim 400$) of examples. Then we run training on this new data set and observe the learning speed in each (subsampled) group. The result is shown in Figure 19, which is to be compared with the results without group-size-equalizing in Figure 10 in the main text. The following observations can be made:

1. The learning curves for many of the groups start to overlap with each other.

2. The lower ranked groups now learns faster. For example, the lowest ranked group goes above 30% accuracy near epoch 50. In the run without subsampling (Figure 10a in the main text), this groups is still below 20% accuracy at epoch 50. The model is now learning with a much smaller data set. Since the lower ranked examples are not highly consistent with the rest of the population, this means there are fewer "other examples" to compete with (i.e. those "other examples" will move the weights towards a direction that is less preferable for the lower ranked examples). As a result, the lower ranked groups can now learn faster.

3. On the other hand, the higher ranked groups now learn slower, which is clear from a direct comparison between Figure 10a in the main text and Figure 19 here. This is because for highly regular examples, reducing the data set size means removing consistent examples — that is, there are now less "supporters" as oppose to less "competitors" in the case of lower ranked groups. As a result, the learn speed is now slower.

4. Even though the learning curves are now overlapping, the highest ranked group and the lowest ranked group are still clearly separated. The potential reason is that while the lower ranked examples can be outliers in many different ways, the highest ranked examples are probably regular in a single (or very few) visual clusters (see the top ranked examples in Figure 12). As a result, the within group diversities of the highest ranked groups are still much smaller than the lowest ranked groups.

In summary, the regularity of an example arises from its consistency relation with the rest of the population. A regular example in isolation is no different to an outlier. Moreover, it is also not merely an intrinsic property of the data distribution, but is closely related to the model, loss function and learning algorithms. For example, while a picture with a red lake and a purple forest is likely be considered an outlier in the usual sense, for a model that only uses grayscale information it could be highly regular.

*Figure 17.* Examples from CIFAR-10 (left 5 blocks) and CIFAR-100 (right 5 blocks). Each block shows a single class; the left, middle, and right columns of a block depict instances with top, intermediate, and bottom ranking according to the relative local density score $\hat{C}^{\pm L}$ in the input space, respectively.

*Figure 18.* Examples from CIFAR-10 (left 5 blocks) and CIFAR-100 (right 5 blocks). Each block shows a single class; the left, middle, and right columns of a block depict instances with top, intermediate, and bottom ranking according to the relative local density score $\hat{C}_h^{\pm L}$ in the latent representation space of a trained network, respectively.

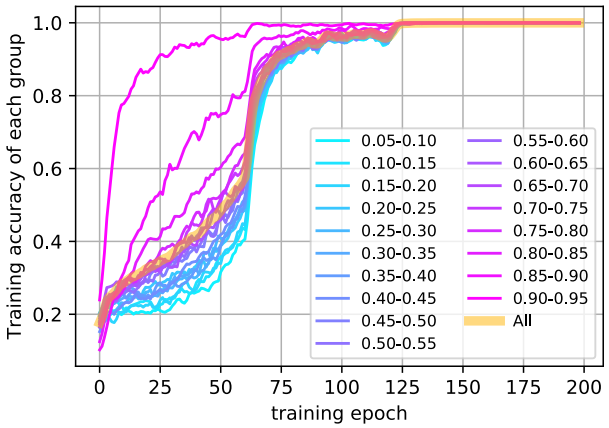



*Figure 19.* Learning speed of group of examples ranked by C-scores, with equal number (400) of examples in each group via subsampling.

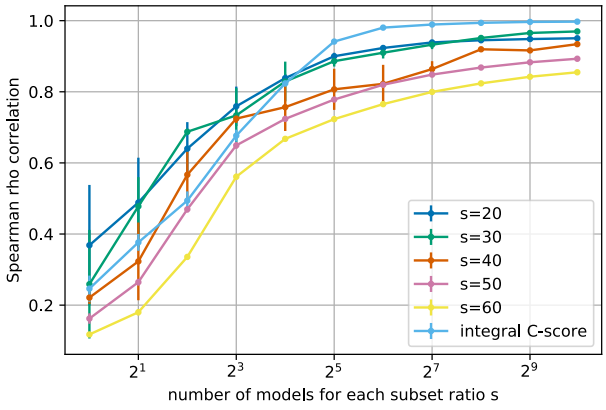



*Figure 20.* The correlation of C-scores estimated with varying numbers of models (the x-axis) and C-scores estimated with 1,000 independent models. The simulations are run with CIFAR-10, and the error bars show standard deviation from 10 runs.

## F. Sensitivity of C-scores to the Number of Models

We used 2,000 models per subset ratio to evaluate C-scores in our experiments to ensure that we get stable estimates. In this section, we study the sensitivity of C-scores with respect to the number of models and evaluate the possibility

to use fewer models in practice. Let $C_{0-2k}$ be the C-scores estimated with the full 2,000 models per subset ratio. We split the 2,000 models for each subset ratio into two halves, and obtain two independent estimates $C_{0-1k}$ and $C_{1k-2k}$. Then for $m \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1000\}$, we sample $m$ random models from the first 1,000 split, and estimate C-scores (denoted by $C_m$) based on those models. We compute the Spearman's $\rho$ correlation between each $C_m$ and $C_{1k-2k}$. The results are plotted in Figure 20. The random sampling of $m$ models is repeated 10 times for each $m$ and the error bars show the standard deviations. The figure shows that a good correlation is found for as few as $m = 64$ models. However, the integral C-score requires training models for various subset ratios (9 different subset ratios in our simulations), so the total number of models needed is roughly $64 \times 9$. If we want to obtain a reliable estimate of the C-score under a single fixed subset ratio, we find that we need $512$ models in order to get a $> .95$ correlation with $C_{1k-2k}$. So it appears that whether we are computing the integral C-score or the C-score for a particular subset ratio, we need to train on the order of 500-600 models.

In the analysis above, we have used $C_{1k-2k}$ as the reference scores to compute correlation to ensure no overlapping between the models used to compute different estimates. Note $C_{1k-2k}$ itself is well correlated with the the full estimate from 2,000 models, as demonstrated by the following correlations: $\rho(C_{0-1k}, C_{1k-2k}) = 0.9996$, $\rho(C_{0-1k}, C_{0-2k}) = 0.9999$, and $\rho(C_{1k-2k}, C_{0-2k}) = 0.9999$.