DATASET META-LEARNING FROM KERNEL RIDGE-REGRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

One of the most fundamental aspects of any machine learning algorithm is the training data used by the algorithm. We introduce the novel concept of ϵ-approximation of datasets, obtaining datasets which are much smaller than or are significant corruptions of the original training data while maintaining similar model performance. We introduce a meta-learning algorithm called Kernel Inducing Points (KIP) for obtaining such remarkable datasets, inspired by the recent developments in the correspondence between infinitely-wide neural networks and kernel ridge-regression (KRR). For KRR tasks, we demonstrate that KIP can compress datasets by one or two orders of magnitude, significantly improving previous dataset distillation and subset selection methods while obtaining state of the art results for MNIST and CIFAR-10 classification. Furthermore, our KIP-learned datasets are transferable to the training of finite-width neural networks even beyond the lazy-training regime, which leads to state of the art results for neural network dataset distillation with potential applications to privacy-preservation.

1 INTRODUCTION

Datasets are a pivotal component in any machine learning task. Typically, a machine learning problem regards a dataset as given and uses it to train a model according to some specific objective. In this work, we depart from the traditional paradigm by instead optimizing a dataset with respect to a learning objective, from which the resulting dataset can be used in a range of downstream learning tasks.

Our work is directly motivated by several challenges in existing learning methods. Kernel methods or instance-based learning (Vinyals et al., 2016; Snell et al., 2017; Kaya & Bilge, 2019) in general require a support dataset to be deployed at inference time. Achieving good prediction accuracy typically requires having a large support set, which inevitably increases both memory footprint and latency at inference time—*the scalability issue*. It can also raise *privacy concerns* when deploying a support set of original examples, e.g., distributing raw images to user devices. Additional challenges to scalability include, for instance, the desire for rapid hyper-parameter search (Shleifer & Prokop, 2019) and minimizing the resources consumed when replaying data for continual learning (Borsos et al., 2020). A valuable contribution to all these problems would be to find surrogate datasets that can mitigate the challenges which occur for naturally occurring datasets without a significant sacrifice in performance.

This suggests the following

Question: *What is the space of datasets, possibly with constraints in regards to size or signal preserved, whose trained models are all (approximately) equivalent to some specific model?*

In attempting to answer this question, in the setting of supervised learning on image data, we discover a rich variety of datasets, diverse in size and human interpretability while also robust to model architectures, which yield high performance or state of the art (SOTA) results when used as training data. We obtain such datasets through the introduction of a novel meta-learning algorithm called Kernel Inducing Points (KIP). Figure 1 shows some example images from our learned datasets.

We explore KIP in the context of compressing and corrupting datasets, validating its effectiveness in the setting of kernel-ridge regression (KRR) and neural network training on benchmark datasets MNIST and CIFAR-10. Our contributions can be summarized as follows:

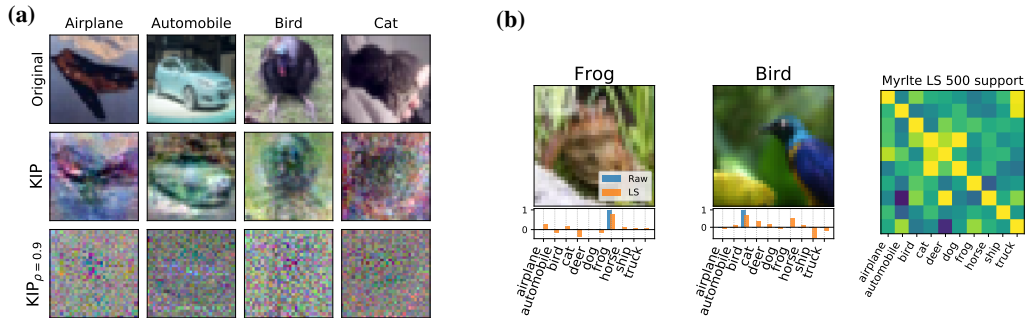

Figure 1: **(a)** Learned samples of CIFAR-10 using `KIP` and its variant `KIP`$_\rho$. With 100 images used in KRR, each set achieves 43.4% and 40.7% CIFAR-10 test accuracy respectively. These images transfer to 1) different kernel types and to 2) finite width neural networks well beyond the guarantees given by infinite-width correspondence. **(b)** Example of labels obtained by label solving (`LS`) (left two) and the covariance matrix between original labels and learned labels(right).

1.1 Summary of contributions

- We formulate a novel concept of ϵ-approximation of a dataset. This provides a theoretical framework for understanding dataset distillation and compression.
- We introduce Kernel Inducing Points (`KIP`), a meta-learning algorithm for obtaining ϵ-approximation of datasets. We establish convergence in the case of a linear kernel in Theorem 1. We also introduce a variant called Label Solve (`LS`), which gives a closed-form solution for obtaining distilled datasets differing only via labels.
- We explore following aspects of ϵ-approximation of datasets:
  1. **Compression (Distillation) for KRR:** For kernel ridge regression (KRR), we improve sample efficiency by over one or two orders of magnitude, e.g. using 10 images to outperform over 1000 images (Table 2 vs Table A3). We obtain SOTA results for MNIST and CIFAR-10 classification while using few enough images (10k) to allow for in-memory inference (Tables A4, A5)
  2. **Compression (Distillation) for NN:** We obtain state of the art dataset distillation results for the training of neural networks, often times even with only a single hidden layer fully-connected network (Tables 1 and 2).
  3. **Privacy:** We obtain datasets with a strong trade-off between corruption and test accuracy, which suggests applications to privacy-preserving dataset creation. In particular, we produce images with 90% of their pixels corrupted with minimal degradation in performance as measured by test accuracy in the appropriate regimes (Figure 3 and Table A6).

2 Related Work

Explicit dataset distillation: Maclaurin et al. (2015); Lorraine et al. (2020) approach dataset distillation through learning the input images from large-scale gradient-based meta-learning of hyperparameters. They did not evaluate the utility of learned images. Properties of distilled input data was first analyzed in Wang et al. (2018). One of main challenges of the method is that optimization is with respect to a specific initialization of a network, or else requires sampling many initializations which is costly. In contrast, our method naturally incorporates meta-learning over ensembles of networks through the infinite-width limit correspondence (Lee et al., 2018; Jacot et al., 2018). The works Sucholutsky & Schonlau (2019); Bohdal et al. (2020) build upon Wang et al. (2018) by distilling labels. More recently, Zhao et al. (2020) proposes condensing training set by gradient matching condition and shows improvement over Wang et al. (2018).

Coresets: Another approach for compressing datasets is via subset selection (i.e. coreset). While there are simple baselines based on random sampling, greedy sampling and clustering mechanism, one notable work is Borsos et al. (2020), utilizing KRR for dataset subselection.

Inducing points/variables: Our approach has as antecedant the inducing point method for Gaussian Processes (Snelson & Ghahramani, 2006; Titsias, 2009). However, whereas the latter requires a probabilistic framework that optimizes for marginal likelihood, in our method we only need to consider minimizing a mean-square error.

Differentiable Convex Optimization: Recent works have considered the meta-learning setup of differentiating through a convex solver in the setting of few-shot learning Bertinetto et al. (2018); Lee et al. (2019b). These prior works optimize for the weights of a backbone network, whereas we optimize directly on the input dataset.

Low-rank kernel approximations: Our `KIP` algorithm for compressing datasets produces low-rank approximations to the kernel matrices involved in KRR. However, unlike other common low-rank approximation methods (Williams & Seeger, 2001; Drineas & Mahoney, 2005), we obtain a low-rank approximation not only of the support-support kernel matrix, but also of the target-support kernel matrix.

Neural network kernels: Our work is motivated by the exact correspondence between infinitely-wide neural networks and kernel methods (Neal, 1994; Lee et al., 2018; Matthews et al., 2018; Jacot et al., 2018; Novak et al., 2019; Garriga-Alonso et al., 2019; Arora et al., 2019a). These correspondences allow us to view both Bayesian inference and gradient descent training of wide neural networks with squared loss as yielding a Gaussian process or kernel ridge regression with neural kernels. Neural network kernels for various architectural components can be easily utilized by the Neural Tangents Novak et al. (2020) library.

Influence functions: Understanding how neural networks depend on individual input was previously studied, e.g., Koh & Liang (2017); Chen et al. (2020). However, replacing a neural network with KRR using the associated neural kernel provides a much more tractable setting: the KRR regressor is a deterministic predictor and has a simple closed-form solution that can be explicitly differentiated using automatic differentiation.

3 SETUP

In this section we define some key concepts in our methods.

Definition 1. A *dataset* in $\mathbb{R}^d$ is a set of N vectors of $\mathbb{R}^d$ for some $N \geq 1$. We refer to each such vector as a *datapoint*. A dataset is *labeled* if each datapoint is paired with a label vector in $\mathbb{R}^C$, for some fixed C. A datapoint along with its corresponding label is a *labeled datapoint*.

We henceforth assume all datasets are labeled. Given a learning algorithm A (e.g. gradient descent on the loss function of a neural network), let A_D denote the resulting model obtained when training A on D. We regard A_D as a mapping from datapoints to prediction labels.

Definition 2. Let D be a dataset drawn from a distribution $\mathcal{P}$ on $\mathbb{R}^d \times \mathbb{R}^C$ and let $\tilde{D}$ be an arbitrary dataset in $\mathbb{R}^d$ with same label space $\mathbb{R}^C$. Let $\ell : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ be a loss function.

1. We say $\tilde{D}$ is a *weak ϵ-approximation* of $(D, \mathcal{P})$ with respect to learning algorithms $(A, \tilde{A})$ and loss function ℓ if
$$\left| \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big(\ell(\tilde{A}_{\tilde{D}}(x), y)\Big) - \mathbb{E}_{x\sim\mathcal{P}}\Big(\ell(A_D(x), y)\Big) \right| \leq \epsilon. \tag{1}$$

2. We say $\tilde{D}$ is a *strong ϵ-approximation* of $(D, \mathcal{P})$ with respect to learning algorithms $(A, \tilde{A})$ and loss function ℓ if
$$\mathbb{E}_{(x,y)\sim\mathcal{P}}\Big|\ell(\tilde{A}_{\tilde{D}}(x), y) - \ell(A_D(x), y)\Big| \leq \epsilon. \tag{2}$$

For brevity, if $\tilde{A} = A$, we will also speak of ϵ-approximation with respect to A. We will also call such a $\tilde{D}$ as above an ϵ-approximate dataset and omit reference to ℓ, A, $\tilde{A}$, and $\mathcal{P}$ when their values are understood.

We provide some justification for this definition in the Appendix. In this paper, we will measure ϵ-approximation with respect to 0-1 loss for multiway classification (i.e. accuracy). We focus on weak

ϵ-approximation, since in most of our experiments, we consider models in the low-data regime with large classification error rates, in which case, sample-wise agreement of two models is not of central importance. On the other hand, observe that if two models have population classification error rates less than $\epsilon/2$, then (2) is automatically satisfied, in which case, the notions of weak-approximation and strong-approximation converge.

The easiest way to obtain a strong ϵ-approximating dataset $\tilde{D}$ is for a model trained on $\tilde{D}$ to be (approximately) equal to a model trained on D. We list several examples:

Example 1: Support Vector Machines. Given a dataset D of size N, train an SVM on D and obtain M support vectors. Those M support vectors yield a dataset $\tilde{D}$ that is a 0-approximation to D.

Example 2: Regression. Any two datasets D and D' of $\mathbb{R}^d$ that determine the same linear (ridge) regressor are 0-equivalent. In particular, for any ridge parameter $\lambda > 0$, owing to the convexity of ridge regression, we can always find a 0-approximate dataset of any size.

We are interested in learning algorithms A given by KRR and neural networks. These can be investigated in unison via neural tangent kernels. Furthermore, we study two settings for the usage of ϵ-approximate datasets, though there are bound to be others:

1. (Sample efficiency / compression) Fix ϵ. What is the minimum size of $\tilde{D}$ needed in order for $\tilde{D}$ to be an ϵ-approximate dataset? (Thus if learning on D achieves (nearly) zero test error, we want the minimum number of samples needed to get ϵ test error.)
2. (Privacy guarantee) Can an ϵ-approximate dataset be found such that the distribution from which it is drawn and the distribution from which the original training dataset is drawn satisfy a given upper bound in mutual information?

Motivated by these questions, we introduce the following definitions:

Definition 3. Fix $(D, \mathcal{P})$ and a positive integer N. Let D_N^* be a subset of D of size at most N that is minimal with respect to population loss, i.e.

$$D_N^* \in \underset{\tilde{D} \subset D,\, |D| \leq N}{\arg\min} \quad \mathbb{E}_{x \sim \mathcal{P}} \ell_{A,\tilde{D}}(x).$$

Let $\tilde{D}$ be a weak ϵ-approximation to D_N^* such that $|\tilde{D}| < N$. We call $|D_N^*|/|\tilde{D}|$ the *compression ratio*.

Definition 4. Let Γ be an algorithm that takes a dataset D in $\mathbb{R}^d$ and returns a (random) collection of datasets in $\mathbb{R}^d$. For $0 \leq \rho \leq 1$, we say that Γ is *ρ-corrupted* if for any input dataset D, every datapoint[1] drawn from the datasets of $\Gamma(D)$ has at least ρ fraction of its coordinates independent of D.

In other words, datasets produced by Γ can only contain ρ fraction of the information contained of the datasets sampled from $\mathcal{P}$. A 0-corrupted algorithm can use all of the input dataset; a 1-corrupted algorithm does not depend on the input dataset.

Corrupting information is a naturally a way of enhancing privacy, as it makes it more difficult for an attacker to obtain useful information about the data used to train a model. Corruption of the inputs to neural network or of its gradient updates can be shown to provide differentially private guarantees Abadi et al. (2016).

4 Kernel Inducing Points

Given D and $\mathcal{P}$, we want to find a small dataset $\tilde{D}$ that is an ϵ-approximation to D (or perhaps some large subset of D) with respect to an algorithm A. Making the approximation

$$\mathbb{E}_{(x,y)\in\mathcal{P}}\, \ell(\tilde{A}_{\tilde{D}}(x), y) \approx \mathbb{E}_{(x,y)\in D}\, \ell(\tilde{A}_{\tilde{D}}(x), y), \tag{3}$$

[1] We ignore labels in our notion of ρ-corrupted since typically the label space has much smaller dimension than that of the datapoints.

Algorithm 1: Kernel Inducing Point (KIP)

Require: A target labeled dataset (X_t, y_t) along with a kernel or family of kernels.

1: Initialize a labeled support set (X_s, y_s).
2: **while** not converged **do**
3: Sample a random kernel. Sample a random batch $(\bar{X}_s, \bar{y}_s)$ from the support set. Sample a random batch $(\bar{X}_t, \bar{y}_t)$ from the target dataset.
4: Compute the λ-ridge regression loss given by (4) using the sampled kernel and the sampled support and target data.
5: Backpropagate through $\bar{X}_s$ (and optionally $\bar{y}_s$ and any hyper-parameters of the kernel) and update the support set (X_s, y_s).
6: **end while**
7: **return** Learned support set (X_s, y_s)

this suggests we should look to optimize the right-hand side of (3) with respect to $\tilde{D}$, using D as a validation set.

For general algorithms, the outer optimization for $\tilde{D}$ is computationally expensive and involves second-order derivatives, since one has to optimize over the inner loop encoded by the learning algorithm A. We are thus led to consider the class of algorithms drawn from kernel ridge-regression (KRR). The reason for this are twofold. First, KRR performs convex-optimization resulting in a closed-form solution. Thus, when optimizing for the training parameters of KRR (in particular, the support data), we only have to consider first-order optimization (the inner loop has been replaced by the closed-form solution). Second, under the correspondence between infinitely-wide neural networks and KRR, the use of neural kernels in KRR will yield meaningful results when training finite-width networks. (This will be validated in our experiments.)

This leads to our first-order meta-learning algorithm KIP (Kernel Inducing Points), which uses kernel-ridge regression to learn ϵ-approximate datasets. It can be regarded as an adaption of the inducing point method for GPs (Snelson & Ghahramani, 2006) to the case of KRR. Moreover, since kernel-ridge regression for neural kernels approximates the training of the corresponding wide neural network (Jacot et al., 2018; Lee et al., 2019a; Arora et al., 2019a; Lee et al., 2020), we expect $\tilde{D}$ to be an ϵ-approximation to D for a broad class of neural networks trainings as well.

Given a kernel K, the λ-KRR loss function evaluated on (X_t, y_t) is given by

$$L(X_s, y_s) = \frac{1}{2}\|y_t - K_{X_t X_s}(K_{X_s X_s} + \lambda I)^{-1} y_s\|^2. \tag{4}$$

where if U and V are sets, K_{UV} is the matrix of kernel elements $(K(u, v))_{u \in U, v \in V}$. (If $\lambda = 0$, we always interpret the inverse operation as yielding the pseudoinverse).

The KIP algorithm consists of optimizing (4) with respect to the support set (either just the X_s or along with the labels y_s), see algorithm 1. We found it very helpful to use families of kernels (Step 3) because then KIP produces datasets that are ϵ-approximations for a variety of kernels instead of a single one. We will see that this leads to a corresponding robustness for the learned datasets when used for neural network training. We remark on best experimental practices for sampling methods and initializations for KIP in the Appendix. Theoretical analysis for the convergence properties of KIP for the case of a linear kernel is provided by Theorem 1.

KIP variations: i) We can also randomly augment the sampled target batches in KIP . This effectively enhances the target dataset (X_t, y_t), and we obtain SOTA results in this way, with no extra computational cost with respect to the support size. ii) We also can choose a corruption fraction $0 \leq \rho < 1$ and do the following. Initialize a random ρ-percent of the coordinates of each support datapoint via some corruption scheme (zero out all such pixels or initialize with noise). Next, do not update such corrupted coordinates during the KIP training algorithm (i.e. we only perform gradient updates on the complementary set of coordinates). Call this resulting algorithm $\texttt{KIP}_\rho$. In this way, $\texttt{KIP}_\rho$ is ρ-corrupted according to Definition 4 and we use it to obtain our highly corrupted datasets.

Label solving: In addition to KIP , where we learn the support dataset via gradient descent, we propose another inducing point method, Label Solve (LS), in which we directly find the minimum of (4) with respect to the support labels. This is simple because the loss function is quadratic in y_s.

We refer to the resulting labels

$$y_s^* = K_{X_s X_s}(K_{X_s X_t} K_{X_t X_s} + \lambda I)^{-1} K_{X_s X_t}\, y_t\,. \tag{5}$$

as *solved labels*, where the matrix multiplying y_t in (5) is the λ-regularized pseudo-inverse of $K_{X_t X_s}(K_{X_s X_s})^{-1}$.

5 EXPERIMENTS

We perform three sets of experiments to validate the efficacy of `KIP` and `LS` for dataset learning. The first set of experiments investigates optimizing `KIP` and `LS` for compressing datasets and achieving state of the art performance for individual kernels. The second set of experiments explores transferability of such learned datasets across different kernels. The third set of experiments investigate the transferability of `KIP` -learned datasets to training neural networks. Further experimental details can be found in the Appendix.

For the experiments, we mainly focus on MNIST (LeCun et al., 2010) and CIFAR-10 datasets for comparison to previous methods. For `LS` , we also use Fashion-MNIST. All our kernel-based experiments use the Neural Tangents library (Novak et al., 2020), built on top of JAX (Bradbury et al., 2018). In what follows, we use FCm and Convm to denote a depth m fully-connected or fully-convolutional network with ReLU activation. (Whether we mean a finite-width neural network or else the corresponding neural tangent kernel will be understood from the context.) RBF denotes the radial-basis function kernel. We use ReLU activation version of Myrtle-N architecture following Shankar et al. (2020) where simple combination of N convolutional layers with $(2, 2)$ average pooling layers inter-weaved to reduce internal patch-size.

5.1 SINGLE KERNEL RESULTS

We apply `KIP` to learn support datasets of various sizes for MNIST and CIFAR-10. The objective is to distill the entire training dataset down to datasets of various fixed, smaller sizes. We obtain state of the art results for kernel performance on MNIST and CIFAR-10, for the RBF and FC1 kernels, both in terms of accuracy and number of images required, see Tables 1 and 2. In particular, our method produces datasets such that RBF and FC1 kernels fit to them rival the performance of deep convolutional neural networks on MNIST (exceeding 99.2%). By comparing Tables 2 and A3, we see that, e.g. 10 `KIP` images for RBF and FC1 outperform a subset of 1,280 randomly sampled images, resulting in a compression factor of more than 100.

Label solve compression ratio: We apply `LS` to various subsets of MNIST, Fashion-MNIST, and CIFAR-10 using kernels FC1, Myrtle-5, Myrtle-10, shown in Figure 2. Different plots show how many targets are used in `LS` . The more targets are used, the better the performance. We see that when all possible targets are used, we get an optimal compression ratio of roughly one order of magnitude at intermediate support sizes.

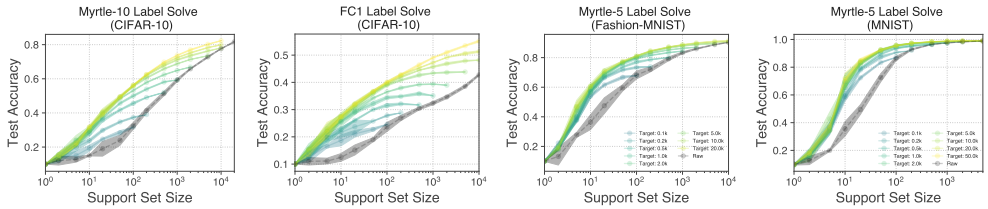

Figure 2: **LS performance for Myrtle-(5/10) and FC on CIFAR-10/Fashion-MNIST/MNIST.**

5.2 KERNEL TO KERNEL RESULTS

Here we see how robust our learned datasets are across various neural kernels. We evaluate our datasets using kernels coming from FC and Conv layers of depths 1-3, since such components form the basic building blocks of neural networks. We would have used deeper and more diverse architectures, but computational limits prevented otherwise, see the Experiment Details in Section D.

Table 1: **MNIST: KIP and LS vs baselines.** Baselines are Dataset Condensation (DC) (Zhao et al., 2020), Label Distillation (LD) (Bohdal et al., 2020) Soft Label Dataset Distillation (SLDD) (**?**) and Dataset Distillation (DD) (Wang et al., 2018). See Table A1 for a full comparison.

# Images	10	100	500	5000	10000
KIP RBF	**89.66±0.07**	97.45±0.05	98.47±0.05	98.78±0.02	98.80±0.02
KIP RBF (a + l)	91.48±0.05	**97.87±0.06**	**98.95±0.03**	**99.34±0.03**	**99.33±0.01**
KIP FC1 (a + l)	85.5±0.04	97.15±0.11	98.36±0.08	99.18±0.04	99.26±0.03
KIP → NN FC1	86.49±0.40	88.96±0.37	95.70±0.09	97.97±0.07	-
DC ConvNet	**91.7±0.5**	**97.4±0.2**	-	-	-
DD	-	79.5±8.1	-	-	-
LS Myrtle-5	**70.24±1.59**	**95.44±0.17**	**98.32±0.91**	**99.17±0.01**	-
LD	64.57±2.67	87.85±0.43	94.75±0.29	-	-

Table 2: **CIFAR-10: KIP and LS vs baselines.** See Table A2 for a full comparison.

# Images	10	100	500	5000	10000
KIP RBF (a + l)	**41.3±0.1**	**54.0±0.1**	**60.1±0.1**	**64.6±0.2**	64.6 ±0.2
KIP FC1 (a + l)	40.5±0.4	53.1±0.5	58.6±0.4	63.8±0.4	**64.7±0.4**
KIP → NN FC1	**36.2±0.1**	**45.7±0.3**	46.9±0.2	50.1±0.4	51.7±0.4
DC ConvNet	28.3±0.5	44.9±0.5	-	-	-
SLDD	-	39.8±0.8	-	-	-
LS Myrtle-10	**31.7±0.2**	**56.0±0.5**	**69.8±0.1**	**80.2±0.1**	**82.3±0.1**
LD	25.69±0.72	38.33±0.44	43.16±0.47	-	-

Here, we validate the fact that sampling multiple kernels during KIP makes the learned dataset robust. The resulting dataset performs well at test with respect to all the kernels used during train time. See Figure A2, which shows that KIP -datasets trained with a single kernel, do on average, worse than KIP -datasets trained using a family of kernels.

Remarkably, LS between FC1 and Myrtle-10 kernels on CIFAR-10 is highly robust, see Figure 2. That is, one can label solve using FC1 and train Myrtle-10 using those labels and vice versa. There is only a negligible difference in performance in nearly all instances between data with transferred learned labels and with natural labels.

5.3 Kernel to Neural Networks Results

Significantly, KIP -learned datasets transfer remarkably well to the task of training neural networks. We verify this conclusion along the following dimensions:

Robustness to dataset size: For MNIST, we considered six datasets: those with 10, 100, and 200 images that are either uncorrupted (uncrpt) or 90% corrupted (crpt), see Table A6. The rows marked KIP→NN use KIP -generated data (with kernels sampled from FC1-3, Conv1-2) for uncorrupted data and KIP $_{\rho=0.9}$ for corrupted data (where 90% of the pixels are corrupted with uniform noise) to train the neural network NN. The rows with a neural network use natural image data (uncorrupted) or natural image data with 90% pixels corrupted with random noise.

We find that our in-distribution transfer (the downstream network FC1 has its neural kernel included among the kernels sampled by KIP) performs remarkably well, with both corrupted *and* uncorrupted KIP images beating the uncorrupted natural images of corresponding size. Out of distribution (LeNet neural network trainings) have less transferability: the uncorrupted images still outperform natural images, and corrupted KIP images still outperform corrupted natural images, but corrupted KIP images no longer outperform uncorrupted natural images.

Robustness to hyperparameters: For CIFAR-10, we took 100 images (clean and 90% corrupted with noise) and trained networks on a wide variety of hyperparameters for various neural architectures, both those whose corresponding neural kernels were involved in training the 100 images and those that were not. We found that the KIP -learned images almost always outperform 100 random

natural images, with the optimal set of hyperparameters yielding a margin close to that predicted from the KRR setting, see Figure 3.

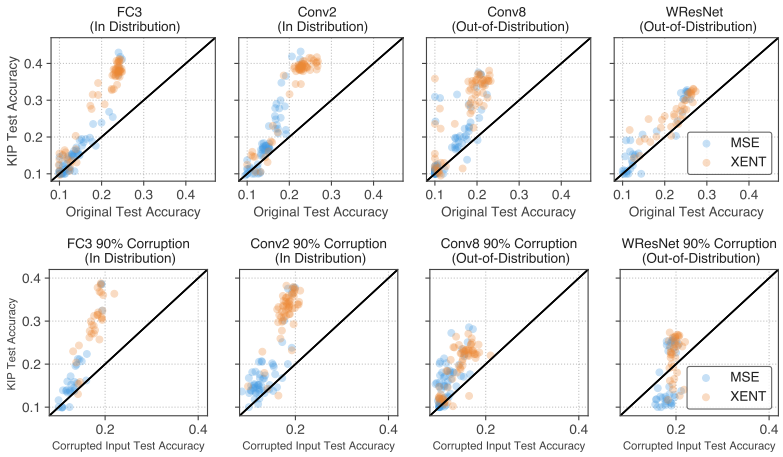

Figure 3: **KIP learned images transfers well to finite neural networks.** Random subset of 100 images of CIFAR-10 and KIP learned images are compared. Each scatter point corresponds to varying hyperparameters for training (e.g. learning rate or L2 regularization coefficient). We observe that KIP learned dataset can train better performing neural networks compared to random subset of training points of the same size.

6 Conclusion

We introduced novel algorithms `KIP` and `LS` for the meta-learning of datasets. We obtained a variety of compressed and corrupted datasets, achieving state of the art results for KRR and neural network dataset distillation methods. This was achieved even using the simplest of kernels and neural networks (shallow fully-connected networks and purely-convolutional networks without pooling), which notwithstanding their expressive disadvantage, outperform most baselines that use deeper architectures. Followup work will involve scaling up `KIP` to deeper architectures with pooling (achievable with multi-device training) for which we expect to obtain even more highly performant datasets, both in terms of overall accuracy and architectural flexibility. Finally, we obtained datasets showing a surprising amount of robustness to corruption in performance, which when developed at scale, could lead to practical applications for privacy-preserving machine learning.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Oct 2016. doi: 10.1145/2976749.2978318. URL `http://dx.doi.org/10.1145/2976749.2978318`.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pp. 8141–8150. Curran Associates, Inc., 2019a.

Sanjeev Arora, Simon S Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. *arXiv preprint arXiv:1910.01663*, 2019b.

Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Flexible dataset distillation: Learn labels instead of images. *arXiv preprint arXiv:2006.08572*, 2020.

Zalán Borsos, Mojmír Mutnỳ, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *arXiv preprint arXiv:2006.03875*, 2020.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Hongge Chen, Si Si, Yang Li, Ciprian Chelba, Sanjiv Kumar, Duane Boning, and Cho-Jui Hsieh. Multi-stage influence function, 2020.

Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *journal of machine learning research*, 6(Dec):2153–2175, 2005.

Adrià Garriga-Alonso, Laurence Aitchison, and Carl Edward Rasmussen. Deep convolutional networks as shallow gaussian processes. In *International Conference on Learning Representations*, 2019.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

Mahmut Kaya and H.s Bilge. Deep metric learning: A survey. *Symmetry*, 11:1066, 08 2019. doi: 10.3390/sym11091066.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Sam Schoenholz, Jeffrey Pennington, and Jascha Sohl-dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*, 2019a.

Jaehoon Lee, Samuel S Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *arXiv preprint arXiv:2007.15801*, 2020.

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10657–10665, 2019b.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552. PMLR, 2020.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pp. 2113–2122, 2015.

Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems*, 2014.

Alexander G. de G. Matthews, Jiri Hron, Mark Rowland, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations*, 2018.

Radford M. Neal. Priors for infinite networks (tech. rep. no. crg-tr-94-1). *University of Toronto*, 1994.

Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2019.

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

Vaishaal Shankar, Alex Chengyu Fang, Wenshuo Guo, Sara Fridovich-Keil, Ludwig Schmidt, Jonathan Ragan-Kelley, and Benjamin Recht. Neural kernels without tangents. In *International Conference on Machine Learning*, 2020.

Sam Shleifer and Eric Prokop. Using small proxy datasets to accelerate hyperparameter search. *arXiv preprint arXiv:1906.04887*, 2019.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4077–4087. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning.pdf`.

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pp. 1257–1264, 2006.

Jascha Sohl-Dickstein, Roman Novak, Samuel S Schoenholz, and Jaehoon Lee. On the infinite width limit of neural networks with a standard parameterization. *arXiv preprint arXiv:2001.07301*, 2020.

Ilia Sucholutsky and Matthias Schonlau. Soft-label dataset distillation and text dataset distillation. *arXiv preprint arXiv:1910.02551*, 2019.

Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pp. 567–574, 2009.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3630–3638. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6385-matching-networks-for-one-shot-learning.pdf`.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Christopher KI Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pp. 682–688, 2001.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. *arXiv preprint arXiv:2006.05929*, 2020.

A REMARKS ON DEFINITION OF ϵ-APPROXIMATION

Here, we provide insight to the formulation of Definition 2. On the one hand, we should allow for a flexible set of algorithms when comparing D and $\tilde{D}$. For instance, a mere preprocessing of the dataset (e.g. by rescaling it), should be regarded as producing an equivalent dataset, yet such a rescaling may affect the hyperparameters needed to train an equivalent model (e.g. the learning rate). Thus, in considering various datasets, one must allow the relevant hyperparameters of an algorithm to vary. On the other hand, it would be impossible to compare two datasets meaningfully if the learned algorithms trained upon them differed too significantly. For instance, if D is a much larger dataset than $\tilde{D}$, but A is a much less expressive algorithm than $\tilde{A}$, then the two datasets may be ϵ-approximations of each other, but it would be strange to compare D and $\tilde{D}$ in this way. Thus, we treat the notion of what class of algorithms to consider informally, and leave its specification as a practical matter for each use case. In practice, the pair of algorithms we use to compare datasets should be drawn from family in which some reasonable range of hyperparameters are varied, the ones one typically tunes when learning on an unknown dataset. In what follows, we usually omit our reference to the algorithm A, since it is either implicit from the context or else best left underspecified.

Another key feature of our definition is that datapoints of an ϵ-approximating dataset must have the same size. This makes our notion of a dataset more restrictive than merely returning a specialized set of extracted features from some initial dataset.

B TUNING KIP

Sampling: For optimizing for KRR performance with support dataset size N, it is best to learn a support set $\tilde{D}$ of size N and sample this entire set during KIP training. It is our observation that subsets of size $M < N$ of $\tilde{D}$ will not perform as well as optimizing directly for a size M dataset through KIP. Conversely, sampling subsets of size M from a support dataset of size N during KIP will not lead to a dataset that does as well as optimizing for all N points. This is sensible: optimizing for small support size requires a resourceful learning of coarse features at the cost of learning fine-grained features from many support datapoints. In reverse, optimizing a large support set means the learned support set has leveraged higher-order information, which will degrade when restricted to smaller subsets.

For sampling from the target set, which we always do in a class-balanced way, we found larger batch sizes to perform better if the kernel K is fixed. For MNIST, sampling 10% of training dataset for the targets sufficed. For CIFAR, it was always best to sample from the entire training dataset for the targets (i.e. full gradient descent).

Initialization: We tried two sets of initializations. The first ("image init") initializes (X_s, y_s) to be a subset of (X_t, y_t). The second ("noise init") initializes X_s with uniform noise and y_s with centered, 1-hot labels (in a class-balanced way). We found image initialization to perform better.

Number of Training Iterations: Remarkably, KIP converges very quickly in all experimental settings we tried. After only a hundred iterations, independently of the support size, kernel, and corruption factor, the learned support set has already undergone the majority of its learning (test accuracy is within more than 90% of the final test accuracy). For the platforms available to us, using a single V100 GPU, one hundred steps takes on the order of about 10 minutes of training time for the experiments we ran involving target batch sizes that were a few thousand. When we add augmentations to our targets, performance continues to improve slowly over time before flattening out after several thousands of iterations.

C THEORETICAL RESULTS

Here, we analyze convergence properties of KIP in returning an ϵ-approximate dataset. In what follows, we refer to gradient-descent KIP as the case when we sample from the entire support and train datasets for each update step to KIP. Without loss of generality and to simplify notation, we assume that the dimensionality of label spaces is $C = 1$.

We sketch a proof of the below convergence theorem for the case of a linear kernel[2]:

Theorem 1. *Let $D = (X_t, y_t)$ be a generic[3] dataset in $\mathbb{R}^d$ sampled from a distribution $\mathcal{P}$ which is supported on $\|x\| \leq 1$. Let $w_\lambda \in \mathbb{R}^d$ be the coefficients of the λ-ridge regressor trained on (X_t, y_t).*

1. *For generic initial conditions and sufficiently small $\lambda > 0$, gradient descent* `KIP` *yields a dataset $\tilde{D}$ whose λ-ridge regressor $\tilde{w} \in \mathbb{R}^d$ is a least squares classifier for D.*

2. *The dataset $\tilde{D}$ is a strong ϵ-approximation to D with respect to λ-KRR, where*

$$\epsilon = \|\tilde{w} - w_\lambda\|. \tag{A1}$$

   In particular, if D has a unique least squares classifier, then $\epsilon \to 0$ as $\lambda \to 0$.

3. *The same holds for* $\texttt{KIP}_\rho$ *for many choices of ρ that is not too large and corresponding corruption mask.*

Proof. We discuss the case where X_s is optimized, with the case where both (X_s, y_s) are optimized proceeding similarly. First, we establish convergence. For a linear kernel, we can write our loss function as

$$L(X_s) = \frac{1}{2}\|y_t - X_t X_s^T (X_s X_s^T + \lambda I)^{-1} y_s\|, \tag{A2}$$

defined on the space $\mathbb{M}_{n_s \times d}$ of $n_s \times d$ matrices. It is the pullback of the loss function

$$L_{d \times \mathbb{R}^{n_s}}(\Phi) = \frac{1}{2}\|y_t - X_t \Phi y_s\|, \qquad \Phi \in \mathbb{R}^{d \times n_s} \tag{A3}$$

under the map $X_s \mapsto \Phi(X_s) = X_s^T (X_s X_s^T + \lambda I)^{-1}$. The function (A3) is quadratic in Φ and all its local minima are global minima given by an affine subspace $\mathcal{M} \subset \mathbb{M}_{d \times n_s}$. Moreover, each point of $\mathcal{M}$ has a stable manifold of maximal dimension equal to the codimension of $\mathcal{M}$. Thus, the functional L has global minima given by the inverse image $\Phi^{-1}(\mathcal{M})$ (which will be nonempty for sufficiently small λ). Any other potential critical points of L will be saddle points and not local minima, i.e. the Hessian of L at such a critical point will be negative semi-definite. If such a critical point has a negative eigenvalue, then the set of points which do not converge to such a critical is open and dense. Generic choices of (X_t, y_t) and sufficiently small $\lambda > 0$ will be such that all such non-minimal critical points will not have identically zero Hessian.

Thus, a generic gradient trajectory γ of L will be such that $\Phi \circ \gamma$ is a gradient-like[4] trajectory for $L_{d \times \mathbb{R}^{n_s}}$ that converges to $\mathcal{M}$. We have to show that γ itself converges. Observe that $\Phi(X_s)$ computes the λ-regularized pseudo-inverse of X. Thus, if $X = V \Sigma U^T$ is a singular decomposition of X, with Σ a diagonal matrix of singular values, we have $A(X) = U \phi(\Sigma) V^T$, where $\phi(\Sigma)$ denotes the diagonal matrix with $\phi(\mu) = \frac{\mu}{\mu^2 + \lambda}$ applied to each singular value. Since ϕ is a two-to-one map onto its image, away from $\mu \in \{0, \sqrt{\lambda}\}$, then same is true of Φ, away from locus of matrices with those exceptional singular values. It is convenient to consider ϕ as a map on the one-point compactification $[0, \infty] \supset \mathbb{R}^{\geq 0}$, so as to make ϕ a two-to-one map away from $\mu \neq \sqrt{\lambda}$. Via the singular-value decomposition, this induces a corresponding compactification $\overline{\mathbb{M}}^{n_s \times d}$ of $\mathbb{M}^{n_s \times d}$, and we can naturally extend Φ to such a compactification. We have that γ converges to an element of $\tilde{\mathcal{M}} := \Phi^{-1}(\mathcal{M}) \subset \overline{\mathbb{M}}^{n_s \times d}$, where we need the compactification to account for the fact that when $\Phi \circ \gamma$ converges to a matrix that has a zero singular value, γ may have one of its singular values growing to infinity. Let $\mathcal{M}_0$ denote the subset of $\mathcal{M}$ with a zero singular value. Then γ converges to an element of $\mathbb{M}^{n_s \times d}$ precisely when γ does not converge to an element of $\tilde{\mathcal{M}}_\infty := \Phi^{-1}(\mathcal{M}_0) \cap (\overline{\mathbb{M}}^{n_s \times d} \setminus \mathbb{M}^{n_s \times d})$. However, $\mathcal{M}_0 \subset \mathcal{M}$ has codimension 1 and hence so does $\tilde{\mathcal{M}}_\infty \subset \Phi^{-1}(\mathcal{M})$. Thus, the stable set to $\tilde{\mathcal{M}}_\infty$ will be nongeneric. Hence, we have generic convergence of a gradient trajectory of L to a (finite) solution.

[2] A more exacting analysis would go into the precise details of the genericity properties that we invoke, which is outside our main focus.

[3] A set can be generic by either being open and dense or else having probability one.

[4] A vector field v is gradient-like for a function f if $v \cdot \mathrm{grad}(f) \geq 0$ everywhere.

Observe that L is also the pullback of the loss function

$$L_{\mathbb{R}^d}(w) = \frac{1}{2}\|y_t - X_t w\|, \qquad w \in \mathbb{R}^d. \tag{A4}$$

under the map $X_s \mapsto w(X_s) = X_s(X_s X_s^T + \lambda I)^{-1} y_s$. The function $L_{\mathbb{R}^d}(w)$ is quadratic in w and has a unique minimum value, with the space of global minimum being an affine subspace W^* of $\mathbb{R}^d$ given by the least squares classifiers for the dataset (X_t, y_t). Thus, the global minima of L are the preimage of W^* under the map $w(X_s)$. Altogether, this and the convergence result establish Part 1.

For part 2, the difference of any two linear models, when evaluated on $\mathcal{P}$, can be pointwise bounded by the norm of the difference of the model coefficients. It follows from the triangle inequality that $\tilde{D}$ is a strong ϵ-approximation to D with respect to λ-KRR, where ϵ is given by (A1).

Finally, for part 3, we repeat our prior analysis, only now we restrict our analysis to subspaces of matrices whose entries are fixed. For corruptions that do not violate genericity assumptions, (by being mild or else not doing something as disastrous as zeroing out entire rows and columns to enforce zero singular values), the preceding analysis can be repeated verbatim. □

Though trivial to prove, we make the following useful observation for the case of general kernels concerning the optimal output of `KIP` for general kernels.

Theorem 2. *Fix a dataset (X_t, y_t) of size N. Regard $y_t \in \mathbb{R}^{n_t}$. Consider the family of all subspaces $\mathcal{S}$ of $\mathbb{R}^N$ given by $\{\operatorname{im} K_{X_t X}\}$ where X varies over the space of all possible inputs of size M. Then the infimum of the loss (4) over all possible (X_s, y_s) is equal to $\epsilon^* := \inf_{S \in \mathcal{S}} |\Pi_S^{\perp} y_t|$ where $\Pi_S^{\perp}$ is orthogonal projection onto the orthogonal complement of S. Thus, insofar as train loss approximates population loss, gradient-descent `KIP` returns an ϵ^*-approximate dataset to (X_t, y_t) with respect to λ-KRR.*

Proof. Since y_s is trainable, $(K_{X_s X_s} + \lambda)^{-1} y_s$ is an arbitrary vector in $\mathbb{R}^{n_s}$. Thus, minimizing the training objective corresponds to maximizing the range of the linear map $K_{X_t X_s}$ over all possible X_s. The result follows. □

D Experiment Details

In all `KIP` trainings, we used the Adam optimizer. We used learning rates 0.01 and 0.04 for the MNIST and CIFAR-10 datasets, respectively. When sampling target batches, we always do so in a class-balanced way. When augmenting data, we used the ImageGenerator class from Keras, which enables us to add horizontal flips, height/width shift, rotatations (up to 10 degrees), and channel shift (for CIFAR-10).

Single kernel results: For FC, we used kernels with NTK parametrization and width=1024. For RBF, we set the scale $\gamma = 1.0$, where our rbf kernel is given by

$$\mathrm{rbf}(x1, x2) = \exp(-\gamma \|x_1 - x_2\|^2 / d) \tag{A5}$$

where d is the dimension of the inputs, and treat that as a learnable parameter during `KIP` . We found that training the kernel parameters for FC (the weight initialization standard deviation) did not prove helpful.

For MNIST, we found target batch size equal to 6k (10%) sufficient. For CIFAR-10, it helped to sample the entire training dataset of 50k images per step (hence, along with sampling the full support set, we are doing full gradient descent training).

We train for 10-20k iterations and took 5 random subsets of images for initializations. Take 5 checkpoints with lowest train mse loss and look at distribution of test loss. This gives 25 checkpoints per hps and we take mean and std. We find train/test mse/acc highly correlated with no overfitting, hence the std being so low

Kernel transfer results: For kernel transfer, both to other kernels and to neural networks, we found it useful to use smaller target batch sizes (1k-5k), else the images overfit to their source kernel.

For random sampling of kernels, we used kernels FC1-3 with width 1024, and Conv1-2 with width 128.[5] The result of this is that the KIP learned images do well on a variety of FC and Conv kernels as well as their neural network counterparts. Unfortunately, the high compute cost and memory footprint of convolutional kernels means that, using a single V100 GPU with 16GB of RAM, we were (i) only able to sample shallow kernels; (ii) for convolutional kernels, limited to small support sets and small target batch sizes; (iii) unable to use pooling if learning more than just a few images. Scaling up KIP to deeper, more expensive architectures, which would require multi-device training, will be the subject of future exploration.

For finite network transfer experiments on MNIST images, we trained the networks with constant learning rate and Adam optimizer with cross entropy loss. Learning rate was tuned over small grid search space. All networks used ReLU activation function. For FC we use width of 1024.

For finite network transfer experiments on CIFAR-10 images, we trained the networks with constant learning rate, momentum optimizer with momentum 0.9. Learning rate, L2 regularization, parameterization (standard vs NTK) and loss type (mean squared error, softmax-cross-entropy) was tuned over small grid search space. All network used ReLU activation function with $\sigma_w^2 = 2$. Convolution layers uses $(3, 3)$ filter with stride 1 and SAME padding. Vanilla network uses same width for all depth and for FC we use width of 1024 and for Conv we used 512 channels Conv2 and 128 channels for Conv8. No pooling layers were used except for the WideResNet experiment where we follow the original architecture of Zagoruyko & Komodakis (2016) except for not using exponential moving average for batch normalization parameters.

E TABLES AND FIGURES

E.1 DATASET DISTILLATION BASELINES

We present our KIP results alongside prior art for both MNIST (Table A1) and CIFAR-10 (Table A2). The first block of Table A1 shows KRR results. Rows with (a + l) denote the experiments with augmentations plus learning of labels in KIP , while those without (a + l) involve only the learning of the support images. The second block of the table shows neural network trainings on distilled datasets involving images and/or labels. The last block shows only label distillation (LD) or else label solving (LS). These comparisons occur cross-architecturally, but aside from LS Myrtle-5, all our results involve shallow architectures or kernels, whereas prior art use deeper architectures. Corresponding comments apply to Table A2. In nearly all instances, we achieve higher test performance per dataset size.

Table A1: **MNIST: KIP and LS vs baselines.**

# Images	10	100	500	5000	10000
KIP RBF	**89.66±0.07**	97.45±0.05	98.47±0.05	98.78±0.02	98.80±0.02
KIP RBF (a + l)	91.48±0.05	**97.87±0.06**	**98.95±0.03**	**99.34±0.03**	**99.33±0.01**
KIP FC1	89.3±0.01	96.64±0.08	97.64±0.06	98.52±0.04	98.59±0.05
KIP FC1 (a + l)	85.5±0.04	97.15±0.11	98.36±0.08	99.18±0.04	99.26±0.03
KIP → NN FC1	86.49±0.40	88.96±0.37	95.70±0.09	97.97±0.07	-
DC ConvNet	**91.7±0.5**	**97.4±0.2**	-	-	-
DC LeNet†	-	93.9±0.6	-	-	-
SLDD	-	82.7±2.8	-	-	-
DD	-	79.5±8.1	-	-	-
LS FC1	61.0±0.1	87.6±0.7	94.3±0.1	97.6±0.0	-
LS Myrtle-5	**70.24±1.59**	**95.44±0.17**	**98.32±0.91**	**99.17±0.01**	-
LD	64.57±2.67	87.85±0.43	94.75±0.29	-	-

† DC LeNet (Zhao et al., 2020), LD (Bohdal et al., 2020), SLDD (Sucholutsky & Schonlau, 2019), DD (Wang et al., 2018) use the LeNet architecture.

[5] While with NTK parameterization Jacot et al. (2018), the kernel does not have any width dependence, with the improved standard parameterization of Sohl-Dickstein et al. (2020), one could encode width dependence into NTK.

Table A2: **CIFAR-10: KIP and LS vs baselines.**

# Images	10	100	500	5000	10000
KIP RBF	40.1± 1.0	50.2±0.3	51.8±0.6	52.7±0.6	52.2± 0.4
KIP RBF (a + l)	**41.3±0.1**	**54.0±0.1**	**60.1±0.1**	**64.6±0.2**	64.6 ±0.2
KIP FC1	39.3±1.6	49.1±1.1	52.1±0.8	54.5±0.9	54.9±0.6
KIP FC1 (a + l)	40.5±0.4	53.1±0.5	58.6±0.4	63.8±0.4	**64.7±0.4**
KIP → NN FC1	**36.2±0.1**	**45.7±0.3**	46.9±0.2	50.1±0.4	51.7±0.4
DC ConvNet	28.3±0.5	44.9±0.5	-	-	-
DC AlexNet‡	-	39.1±1.2	-	-	-
SLDD	-	39.8±0.8	-	-	-
DD	-	36.8±1.2	-	-	-
LS FC1	27.5±0.3	40.1±0.3	46.4±0.4	53.5±0.2	55.1±0.3
LS Myrtle-10	**31.7±0.2**	**56.0±0.5**	**69.8±0.1**	**80.2±0.1**	**82.3±0.1**
LD	25.69±0.72	38.33±0.44	43.16±0.47	-	-

‡ DC AlexNet, LD, SLDD, DD use the AlexNet architecture.

E.2 KERNEL BASELINES

We report various baselines of kernels on natural images. Table A3 shows how various kernels vary in performance with respect to random subsets of CIFAR-10. We compute the RBF and FC1 columns ourselves, where RBF denotes the rbf kernel (A5) with $\gamma = 1$ and FC1 uses standard parametrization and width 1024. Comparing with Table A2, we see that KIP on 10 images outperforms 1028 natural images, thereby achieving a compression ratio of over 100. This compression ratio narrows as the support size increases towards the size of the training data.

Next, Table A4 compares KIP on RBF and FC1 to other kernels on MNIST (other numbers taken from Shankar et al. (2020)). We see that our KIP approach, even with 10k images (which fits into memory), leads to RBF and FC1 matching the performance of convolutional kernels on the entire training dataset.

Table A5 shows state of the art of FC kernels on CIFAR-10 (numbers taken from Lee et al. (2020)). The prior state of the art used kernel ensembling on batches of augmented data in Lee et al. (2020) to obtain test accuracy of 61.5% and 62.4% for the NTK and NNGP kernels. By distilling augmented images using KIP , we are able to obtain 64.7% using only 10k images.

Table A3: **Accuracy on random subsets of CIFAR-10 with standard deviations over 20 resamplings.**

# Images \ Kernel	Linear	RBF	FC1	CNTK†	Myrtle10-G‡
10	12.94 ± 0.74	15.7 ± 2.1	16.4 ± 1.8	15.33 ± 2.43	19.15 ± 1.94
20	13.54 ± 0.69	17.1 ± 1.7	18.0 ± 1.9	18.79 ± 2.13	21.65 ± 2.97
40	14.66 ± 0.60	19.7 ± 1.8	20.6 ± 1.8	21.34 ± 1.91	27.20 ± 1.90
80	15.54 ± 0.61	23.0 ± 1.5	23.9 ± 1.6	25.48 ± 1.91	34.22 ± 1.08
160	17.15 ± 0.64	25.8 ± 1.4	26.5 ± 1.4	30.48 ± 1.17	41.89 ± 1.34
320	19.18 ± 0.71	29.2 ± 1.2	29.9 ± 1.1	36.57 ± 0.88	50.06 ± 1.06
640	22.30 ± 0.57	32.8 ± 0.9	33.4 ± 0.8	42.63 ± 0.68	57.60 ± 0.48
1280	25.64 ± 0.61	35.9 ± 0.7	36.7 ± 0.6	48.86 ± 0.68	64.40 ± 0.48

† Conv14 kernel with global average pooling, Arora et al. (2019b)
‡ Myrtle10-Gaussian kernel, Shankar et al. (2020)

E.3 KIP AND LS TRANSFER

In Figure A1, we plot how LS learned labels using Myrtle-10 kernel transfer to the FC1 kernel and vice versa. We vary the number of targets and support size. We find remarkable stability across all

Table A4: **Classification performance on MNIST.**

Method	Accuracy
NTK	98.6
ArcCosine Kernel	98.8
Gaussian Kernel	98.8
KIP FC1 (a + l), 10k images (This work)	99.2
LeNet-5 (LeCun et al., 1998)	99.2
KIP RBF (a + l), 10k images (This work)	99.3
Myrtle5 Kernel(Shankar et al., 2020)	99.5
CKN (Mairal et al., 2014)	99.6

Table A5: **CIFAR-10 test accuracy for FC kernels.**

Architecture	Method	NTK	NNGP
FC	Novak et al. (2019)	-	59.9
	Lee et al. (2020)	61.5	62.4
	KIP FC (a + l), 10k images	**64.7**	-

these dimensions in the sense that while the gains from `LS` may be kernel-specific, `LS` -labels do not perform meaningfully different from natural labels when switching the train and evaluation kernels.

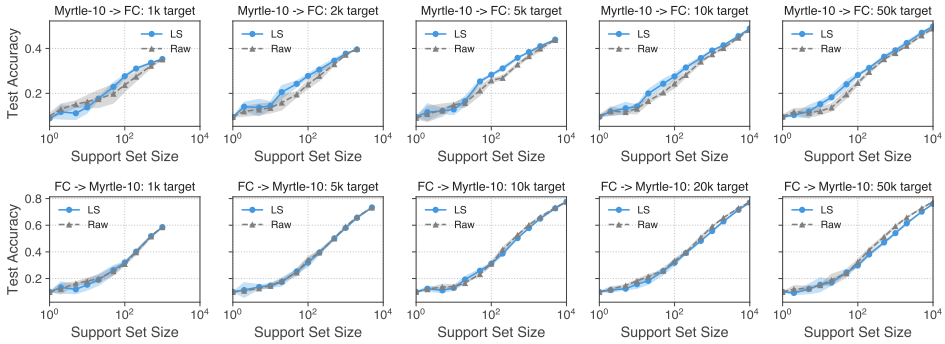

Figure A1: **Label Solve transfer between Myrtle-10 and FC.**

In Figure A2, we plot how `KIP` (with only images learned) performs across kernels. We have seven training scenarios: training individually on FC1, FC2, FC3, Conv1, Conv2, Conv3 NTK kernels and random sampling from among all six kernels uniformly (Avg All). We learn datasets of size 10, 100, 200 and then evaluate them by averaging over all of FC1-3, Conv1-3, both with the NTK and NNGP kernels for good measure. This average performance is recorded along the y-axis. We see that AvgAll leads to overall boost in performance across kernels. Another observation is that Conv kernels alone tend to do a bit better, averaged over the kernels considered, than FC kernels alone.

Next, we have `KIP` to neural network training experiments. Results for MNIST and CIFAR are shown in Tables A6 and E.3. For the interpretation of these tables, see Section 5.3.

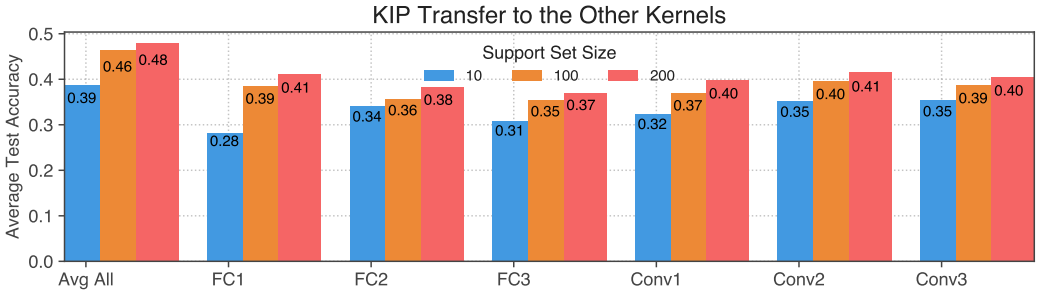

Figure A2: **Studying transfer between kernels.**

Table A6: **Test accuracy of KIP transfer to NN vs NN baselines on MNIST.** For each group of four experiments, the best number is marked **boldface**, while the second best number is in *italics*.

Method	10 uncrpt	10 crpt	100 uncrpt	100 crpt	200 uncrpt	200 crpt
KIP $\rightarrow$ FC1	**73.57±1.51**	*44.95±1.23*	**86.84±1.65**	*79.73±1.10*	**89.55±0.94**	*83.38±1.37*
FC1	42.28±1.59	35.00±2.33	72.65±1.17	45.39±2.25	81.70±1.03	54.20±2.61
KIP $\rightarrow$ LeNet	**59.69±8.98**	38.25±6.42	**87.85±1.46**	69.45±3.99	**91.08±1.65**	70.52±4.39
LeNet	*48.69±4.10*	30.56±4.35	*80.32±1.26*	59.99±0.95	*89.03±1.13*	62.00±0.94

Table A7: **KIP transfer to NN vs NN baselines on CIFAR-10.** For each block, the best number is marked **boldface**, while the second best number is in *italics*.

Method	100 uncrpt	100 crpt
KIP $\rightarrow$ FC3	**43.09±0.20**	*37.71±0.38*
FC3	24.48±0.15	18.92±0.61
KIP $\rightarrow$ Conv2	**43.68±0.46**	*37.08±0.48*
Conv2	26.23 ±0.69	17.10±1.33
KIP $\rightarrow$ WideResNet	**33.29±1.14**	23.89±1.30
WideResNet	*27.93±0.75*	19.00±1.01

F EXAMPLES OF KIP LEARNED SAMPLES

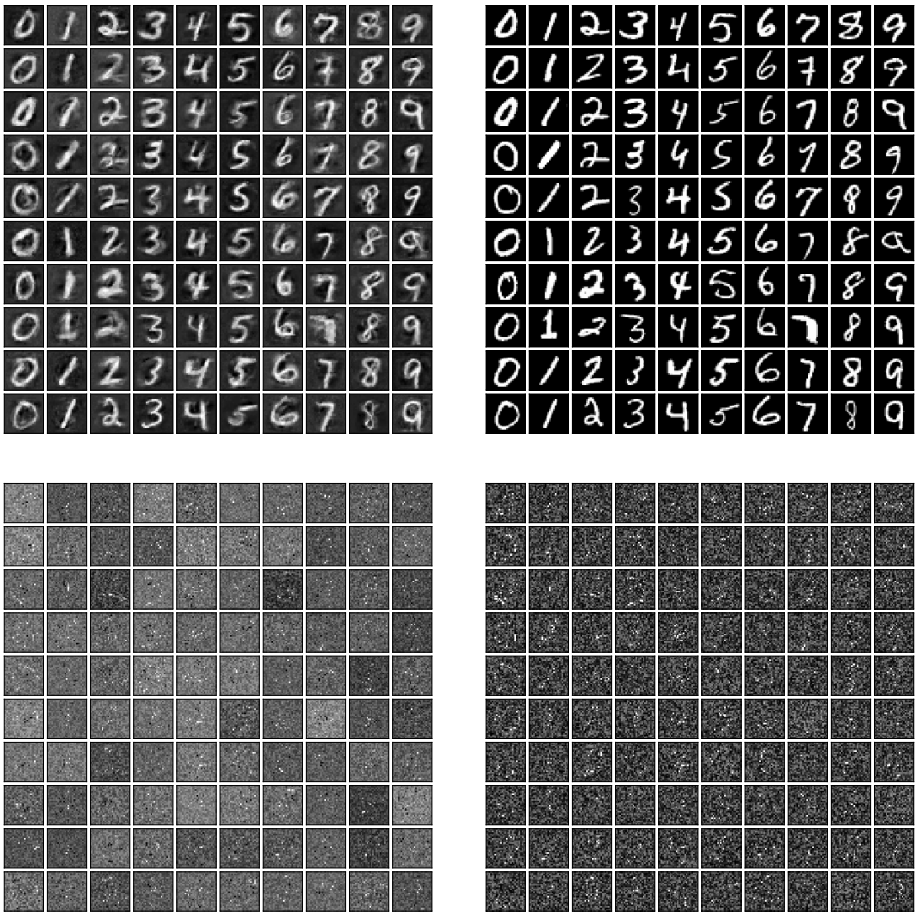

Figure A3: **KIP learned images (left) vs untrained MNIST images (right).** Natural images (upper), 90% noise corruption (lower).

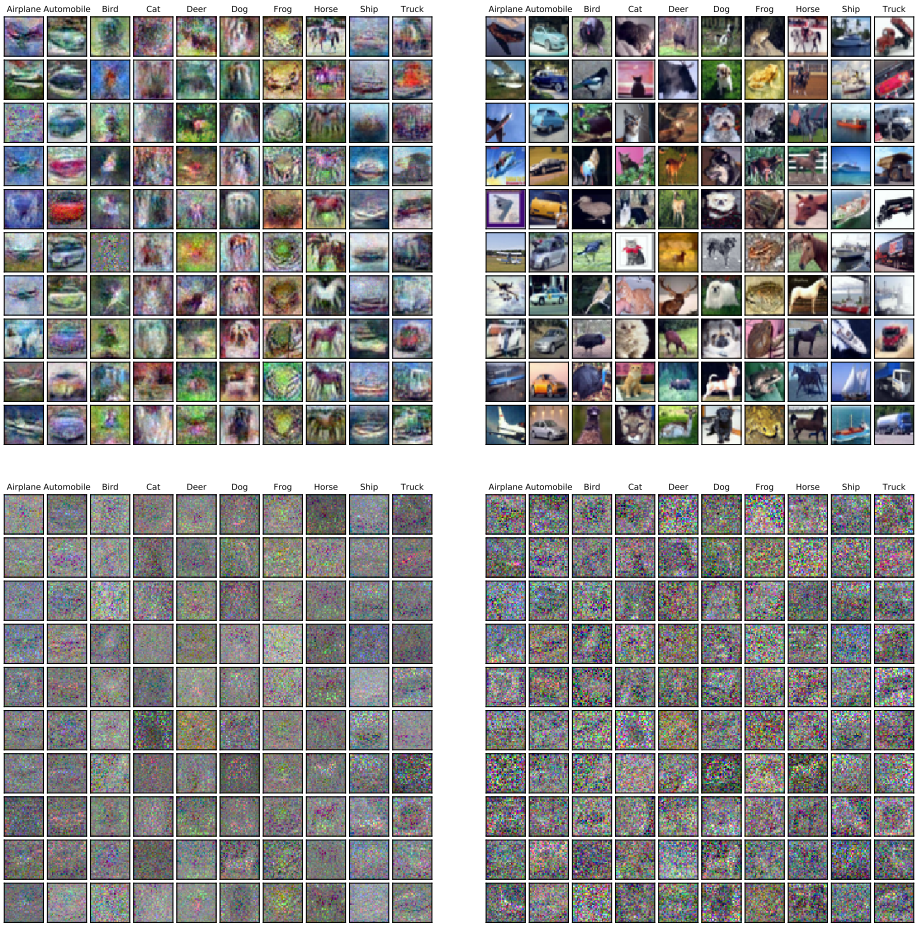

Figure A4: **KIP learned images (left) vs untrained CIFAR-10 images (right).** Natural images (upper), 90% noise corruption (lower).