Convexified Convolutional Neural Networks

Yuchen Zhang [1] **Percy Liang** [1] **Martin J. Wainwright** [2]

Abstract

We describe the class of convexified convolutional neural networks (CCNNs), which capture the parameter sharing of convolutional neural networks in a convex manner. By representing the nonlinear convolutional filters as vectors in a reproducing kernel Hilbert space, the CNN parameters can be represented in terms of a low-rank matrix, and the rank constraint can be relaxed so as to obtain a convex optimization problem. For learning two-layer convolutional neural networks, we prove that the generalization error obtained by a convexified CNN converges to that of the best possible CNN. For learning deeper networks, we train CCNNs in a layer-wise manner. Empirically, we find that CCNNs achieve competitive or better performance than CNNs trained by backpropagation, SVMs, fully-connected neural networks, stacked denoising auto-encoders, and other baseline methods.

1. Introduction

Convolutional neural networks (CNNs) (LeCun et al., 1998) have proven successful across many tasks including image classification (LeCun et al., 1998; Krizhevsky et al., 2012), face recognition (Lawrence et al., 1997), speech recognition (Hinton et al., 2012), text classification (Wang et al., 2012), and game playing (Mnih et al., 2015; Silver et al., 2016). There are two principal advantages of a CNN over a fully-connected neural network: (i) sparsity—each nonlinear convolutional filter acts only on a local patch of the input, and (ii) parameter sharing—the same filter is applied to each patch.

However, as with most neural networks, the standard approach to training CNNs is based on solving a nonconvex optimization problem that is known to be NP-hard (Blum & Rivest, 1992). In practice, researchers use some flavor of stochastic gradient method, in which gradients are computed via backpropagation (Bottou, 1998). This approach has two drawbacks: (i) the rate of convergence, which is at best only to a local optimum, can be slow due to nonconvexity (for instance, see the paper (Fahlman, 1988)), and (ii) its statistical properties are very difficult to understand, as the actual performance is determined by some combination of the CNN architecture along with the optimization algorithm.

In this paper, with the goal of addressing these two challenges, we propose a new model class known as *convexified convolutional neural networks* (CCNNs). These models have two desirable features. First, training a CCNN corresponds to a convex optimization problem, which can be solved efficiently and optimally via a projected gradient algorithm. Second, the statistical properties of CCNN models can be studied in a precise and rigorous manner. We obtain CCNNs by convexifying two-layer CNNs; doing so requires overcoming two challenges. First, the activation function of a CNN is nonlinear. In order to address this issue, we relax the class of CNN filters to a reproducing kernel Hilbert space (RKHS). This approach is inspired by the paper (Zhang et al., 2016a), which put forth a relaxation for fully-connected neural networks. Second, the parameter sharing induced by CNNs is crucial to its effectiveness and must be preserved. We show that CNNs with RKHS filters can be parametrized by a low-rank matrix. Relaxing this low-rank constraint to a nuclear norm constraint leads to our final formulation of CCNNs.

On the theoretical front, we prove an *oracle inequality* on the generalization error achieved by our class of CCNNs, showing that it is upper bounded by the best possible performance achievable by a two-layer CNN given infinite data—a quantity to which we refer as the oracle risk—plus a model complexity term that decays to zero polynomially in the sample size. Our results suggest that the sample complexity for CCNNs is significantly lower than that of the convexified fully-connected neural network (Zhang et al., 2016a), highlighting the importance of parameter sharing. For models with more than one hidden layer, our theory does not apply, but we provide encouraging empirical results using a greedy layer-wise training heuristic. Finally, we apply CCNNs to the MNIST handwritten digit dataset

[1]Stanford University, CA, USA [2]University of California, Berkeley, CA, USA. Correspondence to: Yuchen Zhang <zhangyuc@cs.stanford.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

as well as four variation datasets (VariationsMNIST), and find that it achieves state-of-the-art accuracy.

Related work. With the empirical success of deep neural networks, there has been an increasing interest in understanding its connection to convex optimization. Bengio et al. (2005) showed how to formulate neural network training as a convex optimization problem involving an infinite number of parameters. Aslan et al. (2013; 2014) propose a method for learning multi-layer latent-variable models. They showed that for certain activation functions, the proposed method is a convex relaxation for learning fully-connected neural networks.

Past work has studied learning translation-invariant features without backpropagation. Mairal et al. (2014) present convolutional kernel networks. They propose a translation-invariant kernel whose feature mapping can be approximated by a composition of the convolution, non-linearity and pooling operators, obtained through unsupervised learning. However, this method is not equipped with the optimality guarantees that we provide for CCNNs in this paper, even for learning one convolutional layer. The ScatNet method (Bruna & Mallat, 2013) uses translation and deformation-invariant filters constructed by wavelet analysis; however, these filters are independent of the data, in contrast to CCNNs. Daniely et al. (2016) show that a randomly initialized CNN can extract features as powerful as kernel methods, but it is not clear how to provably improve the model from random initialization.

Notation. For any positive integer n, we use $[n]$ as a shorthand for the discrete set $\{1, 2, \dots, n\}$. For a rectangular matrix A, let $\|A\|_*$ be its nuclear norm, $\|A\|_2$ be its spectral norm (i.e., maximal singular value), and $\|A\|_{\mathrm{F}}$ be its Frobenius norm. We use $\ell^2(\mathbb{N})$ to denote the set of countable dimensional vectors $v = (v_1, v_2, \dots)$ such that $\sum_{\ell=1}^{\infty} v_\ell^2 < \infty$. For any vectors $u, v \in \ell^2(\mathbb{N})$, the inner product $\langle u, v\rangle := \sum_{\ell=1}^{\infty} u_i v_i$ and the ℓ_2-norm $\|u\|_2 := \sqrt{\langle u, u\rangle}$ are well defined.

2. Background and problem setup

In this section, we formalize the class of convolutional neural networks to be learned and describe the associated non-convex optimization problem.

2.1. Convolutional neural networks

At a high level, a two-layer CNN[1] is a function that maps an input vector $x \in \mathbb{R}^{d_0}$ (e.g., an image) to an output vector in $y \in \mathbb{R}^{d_2}$ (e.g., classification scores for d_2 classes). This mapping is formed in the following manner:

- First, we extract a collection of P vectors $\{z_p(x)\}_{p=1}^P$ of the full input vector x. Each vector $z_p(x) \in \mathbb{R}^{d_1}$ is referred to as a *patch*, and these patches may depend on overlapping components of x.
- Second, given some choice of activation function $\sigma : \mathbb{R} \to \mathbb{R}$ and a collection of weight vectors $\{w_j\}_{j=1}^r$ in $\mathbb{R}^{d_1}$, we define the functions
$$h_j(z) := \sigma(w_j^\top z) \quad \text{for each patch } z \in \mathbb{R}^{d_1}. \tag{1}$$
Each function h_j (for $j \in [r]$) is known as a *filter*, and note that the same filters are applied to each patch—this corresponds to the *parameter sharing* of a CNN.
- Third, for each patch index $p \in [P]$, filter index $j \in [r]$, and output coordinate $k \in [d_2]$, we introduce a coefficient $\alpha_{k,j,p} \in \mathbb{R}$ that governs the contribution of the filter h_j on patch $z_p(x)$ to output $f_k(x)$. The final form of the CNN is given by $f(x) := (f_1(x), \dots, f_{d_2}(x))$, where the k^{th} component is given by
$$f_k(x) := \sum_{j=1}^{r} \sum_{p=1}^{P} \alpha_{k,j,p} h_j(z_p(x)). \tag{2}$$

Taking the patch functions $\{z_p\}_{p=1}^P$ and activation function σ as fixed, the parameters of the CNN are the filter vectors $\mathbf{w} := \{w_j \in \mathbb{R}^{d_1} : j \in [r]\}$ along with the collection of coefficient vectors $\boldsymbol{\alpha} := \{\alpha_{k,j} \in \mathbb{R}^P : k \in [d_2], j \in [r]\}$. We assume that all patch vectors $z_p(x) \in \mathbb{R}^{d_1}$ are contained in the unit ℓ_2-ball. This assumption can be satisfied without loss of generality by normalization: By multiplying a constant $\gamma > 0$ to every patch $z_p(x)$ and multiplying $1/\gamma$ to the filter vectors $\mathbf{w}$, this assumption holds without changing the the output of the network.

Given some positive radii B_1 and B_2, we consider the model class

$$\mathcal{F}_{\text{cnn}}(B_1, B_2) := \Big\{ f \text{ of the form (2)} : \max_{j \in [r]} \|w_j\|_2 \le B_1 \text{ and } \max_{k \in [d_2], j \in [r]} \|\alpha_{k,j}\|_2 \le B_2 \Big\}. \tag{3}$$

When the radii (B_1, B_2) are clear from context, we adopt $\mathcal{F}_{\text{cnn}}$ as a convenient shorthand.

2.2. Empirical risk minimization.

Given an input-output pair (x, y) and a CNN f, we let $\mathcal{L}(f(x); y)$ denote the loss incurred when the output y is predicted via $f(x)$. We assume that the loss function $\mathcal{L}$ is convex and L-Lipschitz in its first argument given any value of its second argument. As a concrete example, for multiclass classification with d_2 classes, the output vector y takes values in the discrete set $[d_2] = \{1, 2, \dots, d_2\}$. For example, given a vector $f(x) = (f_1(x), \dots, f_{d_2}(y)) \in \mathbb{R}^{d_2}$ of classification scores, the associated multiclass logistic loss for a pair (x, y) is given by $\mathcal{L}(f(x); y) := -f_y(x) +$

[1] Average pooling and multiple channels are also an integral part of CNNs, but these do not present any new technical challenges, so that we defer these extensions to Section 4.

$\log\big(\sum_{y'=1}^{d_2}\exp(f_{y'}(x))\big)$.

Given n training examples $\{(x_i, y_i)\}_{i=1}^n$, we would like to compute an empirical risk minimizer:

$$\widehat{f}_{\text{cnn}} \in \arg\min_{f\in\mathcal{F}_{\text{cnn}}} \sum_{i=1}^{n} \mathcal{L}(f(x_i); y_i). \tag{4}$$

Recalling that functions $f \in \mathcal{F}_{\text{cnn}}$ depend on the parameters $\mathbf{w}$ and $\boldsymbol{\alpha}$ in a highly nonlinear way (2), this optimization problem is nonconvex. As mentioned earlier, heuristics based on stochastic gradient methods are used in practice, which makes it challenging to gain a theoretical understanding of their behavior. Thus, in the next section, we describe a relaxation of the class $\mathcal{F}_{\text{cnn}}$ for which empirical risk minimization is convex.

3. Convexifying CNNs

We now turn to the development of the class of convexified CNNs. We begin in Section 3.1 by illustrating the procedure for the special case of the linear activation function. Although the linear case is not of practical interest, it provides intuition for our more general convexification procedure, described in Section 3.2, which applies to nonlinear activation functions. In particular, we show how embedding the nonlinear problem into an appropriately chosen reproducing kernel Hilbert space (RKHS) allows us to again reduce to the linear setting.

3.1. Linear activation functions: low rank relaxations

In order to develop intuition for our approach, let us begin by considering the simple case of the linear activation function $\sigma(t) = t$. In this case, the filter h_j when applied to the patch vector $z_p(x)$ outputs a Euclidean inner product of the form $h_j(z_p(x)) = \langle z_p(x),\, w_j\rangle$. For each $x \in \mathbb{R}^{d_0}$, we first define the $P \times d_1$-dimensional matrix

$$Z(x) := \begin{bmatrix} z_1(x)^\top \\ \vdots \\ z_P(x)^\top \end{bmatrix}. \tag{5}$$

We also define the P-dimensional vector $\alpha_{k,j} := (\alpha_{k,j,1}, \ldots, \alpha_{k,j,P})^\top$. With this notation, we can rewrite equation (2) for the k^{th} output as

$$\begin{aligned} f_k(x) &= \sum_{j=1}^{r}\sum_{p=1}^{P} \alpha_{k,j,p}\langle z_p(x),\, w_j\rangle = \sum_{j=1}^{r} \alpha_{k,j}^\top Z(x) w_j \\ &= \operatorname{tr}\Big(Z(x)\Big(\sum_{j=1}^{r} w_j \alpha_{k,j}^\top\Big)\Big) = \operatorname{tr}(Z(x)A_k), \end{aligned} \tag{6}$$

where in the final step, we have defined the $d_1 \times P$-dimensional matrix $A_k := \sum_{j=1}^r w_j \alpha_{k,j}^\top$. Observe that f_k now depends linearly on the matrix parameter A_k. Moreover, the matrix A_k has rank at most r, due to the parameter sharing of CNNs. See Figure 1 for a graphical illustration of this model structure.

Letting $A := (A_1, \ldots, A_{d_2})$ be a concatenation of these matrices across all d_2 output coordinates, we can then define a function $f^A : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ of the form

$$f^A(x) := (\operatorname{tr}(Z(x)A_1), \ldots, \operatorname{tr}(Z(x)A_{d_2})). \tag{7}$$

Note that these functions have a linear parameterization in terms of the underlying matrix A. Our model class corresponds to a collection of such functions based on imposing certain constraints on the underlying matrix A: in particular, we define $\mathcal{F}_{\text{cnn}}(B_1, B_2)$ to be the set of functions f^A which satisfies: **(C1)** $\max_{j\in[r]} \|w_j\|_2 \le B_1$, $\max_{k\in[d_2], j\in[r]} \|\alpha_{k,j}\|_2 \le B_2$; and **(C2)** $\operatorname{rank}(A) = r$. This is simply an alternative formulation of our original class of CNNs defined in equation (3). Notice that if the filter weights w_j are not shared across all patches, then the constraint (C1) still holds, but constraint (C2) no longer holds. Thus, the parameter sharing of CNNs is realized by the low-rank constraint (C2). The matrix A of rank r can be decomposed as $A = UV^\top$, where both U and V have r columns. The column space of matrix A contains the convolution parameters $\{w_j\}$, and the row space of A contains to the output parameters $\{\alpha_{k,j}\}$.

The matrices satisfying constraints (C1) and (C2) form a nonconvex set. A standard convex relaxation is based on the nuclear norm $\|A\|_*$ corresponding to the sum of the singular values of A. It is straightforward to verify that any matrix A satisfying the constraints (C1) and (C2) must have nuclear norm bounded as $\|A\|_* \le B_1 B_2 r\sqrt{d_2}$. Consequently, if we define the function class

$$\mathcal{F}_{\text{ccnn}} := \Big\{ f^A :\ \|A\|_* \le B_1 B_2 r\sqrt{d_2} \Big\}, \tag{8}$$

then we are guaranteed that $\mathcal{F}_{\text{ccnn}} \supseteq \mathcal{F}_{\text{cnn}}$.

We propose to minimize the empirical risk (4) over $\mathcal{F}_{\text{ccnn}}$ instead of $\mathcal{F}_{\text{cnn}}$; doing so defines a convex optimization problem over this richer class of functions

$$\widehat{f}_{\text{ccnn}} := \arg\min_{f^A\in\mathcal{F}_{\text{ccnn}}} \sum_{i=1}^{n} \mathcal{L}(f^A(x_i); y_i). \tag{9}$$

In Section 3.3, we describe iterative algorithms that can be used to solve this convex problem in the more general setting of nonlinear activation functions.

3.2. Nonlinear activations: RKHS filters

For nonlinear activation functions σ, we relax the class of CNN filters to a reproducing kernel Hilbert space (RKHS). As we will show, this relaxation allows us to reduce the problem to the linear activation case.

Let $\mathcal{K} : \mathbb{R}^{d_1} \times \mathbb{R}^{d_1} \to \mathbb{R}$ be a positive semidefinite kernel function. For particular choices of kernels (e.g., the Gaus-

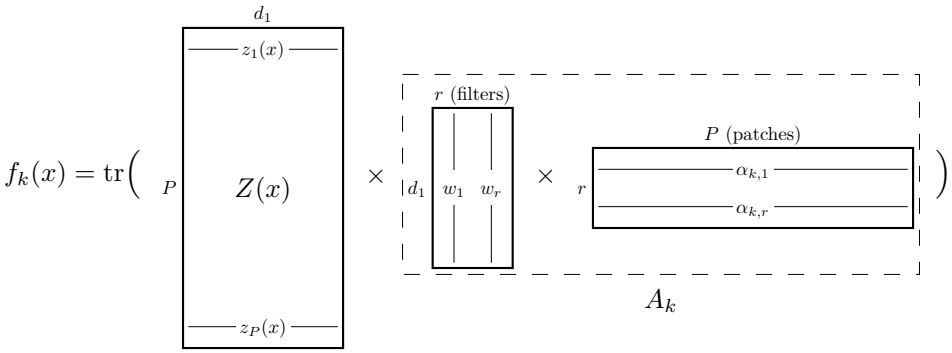

Figure 1. The k^{th} output of a CNN $f_k(x) \in \mathbb{R}$ can be expressed as the product between a matrix $Z(x) \in \mathbb{R}^{P \times d_1}$ whose rows are features of the input patches and a rank-r matrix $A_k \in \mathbb{R}^{d_1 \times P}$, which is made up of the filter weights $\{w_j\}$ and coefficients $\{a_{k,j,p}\}$, as illustrated. Due to the parameter sharing intrinsic to CNNs, the matrix A_k inherits a low rank structure, which can be encouraged via convex relaxation using the nuclear norm.

sian RBF kernel) and a sufficiently smooth activation function σ, we are able to show that the filter $h : z \mapsto \sigma(\langle w, z\rangle)$ is contained in the RKHS induced by the kernel function $\mathcal{K}$. See Section 3.4 for the choice of the kernel function and the activation function. Let $S := \{z_p(x_i) : p \in [P], i \in [n]\}$ be the set of patches in the training dataset. The representer theorem then implies that for any patch $z_p(x_i) \in S$, the function value can be represented by

$$h(z_p(x_i)) = \sum_{(i',p') \in [n] \times [P]} c_{i',p'} \mathcal{K}(z_p(x_i), z_{p'}(x_{i'})) \quad (10)$$

for some coefficients $\{c_{i',p'}\}_{(i',p') \in [n] \times [P]}$. Filters of the form (10) are members of the RKHS, because they are linear combinations of basis functions $z \mapsto \mathcal{K}(z, z_{p'}(x_{i'}))$. Such filters are parametrized by a finite set of coefficients $\{c_{i',p'}\}_{(i',p') \in [n] \times [P]}$, which can be estimated via empirical risk minimization.

Let $K \in \mathbb{R}^{nP \times nP}$ be the symmetric kernel matrix, where with rows and columns indexed by the example-patch index pair $(i,p) \in [n] \times [P]$. The entry at row (i,p) and column (i',p') of matrix K is equal to $\mathcal{K}(z_p(x_i), z_{p'}(x_{i'}))$. So as to avoid re-deriving everything in the kernelized setting, we perform a reduction to the linear setting of Section 3.1. Consider a factorization $K = QQ^\top$ of the kernel matrix, where $Q \in \mathbb{R}^{nP \times m}$; one example is the Cholesky factorization with $m = nP$. We can interpret each row $Q_{(i,p)} \in \mathbb{R}^m$ as a feature vector in place of the original $z_p(x_i) \in \mathbb{R}^{d_1}$, and rewrite equation (10) as

$$h(z_p(x_i)) = \langle Q_{(i,p)}, w \rangle \quad \text{where} \quad w := \sum_{(i',p')} c_{i',p'} Q_{(i',p')}.$$

In order to learn the filter h, it suffices to learn the m-dimensional vector w. To do this, define patch matrices $Z(x_i) \in \mathbb{R}^{P \times m}$ for each $i \in [n]$ so that its p-th row is $Q_{(i,p)}$. Then the problem reduces to learning a linear filter with coefficient vector w. Carrying out all of Section 3.1, solving the ERM gives us a parameter matrix $A \in \mathbb{R}^{m \times Pd_2}$. The only difference is that ℓ_2-norm constraint (C1) needs to be adapted to the norm of the RKHS. See Appendix B for details.

At test time, given a new input $x \in \mathbb{R}^{d_0}$, we can compute a patch matrix $Z(x) \in \mathbb{R}^{P \times m}$ as follows:

- The p-th row of this matrix is the feature vector for patch p, which is equal to $Q^\dagger v(z_p(x)) \in \mathbb{R}^m$, where for any patch z, the vector $v(z)$ is defined as a nP-dimensional vector whose (i,p)-th coordinate is equal to $\mathcal{K}(z, z_p(x_i))$. We note that if x is an instance x_i in the training set, then the vector $Q^\dagger v(z_p(x))$ is exactly equal to $Q_{(i,p)}$. Thus the mapping $Z(x)$ applies to both training and testing.
- We can then compute the predictor $f_k(x) = \mathrm{tr}(Z(x)A_k)$ via equation (6). Note that we do not explicitly need to compute the filter values $h_j(z_p(x))$ to compute the output under the CCNN.

Retrieving filters. When we learn multi-layer CCNNs (Section 4), we need to compute the filters h_j explicitly in order to form the inputs to the next layer. Recall from Section 3.1 that the column space of matrix A corresponds to parameters of the convolutional layer, and the row space of A corresponds to parameters of the output layer. Thus, once we obtain the parameter matrix A, we compute a rank-r approximation $A \approx \widehat{U}\widehat{V}^\top$. Then set the j-th filter h_j to the mapping

$$z \mapsto \langle \widehat{U}_j, Q^\dagger v(z) \rangle \quad \text{for any patch } z \in \mathbb{R}^{d_1}, \quad (11)$$

where $\widehat{U}_j \in \mathbb{R}^m$ is the j-th column of matrix $\widehat{U}$, and $Q^\dagger v(z)$ represents the feature vector for patch z.[2] The matrix $\widehat{V}^\top$ encodes parameters of the output layer, thus

[2] If z is a patch in the training set, namely $z = z_p(x_i)$, then we have equation $Q^\dagger v(z) = Q_{(i,p)}$

Algorithm 1 Learning two-layer CCNNs

Input: Data $\{(x_i, y_i)\}_{i=1}^n$, kernel function $\mathcal{K}$, regularization parameter $R > 0$, number of filters r.

1. Construct a kernel matrix $K \in \mathbb{R}^{nP \times nP}$ such that the entry at column (i, p) and row (i', p') is equal to $\mathcal{K}(z_p(x_i), z_{p'}(x_{i'}))$. Compute a factorization $K = QQ^\top$ or an approximation $K \approx QQ^\top$, where $Q \in \mathbb{R}^{nP \times m}$.
2. For each x_i, construct patch matrix $Z(x_i) \in \mathbb{R}^{P \times m}$ whose p-th row is the (i, p)-th row of Q, where $Z(\cdot)$ is defined in Section 3.2.
3. Solve the following optimization problem to obtain a matrix $\widehat{A} = (\widehat{A}_1, \ldots, \widehat{A}_{d_2})$:

$$\widehat{A} \in \operatorname*{argmin}_{\|A\|_* \le R} \widetilde{\mathcal{L}}(A) \text{ where} \tag{12}$$

$$\widetilde{\mathcal{L}}(A) := \sum_{i=1}^n \mathcal{L}\Big(\big(\operatorname{tr}(Z(x_i)A_1), \ldots, \operatorname{tr}(Z(x_i)A_{d_2})\big); y_i\Big).$$

4. Compute a rank-r approximation $\widehat{U}\widehat{V}^\top \approx \widehat{A}$ where $\widehat{U} \in \mathbb{R}^{m \times r}$ and $\widehat{V} \in \mathbb{R}^{Pd_2 \times r}$.

Output: predictor $\widehat{f}_{\text{ccnn}}(x) := \big(\operatorname{tr}(Z(x)\widehat{A}_1), \ldots, \operatorname{tr}(Z(x)\widehat{A}_{d_2})\big)$ and the convolutional layer output $H(x) := \widehat{U}^\top Z(x)^\top$.

doesn't appear in the filter expression (11). It is important to note that the filter retrieval is not unique, because the rank-r approximation of the matrix A is not unique. The heuristic we suggest is to form the singular value decomposition $A = U\Lambda V^\top$, then define $\widehat{U}$ to be the first r columns of U.

When we apply all of the r filters to all patches of an input $x \in \mathbb{R}^{d_0}$, the resulting output is $H(x) := \widehat{U}^\top Z(x)^\top$ — this is an $r \times P$ matrix whose element at row j and column p is equal to $h_j(z_p(x))$.

3.3. Algorithm

The algorithm for learning a two-layer CCNN is summarized in Algorithm 1; it is a formalization of the steps described in Section 3.2. In order to solve the optimization problem (12), the simplest approach is via projected gradient descent: At iteration t, using a step size $\eta^t > 0$, we form the new matrix A^{t+1} based on the previous iterate A^t according to:

$$A^{t+1} = \Pi_R\Big(A^t - \eta^t \nabla_A \widetilde{\mathcal{L}}(A^t)\Big). \tag{13}$$

Here $\nabla_A \widetilde{\mathcal{L}}$ denotes the gradient of the objective function defined in (12), and Π_R denotes the Euclidean projection onto the nuclear norm ball $\{A : \|A\|_* \le R\}$. This nuclear norm projection can be obtained by first computing the singular value decomposition of A, and then projecting the vector of singular values onto the ℓ_1-ball. This latter projection step can be carried out efficiently by the algorithm of Duchi et al. (2008). There are other efficient optimization algorithms (Duchi et al., 2011; Xiao & Zhang, 2014) for solving the problem (12). All these algorithms can be executed in a stochastic fashion, so that each gradient step processes a mini-batch of examples.

The computational complexity of each iteration depends on the width m of the matrix Q. Setting $m = nP$ allows us to solve the exact kernelized problem, but to improve the computation efficiency, we can use Nyström approximation (Drineas & Mahoney, 2005) or random feature approximation (Rahimi & Recht, 2007); both are randomized methods to obtain a tall-and-thin matrix $Q \in \mathbb{R}^{nP \times m}$ such that $K \approx QQ^\top$. Typically, the parameter m is chosen to be much smaller than nP. In order to compute the matrix Q, the Nyström approximation method takes $\mathcal{O}(m^2 nP)$ time. The random feature approximation takes $\mathcal{O}(mnPd_1)$ time, but can be improved to $\mathcal{O}(mnP \log d_1)$ time using the fast Hadamard transform (Le et al., 2013). The time complexity of project gradient descent also scales with m rather than with nP.

3.4. Theoretical results

In this section, we upper bound the generalization error of Algorithm 1, proving that it converges to the best possible generalization error of CNN. We focus on the binary classification case where the output dimension is $d_2 = 1$.[3]

The learning of CCNN requires a kernel function $\mathcal{K}$. We consider kernel functions whose associated RKHS is large enough to contain any function of the following form: $z \mapsto q(\langle w, z\rangle)$, where q is an arbitrary polynomial function and $w \in \mathbb{R}^{d_1}$ is an arbitrary vector. As a concrete example, we consider the inverse polynomial kernel:

$$\mathcal{K}(z, z') := \frac{1}{2 - \langle z,\, z'\rangle}, \quad \|z\|_2 \le 1, \|z'\|_2 \le 1. \tag{14}$$

This kernel was studied by Shalev-Shwartz et al. (2011) for learning halfspaces, and by Zhang et al. (2016a) for learning fully-connected neural networks. We also consider the Gaussian RBF kernel:

$$\mathcal{K}(z, z') := e^{-\gamma\|z - z'\|_2^2}, \quad \|z\|_2 = \|z'\|_2 = 1. \tag{15}$$

As shown by Appendix A, the inverse polynomial kernel and the Gaussian kernel satisfy the above notion of richness. We focus on these two kernels for the theoretical analysis.

Let $\widehat{f}_{\text{ccnn}}$ be the CCNN that minimizes the empirical risk (12) using one of the two kernels above. Our main theoretical result is that for suitably chosen activation functions, the generalization error of $\widehat{f}_{\text{ccnn}}$ is comparable to that of the best CNN model. In particular, we consider the following types of activation functions σ:

[3] We can treat the multiclass case by performing a standard one-versus-all reduction to the binary case.

(a) arbitrary polynomial functions (e.g., used by Chen & Manning (2014); Livni et al. (2014)).

(b) sinusoid activation function $\sigma(t) := \sin(t)$ (e.g., used by Sopena et al. (1999); Isa et al. (2010)).

(c) erf function $\sigma_{\text{erf}}(t) := 2/\sqrt{\pi}\int_0^t e^{-z^2}dz$, which represents a close approximation to the sigmoid function (Zhang et al., 2016a).

(d) a smoothed hinge loss $\sigma_{\text{sh}}(t) := \int_{-\infty}^t \frac{1}{2}(\sigma_{\text{erf}}(z)+1)dz$, which represents a close approximation to the ReLU function (Zhang et al., 2016a).

To understand how these activation functions pair with our choice of kernels, we consider polynomial expansions of the above activation functions: $\sigma(t) = \sum_{j=0}^{\infty} a_j t^j$, and note that the smoothness of these functions are characterized by the rate of their coefficients $\{a_j\}_{j=0}^{\infty}$ converging to zero. If σ is a polynomial in category (a), then the richness of the RKHS guarantees that it contains the class of filters activated by function σ. If σ is a non-polynomial function in categories (b),(c),(d), then as Appendix A shows, the RKHS contains the filter only if the coefficients $\{a_j\}_{j=0}^{\infty}$ converge quickly enough to zero (the criterion depends on the concrete choice of the kernel). Concretely, the inverse polynomial kernel is shown to capture all of the four categories of activations: thus, (a), (b), (c), and (d) are all are referred as *valid activation functions* for the inverse polynomial kernel. The Gaussian kernel induces a smaller RKHS, so only (a) and (b) are *valid activation functions* for the Gaussian kernel. In contrast, the sigmoid function and the ReLU function are not valid for either kernel, because their polynomial expansions fail to converge quickly enough, or more intuitively speaking, because they are not smooth enough to be contained in the RKHS.

We are ready to state the main theoretical result. In the theorem statement, we use $K(X) \in \mathbb{R}^{P\times P}$ to denote the random kernel matrix obtained from an input vector $X \in \mathbb{R}^{d_0}$ drawn randomly from the population. More precisely, the (p,q)-th entry of $K(X)$ is given by $\mathcal{K}(z_p(X), z_q(X))$.

Theorem 1. *Assume that the loss function $\mathcal{L}(\cdot;y)$ is L-Lipchitz continuous for every $y \in [d_2]$ and that $\mathcal{K}$ is the inverse polynomial kernel or the Gaussian kernel. For any valid activation function σ, there is a constant $C_\sigma(B_1)$ such that by choosing hyper-parameter $R := C_\sigma(B_1)B_2 r$ in Algorithm 1, the expected generalization error is at most*

$$\mathbb{E}_{X,Y}[\mathcal{L}(\widehat{f}_{ccnn}(X);Y)] \le \inf_{f\in\mathcal{F}_{cnn}} \mathbb{E}_{X,Y}[\mathcal{L}(f(X);Y)] + \frac{c\, LC_\sigma(B_1)B_2 r\sqrt{\log(nP)\,\mathbb{E}_X[\|K(X)\|_2]}}{\sqrt{n}}, \quad (16)$$

where $c > 0$ is a universal constant.

Proof sketch The proof of Theorem 1 consists of two parts: First, we consider a larger function class that contains the class of CNNs. This function class is defined as:

$$\mathcal{F}_{\text{ccnn}} := \Big\{ x \mapsto \sum_{j=1}^{r^*}\sum_{p=1}^{P} \alpha_{j,p} h_j(z_p(x)) : \; r^* < \infty \quad (17)$$

$$\text{and} \sum_{j=1}^{r^*} \|\alpha_j\|_2 \|h_j\|_{\mathcal{H}} \le C_\sigma(B_1)B_2 d_2 \Big\}. \quad (18)$$

where $\|\cdot\|_{\mathcal{H}}$ is the norm of the RKHS associated with the kernel. This new function class relaxes the class of CNNs in two ways: 1) the filters are relaxed to belong to the RKHS, and 2) the ℓ_2-norm bounds on the weight vectors are replaced by a single constraint on $\|\alpha_j\|_2$ and $\|h_j\|_{\mathcal{H}}$. We prove the following property for the predictor $\widehat{f}_{\text{ccnn}}$: it must be an empirical risk minimizer of $\mathcal{F}_{\text{ccnn}}$. This property holds even though equation (18) defines a non-parametric function class $\mathcal{F}_{\text{ccnn}}$, while Algorithm 1 optimizes $\widehat{f}_{\text{ccnn}}$ in a parametric function class.

Second, we characterize the Rademacher complexity of this new function class $\mathcal{F}_{\text{ccnn}}$, proving an upper bound for it based on the matrix concentration theory. Combining this bound with the classical Rademacher complexity theory (Bartlett & Mendelson, 2003), we conclude that the generalization loss of $\widehat{f}_{\text{ccnn}}$ converges to the least possible generalization error of $\mathcal{F}_{\text{ccnn}}$. The latter loss is bounded by the generalization loss of CNNs (because $\mathcal{F}_{\text{cnn}} \subseteq \mathcal{F}_{\text{ccnn}}$), which establishes the theorem. See the full version of this paper (Zhang et al., 2016b) for a rigorous proof of Theorem 1. ■

Remark on activation functions. It is worth noting that the quantity $C_\sigma(B_1)$ depends on the activation function σ, and more precisely, depends on the convergence rate of the polynomial expansion of σ. Appendix A shows that if σ is a polynomial function of degree ℓ, then $C_\sigma(B_1) = \mathcal{O}(B_1^\ell)$. If σ is the sinusoid function, the erf function or the smoothed hinge loss, then the quantity $C_\sigma(B_1)$ will be exponential in B_1. From an algorithmic perspective, we don't need to know the activation function for executing Algorithm 1. From a theoretical perspective, however, the choice of σ is relevant from the point of Theorem 1 to compare $\widehat{f}_{\text{ccnn}}$ with the best CNN, whose representation power is characterized by the choice of σ. Therefore, if a CNN with a low-degree polynomial σ performs well on a given task, then CCNN also enjoys correspondingly strong generalization. Empirically, this is actually borne out: in Section 5, we show that the quadratic activation function performs almost as well as the ReLU function for digit classification.

Remark on parameter sharing. In order to demonstrate the importance of parameter sharing, consider a CNN without parameter sharing, so that we have filter weights $w_{j,p}$ for each filter index j and patch index p. With this change,

the new CNN output (2) is

$$f(x) = \sum_{j=1}^{r} \sum_{p=1}^{P} \alpha_{j,p} \sigma(w_{j,p}^\top z_p(x)),$$

where $\alpha_{j,p} \in \mathbb{R}$ and $w_{j,p} \in \mathbb{R}^{d_1}$. Note that the hidden layer of this new network has P times more parameters than that of the convolutional neural network with parameter sharing. These networks without parameter sharing can be learned by the recursive kernel method proposed by Zhang et al. (2016a). Their paper shows that under the norm constraints $\|w_j\|_2 \le B_1'$ and $\sum_{j=1}^r \sum_{p=1}^P |\alpha_{j,p}| \le B_2'$, the excess risk of the recursive kernel method is at most $\mathcal{O}(LC_\sigma(B_1')B_2'\sqrt{K_{\max}/n})$, where $K_{\max} = \max_{z:\|z\|_2 \le 1} \mathcal{K}(z,z)$ is the maximal value of the kernel function. Plugging in the norm constraints of the function class $\mathcal{F}_{\text{cnn}}$, we have $B_1' = B_1$ and $B_2' = B_2 r\sqrt{P}$. Thus, the expected risk of the estimated $\widehat{f}$ is bounded by:

$$\mathbb{E}_{X,Y}[\mathcal{L}(\widehat{f}(X);Y)] \le \inf_{f \in \mathcal{F}_{\text{cnn}}} \mathbb{E}_{X,Y}[\mathcal{L}(f(X);Y)] + \frac{c\, LC_\sigma(B_1)B_2 r\sqrt{PK_{\max}}}{\sqrt{n}}. \quad (19)$$

Comparing this bound to Theorem 1, we see that (apart from the logarithmic terms) they differ in the multiplicative factors of $\sqrt{P\,K_{\max}}$ versus $\sqrt{\mathbb{E}[\|K(X)\|_2]}$. Since the matrix $K(X)$ is P-dimensional, we have

$$\|K(X)\|_2 \le \max_{p\in[P]} \sum_{q\in[P]} |\mathcal{K}(z_p(X), z_q(X))| \le P\,K_{\max}.$$

This demonstrates that $\sqrt{P\,K_{\max}}$ is always greater than $\sqrt{\mathbb{E}[\|K(X)\|_2]}$. In general, the first term can be up to factor of $\sqrt{P}$ times greater, which implies that the sample complexity of the recursive kernel method is up to P times greater than that of the CCNN. This difference is intuitive given that the recursive kernel method learns a model with P times more parameters. Although comparing the upper bounds doesn't rigorously show that one method is better than the other, it gives intuition for understanding the importance of parameter sharing.

4. Learning multi-layer CCNNs

In this section, we describe a heuristic method for learning CNNs with more layers. The idea is to estimate the parameters of the convolutional layers incrementally from bottom to top. Before presenting the multi-layer algorithm, we present two extensions, average pooling and multi-channel inputs.

Average pooling. Average pooling is a technique to reduce the output dimension of the convolutional layer from dimensions $P \times r$ to dimensions $P' \times r$ with $P' < P$. For the CCNN model, if we apply average pooling after the convolutional layer, then the k-th output of the CCNN model becomes $\text{tr}(GZ(x)A_k)$ where $G \in \mathbb{R}^{P' \times P}$ is the pooling matrix. Thus, performing a pooling operation requires only replacing every matrix $Z(x_i)$ in problem (12) by the pooled matrix $GZ(x_i)$. Note that the linearity of the CCNN allows us to effectively pool before convolution, even though for the CNN, pooling must be done after applying the nonlinear filters. The resulting ERM problem is still convex, and the number of parameters have been reduced by P/P'-fold.

Algorithm 2 Learning multi-layer CCNNs

Input: Data $\{(x_i, y_i)\}_{i=1}^n$, kernel function $\mathcal{K}$, number of layers m, regularization parameters $R_1, \dots, R_m$, number of filters $r_1, \dots, r_m$.
Define $H_1(x) = x$. For each layer $s = 2, \dots, m$:
- Train a two-layer network by Algorithm 1, taking $\{(H_{s-1}(x_i), y_i)\}_{i=1}^n$ as training examples and R_s, r_s as parameters. Let H_s be the output of the convolutional layer and $\widehat{f}_s$ be the predictor.

Output: Predictor $\widehat{f}_m$ and the top layer output H_m.

Processing multi-channel inputs. If our input has C channels (corresponding to RGB colors, for example), then the input becomes a matrix $x \in \mathbb{R}^{C \times d_0}$. The c-th row of matrix x, denoted by $x[c] \in \mathbb{R}^{d_0}$, is a vector representing the c-th channel. We define the multi-channel patch vector as a concatenation of patch vectors for each channel:

$$z_p(x) := (z_p(x[1]), \dots, z_p(x[C])) \in \mathbb{R}^{Cd_1}.$$

Then we construct the feature matrix $Z(x)$ using the concatenated patch vectors $\{z_p(x)\}_{p=1}^P$. From here, everything else of Algorithm 1 remains the same. We note that this approach learns a convex relaxation of filters taking the form $\sigma(\sum_{c=1}^C \langle w_c, z_p(x[c])\rangle)$, parametrized by the vectors $\{w_c\}_{c=1}^C$.

Multi-layer CCNN. Given these extensions, we are ready to present the algorithm for learning multi-layer CCNNs, summarized in Algorithm 2. For each layer s, we call Algorithm 1 using the output of previous convolutional layers as input—note that this consists of r channels (one from each previous filter); thus we must use the multi-channel extension. Algorithm 2 outputs a new convolutional layer along with a prediction function, which is kept only at the last layer. We optionally use averaging pooling after each successive layer. to reduce the output dimension of the convolutional layers.

5. Experiments

In this section, we compare the CCNN approach with other methods on the MNIST dataset and more challenging variations (VariationsMNIST), including adding white noise (`rand`), random rotation (`rot`), random image back-

	basic	rand	rot	img	img+rot
SVM$_{rbf}$ (Vincent et al., 2010)	3.03%	14.58%	**11.11%**	22.61%	55.18%
NN-1 (Vincent et al., 2010)	4.69%	20.04%	18.11%	27.41%	62.16%
CNN-1 (ReLU)	3.37%	9.83%	18.84%	14.23%	45.96%
CCNN-1	**2.35%**	**8.13%**	13.45%	**10.33%**	**42.90%**
TIRBM (Sohn & Lee, 2012)	-	-	**4.20%**	-	35.50%
SDAE-3 (Vincent et al., 2010)	2.84%	10.30%	9.53%	16.68%	43.76%
ScatNet-2 (Bruna & Mallat, 2013)	1.27%	12.30%	7.48%	18.40%	50.48%
PCANet-2 (Chan et al., 2015)	**1.06%**	6.19%	7.37%	10.95%	35.48%
CNN-2 (ReLU)	2.11%	5.64%	8.27%	10.17%	32.43%
CNN-2 (Quad)	1.75%	5.30%	8.83%	11.60%	36.90%
CCNN-2	1.39%	**4.64%**	6.91%	**7.44%**	**30.23%**

Table 1. Classification error on the basic MNIST and its four variations. The best performance within each block is bolded. "ReLU" and "Quad" denote using the ReLU and quadratic activation functions, respectively.

ground (`img`) or combining the last two (`img+rot`). For all datasets, we use 10,000 images for training, 2,000 images for validation and 50,000 images for testing. This 10k/2k/50k partitioning is standard for MNIST variations (VariationsMNIST).

For the CCNN method and the baseline CNN method, we train two-layer and three-layer models respectively. The models with k convolutional layers are denoted by CCNN-k and CNN-k. Each convolutional layer is constructed on 5×5 patches with unit stride, followed by 2×2 average pooling. The first and the second convolutional layers contains 16 and 32 filters, respectively. The loss function is chosen as the 10-class logistic loss. We use Gaussian kernel for the CCNN. The feature matrix $Z(x)$ is constructed via random feature approximation (Rahimi & Recht, 2007) with dimension $m = 500$ for the first convolutional layer and $m = 1000$ for the second. Before training each CCNN layer, we preprocess the input vectors $z_p(x_i)$ using local contrast normalization and ZCA whitening (Coates et al., 2010). The convex optimization problem is solved by projected SGD with mini-batches of size 50. Code and reproducible experiments are available on the CodaLab platform[4].

As a baseline approach, the CNN models are activated by the ReLU function $\sigma(t) = \max\{0, t\}$ or the quadratic function $\sigma(t) = t^2$. We train them using mini-batch SGD. The input images are preprocessed by global contrast normalization and ZCA whitening (Srivastava et al., 2014). We compare our method against several alternative baselines. The CCNN-1 model is compared against an SVM with the Gaussian RBF kernel (SVM$_{rbf}$) and a fully connected neural network with one hidden layer (NN-1). The CCNN-2 model is compared against methods that report the state-of-the-art results on these datasets, including the translation-invariant RBM model (TIRBM) (Sohn & Lee, 2012), the stacked denoising auto-encoder with three hidden layers (SDAE-3) (Vincent et al., 2010), the ScatNet-2 model (Bruna & Mallat, 2013) and the PCANet-2 model (Chan et al., 2015).

Table 1 shows the classification errors on the test set. The models are grouped with respect to the number of layers that they contain. For models with one convolutional layer, the errors of CNN-1 are significantly lower than that of NN-1, highlighting the benefits of local filters and parameter sharing. The CCNN-1 model outperforms CNN-1 on all datasets. For models with two or more hidden layers, the CCNN-2 model outperforms CNN-2 on all datasets, and is competitive against the state-of-the-art. In particular, it achieves the best accuracy on the `rand`, `img` and `img+rot` dataset, and is comparable to the state-of-the-art on the remaining two datasets. Further adding a third convolutional layer doesn't notibly improve the performance on these datasets.

In Section 3.4, we showed that if the activation function σ is a polynomial function, then the CCNN (which does not depend on σ) requires lower sample complexity to match the performance of the best possible CNN using σ. More precisely, if σ is a degree-ℓ polynomial, then $C_\sigma(B)$ in the upper bound will be controlled by $\mathcal{O}(B^\ell)$. This motivates us to study the performance of low-degree polynomial activations. Table 1 shows that the CNN-2 model with a quadratic activation function achieves error rates comparable to that with a ReLU activation: CNN-2 (Quad) outperforms CNN-2 (ReLU) on the `basic` and `rand` datasets, and is only slightly worse on the `rot` and `img` dataset. Since the performance of CCNN matches that of the best possible CNN, the good performance of the quadratic activation in part explains why the CCNN is also good.

Acknowledgements. MJW and YZ were partially supported by the Office of Naval Research Grant DOD ONR-N00014 and the NSF Grant NSF-DMS-1612948. PL and YZ were partially supported by the Microsoft Faculty Fellowship.

[4] http://worksheets.codalab.org/worksheets/0x1468d91a878044fba86a5446f52aacde/

References

Aslan, Özlem, Cheng, Hao, Zhang, Xinhua, and Schuurmans, Dale. Convex two-layer modeling. In *Advances in Neural Information Processing Systems*, pp. 2985–2993, 2013.

Aslan, Özlem, Zhang, Xinhua, and Schuurmans, Dale. Convex deep learning via normalized kernels. In *Advances in Neural Information Processing Systems*, pp. 3275–3283, 2014.

Bartlett, Peter L and Mendelson, Shahar. Rademacher and Gaussian complexities: Risk bounds and structural results. *The Journal of Machine Learning Research*, 3: 463–482, 2003.

Bengio, Yoshua, Roux, Nicolas L, Vincent, Pascal, Delalleau, Olivier, and Marcotte, Patrice. Convex neural networks. In *Advances in Neural Information Processing Systems*, pp. 123–130, 2005.

Blum, Avrim L and Rivest, Ronald L. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1): 117–127, 1992.

Bottou, Léon. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

Bruna, Joan and Mallat, Stéphane. Invariant scattering convolution networks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1872–1886, 2013.

Chan, Tsung-Han, Jia, Kui, Gao, Shenghua, Lu, Jiwen, Zeng, Zinan, and Ma, Yi. Pcanet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing*, 24(12):5017–5032, 2015.

Chen, Danqi and Manning, Christopher D. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pp. 740–750, 2014.

Coates, Adam, Lee, Honglak, and Ng, Andrew Y. An analysis of single-layer networks in unsupervised feature learning. *Ann Arbor*, 1001(48109):2, 2010.

Daniely, Amit, Frostig, Roy, and Singer, Yoram. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. *arXiv preprint arXiv:1602.05897*, 2016.

Drineas, Petros and Mahoney, Michael W. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *The Journal of Machine Learning Research*, 6:2153–2175, 2005.

Duchi, John, Shalev-Shwartz, Shai, Singer, Yoram, and Chandra, Tushar. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 272–279. ACM, 2008.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

Fahlman, Scott E. An empirical study of learning speed in back-propagation networks. *Journal of Heuristics*, 1988.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29 (6):82–97, 2012.

Isa, IS, Saad, Z, Omar, S, Osman, MK, Ahmad, KA, and Sakim, HA Mat. Suitable mlp network activation functions for breast cancer and thyroid disease detection. In *2010 Second International Conference on Computational Intelligence, Modelling and Simulation*, pp. 39–44, 2010.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Lawrence, Steve, Giles, C Lee, Tsoi, Ah Chung, and Back, Andrew D. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

Le, Quoc, Sarlós, Tamás, and Smola, Alex. Fastfood-approximating kernel expansions in loglinear time. In *Proceedings of the International Conference on Machine Learning*, 2013.

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

Livni, Roi, Shalev-Shwartz, Shai, and Shamir, Ohad. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pp. 855–863, 2014.

Mairal, Julien, Koniusz, Piotr, Harchaoui, Zaid, and Schmid, Cordelia. Convolutional kernel networks. In *Advances in Neural Information Processing Systems*, pp. 2627–2635, 2014.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Rahimi, Ali and Recht, Benjamin. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pp. 1177–1184, 2007.

Shalev-Shwartz, Shai, Shamir, Ohad, and Sridharan, Karthik. Learning kernel-based halfspaces with the 0-1 loss. *SIAM Journal on Computing*, 40(6):1623–1646, 2011.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Sohn, Kihyuk and Lee, Honglak. Learning invariant representations with local transformations. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 1311–1318, 2012.

Sopena, Josep M, Romero, Enrique, and Alquezar, Rene. Neural networks with periodic and monotonic activation functions: a comparative study in classification problems. In *ICANN 99*, pp. 323–328, 1999.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

VariationsMNIST. Variations on the MNIST digits. http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations, 2007.

Vincent, Pascal, Larochelle, Hugo, Lajoie, Isabelle, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

Wang, Tao, Wu, David J, Coates, Andrew, and Ng, Andrew Y. End-to-end text recognition with convolutional neural networks. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pp. 3304–3308. IEEE, 2012.

Xiao, Lin and Zhang, Tong. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

Zhang, Yuchen, Lee, Jason D, and Jordan, Michael I. ℓ_1-regularized neural networks are improperly learnable in polynomial time. In *Proceedings on the 33rd International Conference on Machine Learning*, 2016a.

Zhang, Yuchen, Liang, Percy, and Wainwright, Martin J. Convexified convolutional neural networks. *CoRR*, abs/1609.01000, 2016b. URL http://arxiv.org/abs/1609.01000.