Efficient Learning of CNNs using Patch Based Features

Alon Brutzkus* [1] **Amir Globerson*** [1] **Eran Malach*** [2] **Alon Regev Netser*** [2] **Shai Shalev-Shwartz*** [2]

Abstract

Recent work has demonstrated the effectiveness of using patch based representations when learning from image data. Here we provide theoretical support for this observation, by showing that a simple semi-supervised algorithm that uses patch statistics can efficiently learn labels produced by a one-hidden-layer Convolutional Neural Network (CNN). Since CNNs are known to be computationally hard to learn in the worst case, our analysis holds under some distributional assumptions. We show that these assumptions are necessary and sufficient for our results to hold. We verify that the distributional assumptions hold on real-world data by experimenting on the CIFAR-10 dataset, and find that the analyzed algorithm outperforms a vanilla one-hidden-layer CNN. Finally, we demonstrate that by running the algorithm in a layer-by-layer fashion we can build a deep model which gives further improvements, hinting that this method provides insights about the behavior of deep CNNs.

1. Introduction

Recently, learning with patch-based representations has become increasingly popular for solving visual tasks. A notable example is the vision transformer (ViT, Dosovitskiy et al. (2020)) which breaks the input image into 16x16 patches, and then applies a transformer architecture on an embedding of the patches. Some follow-up works show that using simple MLPs (Tolstikhin et al., 2021) or Convolutions (Trockman & Kolter, 2022) on top of the patch-based embedding does similarly well.

In fact, an older work by Coates et al. (2011) already introduced a very simple variant of such patch-based learning that achieved state-of-the-art results on CIFAR-10 when published. In that work, an embedding of the image patches is learned in an unsupervised fashion, and then a linear predictor is trained using the patch-based features. This algorithm has been recently improved by Thiry et al. (2021), showing that it can compete with simple Convolutional networks. While these works provide a simple "baseline" for patch-based learning, their theoretical properties have yet to be explored.

In this work we provide a theoretical analysis of the aforementioned learning algorithm. We show that, under some assumptions, this algorithm finds a predictor with small error when the data is labeled by a shallow CNN. The sample complexity and run-time of the algorithm depend on the covering number of the *distribution of patches*, therefore showing that when the patches lie in a low-dimensional space, the algorithm can achieve low error. Our analysis leads to a new and improved variant of the original algorithm. We analyze this new variant, and show that it performs well assuming that the distribution of patches respects some geometrical properties of the target function.

We complement our theoretical results with a thorough empirical study, comparing the original algorithm of Coates et al. (2011) against our new version and against a standard shallow CNN, showing that our version outperforms both. Finally, we show that our algorithm can scale to deeper models, by running it repeatedly in a layer-by-layer fashion, which improves the performance of the shallow model. We show that the ability to scale to deeper models is a unique property of our approach, while the original algorithm does not improve when applied layerwise.

2. Related Work

Learning with Data-Dependent Representation. Learning linear classifiers over fixed representations of the input is a well-studied approach. Specifically, kernel methods (Shawe-Taylor et al., 2004) and random features (Rahimi et al., 2007; Rahimi & Recht, 2008) are some prominent examples of such methods. Recently, the study of the kernel derived from the neural network architecture, known as the Neural Tangent Kernel (NTK), has become a main theme in many theoretical works (Du et al., 2019; 2018b; Arora

*Authors ordered alphabetically. [1]Blavatnik School of Computer Science, Tel Aviv University, Israel [2]School of Computer Science, The Hebrew University of Jerusalem, Israel. Correspondence to: Eran Malach <eran.malach@mail.huji.ac.il>, Alon Regev Netser <alon.netser@mail.huji.ac.il>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

et al., 2019; Ji & Telgarsky, 2019b; Cao & Gu, 2019; Jacot et al., 2018). However, using a fixed representation is known to have its limitations, due to the fact that the representation cannot adapt to the learning task (see for example Kamath et al. (2020); Daniely & Malach (2020); Malach et al. (2021)). Therefore, using representations that depend on properties of the input data instead of fixing representations in advance seems like a sensible way to improve performance. Indeed, such data-dependent representations have been explored for learning over image data. Specifically, as previously noted, Coates et al. (2011) study a learning algorithm for images that uses a linear classifier over a data-dependent representation. A more recent work by Thiry et al. (2021) suggests a similar patch-based representation, and shows that it achieves impressive empirical performance on complex datasets such as CIFAR-10 and ImageNet. However, both Coates et al. (2011) and Thiry et al. (2021) focus on empirical evaluation of the algorithm, while our work gives a theoretical analysis of the algorithm.

Learning Neural Networks under Distributional Assumptions. Efficient learning of neural networks without any distributional assumptions is known to be hard (e.g., Klivans & Sherstov, 2009; Livni et al., 2014). In fact, in some cases learning neural networks under simple distributions such as Gaussian or Uniform distributions was shown to be hard for a large family of algorithms (see Diakonikolas et al., 2020). This fact motivates finding the "right" distributional assumptions, that are both realistic and allow efficient learning of neural networks. To this end, various works have studied learning of feed-forward networks (Ge et al., 2018; Awasthi et al., 2021) and convolutional networks under different distributional assumptions (Brutzkus & Globerson, 2017; Oymak & Soltanolkotabi, 2018; Du & Goel, 2018; Malach & Shalev-Shwartz, 2018; Brutzkus & Globerson, 2020; Du et al., 2018a). However, these assumptions often seem far from capturing natural data distributions. Our work gives a simple characterization of data distributions that both seems reasonable for natural data, and allows efficient learning of convolutional networks.

Several works study theoretical properties of neural networks under a low-dimensional assumption on the data. Cloninger & Klock (2021); Basri & Jacobs (2017) give approximation guarantees for neural networks given that the data is supported on a low-dimensional manifold. Under similar assumptions Chen et al. (2019) and Schmidt-Hieber (2019) give statistical guarantees for neural networks that are ERM minimizers. Note that they do not provide optimization guarantees nor study the inductive bias of optimizers used in practice. Ghorbani et al. (2021) show that neural networks can learn distributions that are labeled by one-layer networks and have a specific low dimensional structure. Goldt et al. (2020) study neural networks under a hidden manifold model and provide asymptotic guarantees. Two recent works show that under the NTK approximation, neural networks can classify one-dimensional curves (Buchanan et al., 2020; Wang et al., 2021). HaoChen et al. (2021) study self-supervised learning under a low-dimensional setting. Assuming that the input data has low intrinsic dimension is a natural assumption, that is also related to the assumption we make in this work. However, unlike previous works, we assume that the *patches* of the image, and not the entire input, lie in a low-dimensional space, an assumption that seems more likely to be satisfied by natural data.

3. Preliminaries

We begin by describing the problem setting considered in the paper. First, we introduce our assumptions on the data distribution and the labeling function. We then describe the learning algorithm that we analyze.

3.1. Data Generating Distribution

We consider distributions over images, where each image is of size d_I. So, let $\mathcal{X} \subseteq \mathbb{R}^{d_I}$ be the input space and $\mathcal{Y} = \{\pm 1\}$ be the label space. Each image $\mathbf{x} \in \mathbb{R}^{d_I}$ contains n patches, i.e. n sub-images, where each sub-image is of size $d_P \geq 2$. We identify each of the n patches in the image with a sequence of indices. That is, we define n sequences $A_1, ..., A_n$ such that for every j, $A_j = (i_1, \ldots, i_{d_P})$ is a sequence of indices satisfying $\{i_1, \ldots, i_{d_P}\} \subseteq [d_I]$ (where we denote $[N] := \{1, \ldots, N\}$). So, A_j defines the set of indices of image pixels associated with the j-th patch, and we denote the j-th patch by $\mathbf{x}[j] := \left(\mathbf{x}_{A_j(1)}, \ldots, \mathbf{x}_{A_j(d_P)}\right)$.

In our theoretical study we make the simplifying assumption that the patches do not overlap, meaning that the sets $A_1, \ldots, A_n$ are disjoint. However, we believe our results can be easily extended to the case of overlapping patches. In our empirical study we use overlapping patches, as in Coates et al. (2011); Thiry et al. (2021).

We now introduce our assumption on the labeling function of the data distribution. Namely, we assume that the data is labeled by a Shallow Convolutional Neural Network (CNN):

Definition 3.1. Let σ be the ReLU activation: $\sigma(x) = \max\{x, 0\}$. A Shallow CNN is any function of the form:

$$F_{\mathbf{W},\mathbf{U}}(\mathbf{x}) = \sum_{i=1}^{n} \left\langle \mathbf{u}^{(i)}, \sigma(\mathbf{W}\mathbf{x}[i]) \right\rangle \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{l \times d_P}$, $\mathbf{U} = \left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}\right) \in \mathbb{R}^{l \times n}$.

Observe that this is a standard ReLU CNN with one-hidden-layer (containing l output channels), followed by a linear "readout" layer. Since we consider classification tasks, this is the natural choice for a shallow network architecture.[1]

[1]For simplicity, we study networks without bias, but our results

For our analysis, we need the following measure of Lipschitz continuity with respect to the input patches:

Definition 3.2. Given a CNN $F_{\mathbf{W},\mathbf{U}}$, we say that $F_{\mathbf{W},\mathbf{U}}$ is L-Patch-Lipschitz if for every $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ it holds that:

$$|F_{\mathbf{W},\mathbf{U}}(\mathbf{x}) - F_{\mathbf{W},\mathbf{U}}(\mathbf{x}')| \leq L \cdot \max_{i \in [n]} \|\mathbf{x}[i] - \mathbf{x}'[i]\|$$

That is, the L-Patch-Lipschitz definition measures how sensitive the CNN is to perturbation of the input patches. Note that since the activation σ is Lipschitz continuous, any shallow CNN is L-Patch-Lipschitz:

Lemma 3.3. *Let $F_{\mathbf{W},\mathbf{U}}$ be some CNN. Then, $F_{\mathbf{W},\mathbf{U}}$ is L-Patch-Lipschitz, with $L \leq \|\mathbf{W}\|_2 \sum_{i=1}^{n} \left\|\mathbf{u}^{(i)}\right\|$.*

Please refer to Appendix B.1 for the proof of this lemma.

Let $\mathcal{I}$ be some distribution over $\mathcal{X} \times \mathcal{Y}$, i.e. a distribution of labeled examples. We make the following assumption on $\mathcal{I}$.

Assumption 3.4. There exists some L-Patch-Lipschitz CNN $F_{\mathbf{W},\mathbf{U}}$ such that $\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}}\left[yF_{\mathbf{W},\mathbf{U}}(\mathbf{x}) \geq 1\right] = 1$.

Simply put, we assume that $\mathcal{I}$ is realizable with margin 1 by a shallow CNN. From Lemma 3.3, any shallow CNN is L-Patch-Lipschitz. We use this term in the definition only in order to track the value of L (which plays a role in our analysis). It does not limit the scope of our results.

Computational Hardness. Observe that while Assumption 3.4 restricts the family of distributions that are considered in the paper, learning under Assumption 3.4 alone is computationally intractable. Indeed, learning one-hidden-layer networks is known to be computationally hard, under various cryptographic assumptions (Klivans & Sherstov, 2006; Song et al., 2021; Daniely & Shalev-Shwartz, 2016; Daniely & Vardi, 2020; 2021). From these results it follows that shallow CNNs *cannot* be learned in run-time that is *polynomial* in the patch dimension d_P. In our results, we show learnability with run-time that depends on the covering number of the distribution of patches. In the worst case, our results translate to learning in run-time that is *exponential* in the patch dimension d_P.

3.2. Covering the Patch Distributions

Let $\mathcal{I}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. We assume that the marginal distribution of $\mathcal{I}$ on $\mathcal{X}$ has a density function and we denote its support by $\mathrm{supp}(\mathcal{I})$. We denote by $P_{\mathcal{I}}$ the set of patches supported by the distribution $\mathcal{I}$. Namely, $P_{\mathcal{I}} = \{\mathbf{x}[i] \mid \mathbf{x} \in \mathrm{supp}(\mathcal{I}),\, 1 \leq i \leq n\}$.

The sample complexity and run-time of our algorithm depend on the *covering* of the distribution of patches. We define the covering with respect to some metric dist, where we denote by $\mathcal{B}_{\mathrm{dist}}(\mathbf{v}, r)$ the ball of radius r around the point $\mathbf{v}$, namely: $\mathcal{B}_{\mathrm{dist}}(\mathbf{v}, r) = \{\mathbf{u} \,:\, \mathrm{dist}(\mathbf{v}, \mathbf{u}) \leq r\}$.

Definition 3.5. For a set A, we say that C is an *r-covering* of A if $A \subseteq \cup_{\mathbf{v}\in C} \mathcal{B}_{\mathrm{dist}}(\mathbf{v}, r)$. The *$r$-covering number* of a set A, denoted by $N_{\mathrm{dist}}(A, r)$, is the minimal size of an r-covering of A.

Specifically, taking dist to be ℓ_2 (the Euclidean norm), we denote $N_{\ell_2}(A, r)$ the r-covering number with respect to ℓ_2. We note that the covering number is a well-known technical tool (see e.g. Vershynin (2017)), used frequently across many fields such as learning theory, statistics and more (e.g., Haussler (1995); Bartlett et al. (1997); Zhou (2002)). Here, we use the covering number to measure the effective dimension of the patch distribution (see Section 4.1.1 for details about the relation to intrinsic dimension). Our main result (Theorem 4.1) implies efficient learnability (i.e., polynomial sample-complexity and run-time) when the effective dimension of the patches distribution is low (i.e., polynomial covering number).

3.3. Learning Algorithm

We now describe the learning algorithm analyzed in this work. The algorithm has two stages. In the first stage, which is unsupervised, a dictionary of patches is learned by clustering patches from an *unlabeled* set of examples. In the second stage, which is supervised, a linear classifier is learned over a feature-map generated using the patch dictionary. We begin by describing the patch embedding feature-map, and continue with a detailed description of the learning algorithm.

Patch-based image embedding Suppose that $D = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\} \subseteq \mathbb{R}^{d_P}$ is a set of patches. We refer to D as a *dictionary*. We next describe how to use a dictionary to obtain a representation of a given image $\mathbf{x}$. For some "query" patch $\mathbf{z} \in \mathbb{R}^{d_P}$, denote by $\text{k-}\pi_{\mathrm{dist}}(\mathbf{z}; D) \subseteq D$ the set of k patches closest to $\mathbf{z}$. Namely, $\text{k-}\pi_{\mathrm{dist}}(\mathbf{z}; D)$ is a subset of size k s.t. for all $\mathbf{v}, \mathbf{v}' \in D$ satisfying $\mathbf{v} \in \text{k-}\pi_{\mathrm{dist}}(\mathbf{z}; D)$ and $\mathbf{v}' \notin \text{k-}\pi_{\mathrm{dist}}(\mathbf{z}; D)$ it holds that $\mathrm{dist}(\mathbf{z}, \mathbf{v}) \leq \mathrm{dist}(\mathbf{z}, \mathbf{v}')$. [2]

Using the *dictionary* D we construct a *patch-embedding* $\phi(\cdot; D) : \mathbb{R}^{d_P} \to \mathbb{R}^t$. That is, we embed each patch into $\mathbb{R}^t$ (for some t) based on its relation to the patches in the dictionary. We consider the following choices for the embedding:

1. **Hard Assignment.** One simple embedding is an embedding with *hard-assignment* of $0/1$, where we assign 1 in the index of the dictionary patches that are closest to the query patch (the patch that we are embedding).

can be extended to networks with bias.

[2] If there are multiple choices for $\text{k-}\pi_{\mathrm{dist}}$, we choose one arbitrarily.

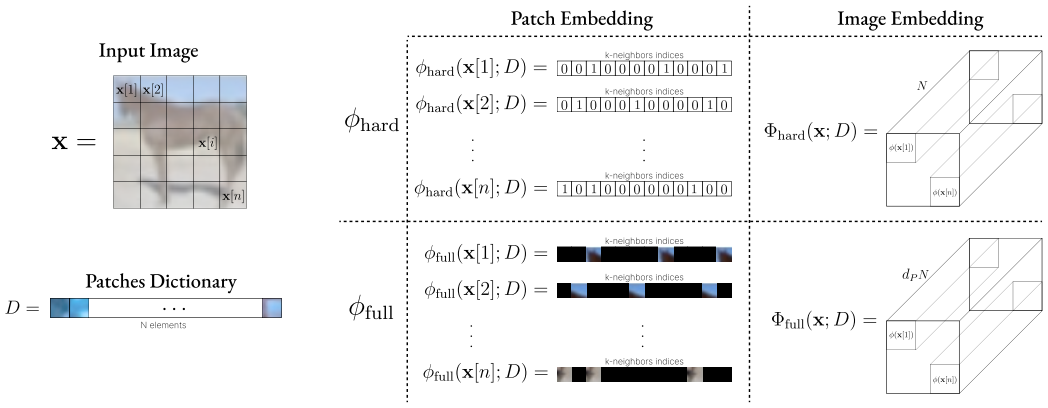

Figure 1: The patch-based image embedding. First, each patch $\mathbf{x}[i]$ is mapped using the patch embedding $\phi_{\text{hard}}(\cdot)$ or $\phi_{\text{full}}(\cdot)$, placing 1 or $\mathbf{x}[i]$ (respectively) in the indices of the k nearest-neighbors of $\mathbf{x}[i]$ in the dictionary D. Then, the representations of all the patches are concatenated together to obtain a representation for the full image $\mathbf{x}$.

Formally, we define the embedding ϕ_{hard} as follows:

$$\phi_{\text{hard}}(\mathbf{z}; D)_i = \begin{cases} 1 & \mathbf{v}_i \in \text{k-}\pi_{\text{dist}}(\mathbf{z}; D) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

2. **Full Information**. The embedding ϕ_{hard} relates each patch to the patches in the dictionary that are closest to it, but does not maintain information about the query patch. To overcome this, we consider an embedding that maintains full information about the query patch:

$$\phi^{(i)}_{\text{full}}(\mathbf{z}; D) = \begin{cases} \mathbf{z} & \mathbf{v}_i \in \text{k-}\pi_{\text{dist}}(\mathbf{z}; D) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $\phi^{(i)}_{\text{full}}(\mathbf{z}; D) := \phi_{\text{full}}(\mathbf{z}; D)_{(i-1)\cdot N+1,\ldots,i\cdot N}$.

Namely, $\phi_{\text{full}}(\mathbf{z}; D) \in \mathbb{R}^{d_P \cdot N}$ consists of N blocks, each of size d_P, and the i-th block gets the value of $\mathbf{z}$ if $\mathbf{v}_i$ is a nearest neighbour of $\mathbf{z}$ in D, and otherwise we set all the block to zero. This way, the embedding maintains information both about the neighborhood of the patch $\mathbf{z}$ in the dictionary, and the value of $\mathbf{z}$.

Using a *patch-embedding* ϕ (i.e, ϕ_{hard} or ϕ_{full}), we define an *input-embedding* $\Phi(\cdot; D)$ as follows:

$$\Phi(\mathbf{x}; D) = [\phi(\mathbf{x}[1]; D), \ldots, \phi(\mathbf{x}[n]; D)] \tag{4}$$

That is, Φ maps each patch of $\mathbf{x}$ using the patch embedding ϕ and concatenates all patch embeddings. Figure 1 shows an illustration of how the embedding Φ is constructed using the dictionary D and the input image $\mathbf{x}$.

Learning algorithm. We consider a semi-supervised algorithm $\mathcal{A}_{Patch}$ for learning image data, which is similar to the algorithm presented by Coates et al. (2011). The algorithm takes as a parameter the dictionary-size N. It consists of an unsupervised stage, followed by a supervised stage.

Unsupervised stage: We assume that we have access to an unlabeled training dataset S_u sampled from the marginal distribution of $\mathcal{I}$ over $\mathcal{X}$. Define the set $P_u = \{\mathbf{x}[i] \mid \mathbf{x} \in S_u, 1 \le i \le n\}$, i.e., the set of all patches of images in S_u. We perform a clustering procedure which given P_u and a number N, returns a set of patches D of size N. For our theoretical analysis we will consider a greedy clustering algorithm, which performs a farthest-first traversal. The clustering algorithm is described in Appendix A. This algorithm was originally proposed for the k-center problem and it is a 2-approximation algorithm (Gonzalez, 1985).[3] We use this clustering algorithm in our theoretical analysis, as it has known run-time guarantees. In practice, other clustering algorithms, such as k-means, can be considered (as we do in our empirical study).

Supervised stage: Here we assume that we have a dictionary $D = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$ of patches obtained by the unsupervised stage. Given a labeled training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ sampled from $\mathcal{I}$, we perform hard linear SVM over the embedding, and return a predictor s.t.

$$\bar{\mathbf{w}} = \arg\min_{\mathbf{w}} \|\mathbf{w}\|^2 \text{ s.t. } \forall i \in [m], y_i \langle \mathbf{w}, \Phi(\mathbf{x}_i; D) \rangle \ge 1$$

Observe that the Hard-SVM objective is equivalent to minimizing the logistic loss using gradient-descent, as shown in Soudry et al. (2018). The prediction of the label of a new point $\mathbf{x}$ is then $h(\mathbf{x}) = \text{sign}(\langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \rangle)$. See Figure 2 for an illustration of the supervised stage.

Remark 3.6. Our learning algorithm is inspired by the algorithm originally proposed by Coates et al. (2011) which was subsequently improved by Thiry et al. (2021). However, in our implementation there are some details that differ from these previous variants (clustering, architecture, etc.). For a detailed comparison see Appendix C.2.

[3] See also Williamson & Shmoys (2011) Section 2.2.

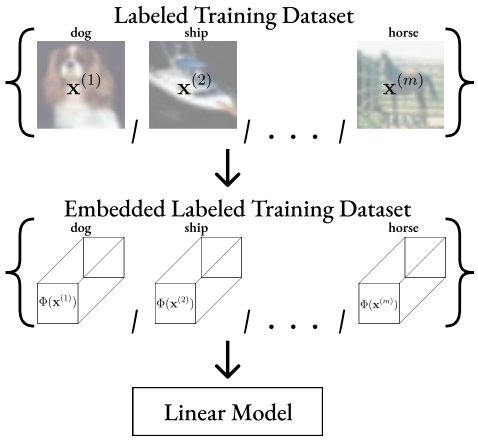

Figure 2: The supervised stage. The labeled training data is transformed using the embedding $\Phi(\cdot; D)$ that was calculated in the unsupervised stage. Then, a linear model is trained on the transformed input images.

4. Main Result

In this section we give a detailed analysis of the algorithm $\mathcal{A}_{Patch}$ presented in the previous section. We show that $\mathcal{A}_{Patch}$ finds a hypothesis with small loss on distributions that are labeled by a shallow CNN. The sample complexity and run-time of the algorithm depend on different notions of covering of the patch distribution, as well as on the choice of embedding function.

We start by analyzing $\mathcal{A}_{Patch}$ with the *hard-assignment* embedding, showing that it learns shallow CNNs, with dependence on the covering number of the patch distribution. Next, we analyze the algorithm when using the *full-information* embedding, and show a learnability result depending on existence of a covering of the patches which "agrees" with the linear regions of the target CNN.

For simplicity, we analyze the algorithm with an embedding that uses 1-nearest-neighbor (i.e., k-π_{dist} with $k = 1$). However, similar analysis can be applied to $k > 1$.

4.1. Hard Assignment Embedding

Our main result in this section shows that for every distribution $\mathcal{I}$ satisfying Assumption 3.4 with covering number $N_{\ell_2}(P_{\mathcal{I}}, r) = N_0$ (with a proper choice of r), given a large enough sample, $\mathcal{A}_{Patch}$ returns a predictor with small error:

Theorem 4.1. *Fix $\epsilon, \delta \in (0, 1)$. Let $\mathcal{I}$ be some distribution satisfying Assumption 3.4 with Lipschitz constant L_0. Fix $r = \frac{1}{24L_0}$ and let $N_0 = N_{\ell_2}(P_{\mathcal{I}}, r)$. Then, there exists some universal constant $\alpha > 0$, s.t. running algorithm $\mathcal{A}_{Patch}$ with Φ_{hard} on the ℓ_2 metric, a dictionary of size $N = N_0$, and number of labeled and unlabeled examples (m, m_u respectively) satisfying*

$$m \geq \alpha \frac{nN_0 \log(1/\epsilon) + \log(3/\delta)}{\epsilon}, \text{ and}$$
$$m_u \geq \frac{6N_0 \cdot m}{\delta} \log\left(\frac{3N_0}{\delta}\right)$$

returns w.p. (over the randomness of S_u and S) at least $1 - \delta$ a hypothesis h satisfying $\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}}[h(\mathbf{x}) \neq y] \leq \epsilon$.

Remark 4.2. In the above theorem we account separately for the number of labeled and unlabeled examples. This immediately implies sample-complexity guarantees in the classical PAC learning framework, where the total number of samples is $m + m_u$ (where we can disregard the labels of m_u examples).

We give the full proof of the theorem in Appendix B.2, and give a sketch of the argument here. To prove Theorem 4.1, we start by showing that sampling enough unlabeled examples from the marginal distribution over the inputs results in a good dictionary, that essentially covers the distribution of patches. This is done by observing all the balls from the covering of $P_{\mathcal{I}}$ that have at least ϵ/N_0 distributional mass, and showing that a sample from each of them is guaranteed with high probability.

Next, we show that the target function $F_{\mathbf{W},\mathbf{U}}$ can be approximated by a linear classifier over the representation Φ, given a good dictionary. We use the fact that the set of patches from the unlabeled data P_u covers the set $P_{\mathcal{I}}$ of patches with non-zero probability. So, after clustering, we still get a good covering. Next, we define the following vector:

$$\hat{\mathbf{w}} = \left(\ldots, \left\langle \mathbf{u}^{(i)}, \sigma(\mathbf{W}\mathbf{v}_j) \right\rangle, \ldots\right) \in \mathbb{R}^{Nn}$$

where the $\mathbf{v}_i$-s are the vectors in the dictionary D. Then, using the guarantees on the unsupervised stage (that the dictionary covers the set of patches), we show that the function $\hat{F}(\mathbf{x}) = \langle \hat{\mathbf{w}}, \Phi(\mathbf{x}; D) \rangle$ approximates $F_{\mathbf{W},\mathbf{U}}$. Thus, the algorithm succeeds because it can get from the unsupervised stage vectors that cover the set of patches, which in turn allows it to produce a sufficiently rich representation for learning the distribution with a linear classifier.

In Section 4.1.2, we show that, up to logarithmic factors, the number of labeled examples used by $\mathcal{A}_{Patch}$ is optimal. As for the run-time of the algorithm, notice that both the unsupervised and the supervised stage use efficient (polynomial time) algorithms. The run-time of the clustering algorithm used in the unsupervised stage is $O(N_0^2 d_P)$, and the hard SVM used in the supervised stage runs in time $O(mn^2N_0^2)$.

Observe that both the run-time and sample complexity of $\mathcal{A}_{Patch}$ do not necessarily depend on the number of neurons of the target network (denoted by l). This is because, when the patches are well-covered, the algorithm can approximate

any Patch-Lipschitz function. Also note that the Patch-Lipschitz constant L does not necessarily grow with l, so the dependence on l is not "hidden" in some other parameter.

4.1.1. Intrinsic Dimension and Covering Number

The analysis in the previous section shows that the sample complexity and run-time of our algorithm depend on one important measure - the covering number of the patch set $P_{\mathcal{I}}$. Observe that the covering number is often used as a measure of the intrinsic dimension of some given space. For example, the d-dimensional ℓ_2-ball $B = \mathcal{B}_{\ell_2}(\mathbf{u}, 1)$ has covering number $N_{\ell_2}(B, r) = C_d(1/r)^d$ (for some C_d which depends on d but not on r), and note that this remains true even if the ball is embedded in a space with larger extrinsic dimension. More generally, a bounded d-dimensional manifold has a covering number that grows exponentially with the intrinsic dimension d (which again can be much smaller than the extrinsic dimension). For more examples and further discussion on the relation between the covering number and measures of intrinsic dimension refer to e.g. Falconer (2004); Hamm & Steinwart (2020).

We see that if the distribution of patches (captured by the set $P_{\mathcal{I}}$) is concentrated on a low-dimensional structure (e.g., a low-dimensional manifold), we can expect the covering number of $P_{\mathcal{I}}$ to be moderate. On the other hand, if the patches fill a truly high-dimensional space, then our complexity guarantees become impractical, as these can grow exponentially with the dimension. That said, we next show that such dependence on the covering number is essentially unavoidable, if one wishes to learn any CNN to small loss.

4.1.2. Sample Complexity Lower Bound

We showed that with sample complexity that depends on the number of patches n and the covering number $N_0 := N_{\ell_2}(P, r)$, the algorithm $\mathcal{A}_{Patch}$ returns with high probability a hypothesis with small loss. Now, we will show that such dependence is unavoidable for guaranteeing such learnability result. Namely, any algorithm that learns all distributions labeled by a CNN with covering number N_0, must have sample complexity of $\Omega(nN_0/\epsilon)$:

Theorem 4.3. *Fix some $\epsilon \in (0, 1/8)$. Let $\mathcal{A}$ be some learning algorithm that uses m samples. Assume that for every $\mathcal{I}$ which satisfies Assumption 3.4 and for which $N_{\ell_2}(P_{\mathcal{I}}, 1/L) = N_0$, $\mathcal{A}$ returns w.p. at least $7/8$ a hypothesis h s.t. $\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}}[h(\mathbf{x}) \neq y] \leq \epsilon/8$. Then, $m \geq \frac{nN_0-2}{2\epsilon}$.*

The proof of the Theorem is in Appendix B.4. It uses similar arguments as in the No-Free-Lunch Theorem (e.g., see Sec. 5 in Shalev-Shwartz & Ben-David (2014)).

4.2. Full Information Embedding

In the previous section, we showed that the algorithm $\mathcal{A}_{Patch}$ finds good predictors on distributions labeled by a CNN, when using the embedding Φ_{hard}. The complexity of the algorithm essentially depends on the covering number of the distribution of patches. The reason that we need the patch distribution to have a small covering number is because we aim to approximate the target function (over the patches) by a function that is constant over each ball. This approximation is good enough when the function is Lipschitz, but requires the balls to be of small size (relative to the inverse of the Lipschitz constant). A possible improvement to this approach is to approximate the target by assigning a linear function (instead of a constant) for each ball, which potentially allows taking a covering with balls of larger size.

In this section, we show that when using the embedding Φ_{full}, the complexity of the algorithm $\mathcal{A}_{Patch}$ depends on a covering with larger balls, exploiting some geometrical properties of ReLU networks. Specifically, our result depends on the characterization of the linear regions of the target CNN. The linear regions are defined as follows:

Definition 4.4. For some function $f : \mathbb{R}^d \to \mathbb{R}^r$, a linear region is a maximal connected set $H \subseteq \mathbb{R}^n$ s.t. $\forall \mathbf{x} \in H$, $f(\mathbf{x}) = \mathbf{W}_H\mathbf{x} + b_H$ for some $\mathbf{W}_H \in \mathbb{R}^{r\times d}$ and $b \in \mathbb{R}^r$.

It is well known that ReLU networks divide the input space to linear regions, where almost every point $\mathbf{x}$ is in the interior of some linear region (Pascanu et al., 2013; Montúfar et al., 2014). We prove our result under an assumption on the relation between the linear regions of the target network and the distribution of the patches. Essentially, we assume that the patch distribution "respects" the linear regions structure of the target CNN in the sense that it is well-clustered inside balls that do not cross between different linear regions.

We start by defining the relation between a cover for the patch distribution and the linear regions of the target CNN:

Definition 4.5. Let C be some r-covering of P. We say that C respects f if

- For every $\mathbf{v} \in C$ there exists a linear region $H_{\mathbf{v}}$ of f s.t. $\mathcal{B}_{\text{dist}}(\mathbf{v}, r) \subseteq H_{\mathbf{v}}$.
- For $\mathbf{v}, \mathbf{v}' \in C$ s.t. $\mathbf{v} \neq \mathbf{v}'$ we have $\text{dist}(\mathbf{v}, \mathbf{v}') > 4r$.

This definition ensures that patches lie in clusters that are well separated from one another, and that each cluster does not cross between linear regions. Under this assumption, we show that $\mathcal{A}_{Patch}$ finds a good predictor when using Φ_{full}.

Theorem 4.6. *Let $\mathcal{I}$ be some distribution which satisfies Assumption 3.4, with some target $F_{\mathbf{W},\mathbf{U}}$. Denote by f the function $f(\mathbf{v}) = \sigma(\mathbf{W}\mathbf{v})$. Assume that for some $r > 0$, there is an r-covering for $P_{\mathcal{I}}$ of size N_0 that respects f. Fix some $\epsilon, \delta \in (0, 1/4)$. Assume that we run $\mathcal{A}_{Patch}$ using*

the embedding Φ_{full}, with dictionary size $N = N_0$, m_u unlabeled samples and m labeled samples. Then, there exists some universal constant $\alpha > 0$, s.t. when taking:

$$m \geq \alpha \frac{nd_P N_0 \log(1/\epsilon) + \log(3/\delta)}{\epsilon}, \text{ and}$$
$$m_u \geq \frac{6N_0 \cdot m}{\delta} \log\left(\frac{3N_0}{\delta}\right)$$

the algorithm returns w.p. at least $1-\delta$ a hypothesis h s.t. $\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}}[h(\mathbf{x}) \neq y] \leq \epsilon$.

Observe that the sample complexity bound looks very similar to the bound in Theorem 4.1. However, the size of the cover N_0 in Theorem 4.6 can be much smaller than the covering number required for Theorem 4.1, since we now do not require $r = O(1/L_0)$. Indeed, the balls that we take for the cover can be of larger size, as long as they do not cross linear regions. Therefore, using the embedding Φ_{full}, instead of Φ_{hard}, can result in better guarantees. The proof can be found in Appendix B.3.

4.3. Improved Complexity with Rank Constraints

We showed that $\mathcal{A}_{Patch}$ learns a good predictor when using the Φ_{full} embedding. The number of *labeled* examples required for learning is $\tilde{O}(nd_P N/\epsilon)$, which is derived from the number of parameters of the learned predictor. In reality, depending on the choice of N and the size of the image n, this number of parameters may become very big. Additionally, since the size of the predictor is $nd_P N$, this can also cause the computational complexity of optimization to be impractical (although, still polynomial in the parameters n, d_P, N). To overcome this, we reduce the computational burden by replacing the single layer of the linear predictor by a linear network that forces more parameter sharing, thus reducing the number of parameters. Effectively, this is equivalent to imposing structural constraints on the linear function we learn. We now introduce the details of such constrained linear network.

Observe that placing a linear classifier on top of the embedding Φ_{full} is equivalent to defining a separate linear function for every patch in the dictionary and every spatial location in the image. This means that for every $i \in [N]$ and for every $j \in [n]$ we have a linear classifier $\mathbf{z} \mapsto \langle \mathbf{w}^{(i,j)}, \mathbf{z} \rangle$. One constraint we can make is to have for each patch in D a single linear function $\mathbf{z} \mapsto \langle \mathbf{w}^{(i)}, \mathbf{z} \rangle$, instead of having different linear functions for different spatial locations in the image. Then we can use another linear function which given the N outputs for each spatial location $j \in [n]$ predicts the final output $f(\Phi_{\text{full}}(\mathbf{x})) = \sum_{j\in[n]} (\langle \mathbf{u}^{(j)}, \psi(x[j]) \rangle$ where $\psi(\mathbf{z})_i = \left\langle \mathbf{w}^{(i)}, \mathbf{z} \right\rangle \cdot \mathbf{1}\{\mathbf{v}_i \in \text{k-}\pi_{\text{dist}}(\mathbf{z}; D)\}$. Note that the total number of parameters for this constrained linear classifier is $N \cdot (d_P + n)$ instead of $N \cdot n \cdot d_P$. See Figure 3 (top) for an illustration of this constrained linear model.

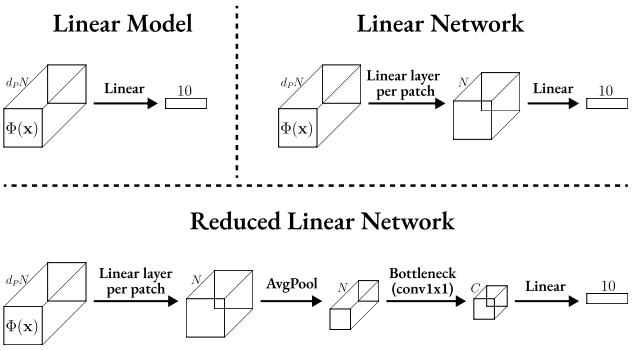

Figure 3: The different linear model variants we experiment with, in order to reduce the dimension of the linear classifier.

To further reduce the number of parameters, we introduce two more linear operations: 1) An average pooling layer after the output of ψ and 2) a "bottleneck" layer, in the form of a 1×1 convolutional layer which reduces the number of channels N to some smaller value (performed after the average pooling layer). Since these layers are linear, we still learn a linear classifier on top of the embedding. In Section 5 we show the effect of these constrains. See Figure 3 (bottom) for an illustration of this reduced linear model.

While the reduction in parameters has its merits, it makes the optimization more complex. That is, while in the "one-layer" linear case there are known guarantees for finding the optimal solution, for linear networks this is not necessarily the case. However, various papers show that deep linear networks can be optimized efficiently using SGD (e.g., Kawaguchi (2016); Hardt & Ma (2016); Ji & Telgarsky (2019a); Arora et al. (2018)), and indeed we show experimentally in Section 5 that SGD performs well in our case.

5. Experiments

In this section we empirically study different variants of the algorithm $\mathcal{A}_{Patch}$ when applied to the CIFAR-10 dataset, to complement our theoretical analysis. In Section 5.2 we examine the performance of the algorithm equipped with the suggested embeddings Φ_{hard} and Φ_{full}, comparing it to a vanilla one-hidden-layer CNN. We also examine the importance of the data-dependent nature of our embedding, by taking a dictionary of patches sampled from a Gaussian distribution. In Section 5.3 we examine how various constraints imposed on the linear classifier affect the performance of the model (see Section 4.3). In Section 5.4 we show that the algorithm $\mathcal{A}_{Patch}$ (equipped with Φ_{full}) can scale to deep models. In Section 5.5 we show that patches from natural images are clustered together, supporting our assumption on the covering number of the patch distribution. Our code is available here: github.com/AlonNT/patch-based-learning.

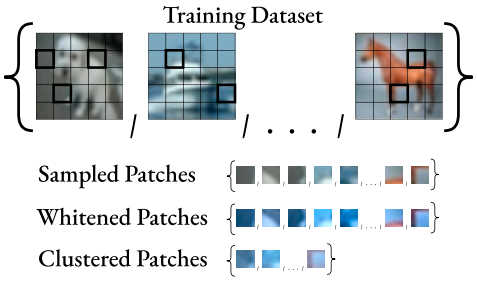

Figure 4: The unsupervised stage. A large amount of patches are sampled from the unlabeled training dataset. The patches are whitened and clustered together to form the patches dictionary D.

Table 1: Comparison between $\mathcal{A}_{Patch}$ and baselines.

	Test Accuracy
Vanilla 1 hidden-layer CNN	80.08% ($\pm$0.16%)
Φ_{hard} with random patches	71.36% ($\pm$0.24%)
Φ_{full} with random patches	76.04% ($\pm$0.13%)
Φ_{hard} with data patches	78.80% ($\pm$0.32%)
Φ_{full} with data patches	**81.23%** ($\pm$0.15%)

Table 2: The effect of the constraint on the linear function.

	Test Accuracy	#Parameters
Simple	76.38% ($\pm$0.34%)	$150M$
Constrained	76.61% ($\pm$0.19%)	$2M$

5.1. Implementation Details

We obtain the patches dictionary $D = \{v_1, \ldots, v_N\}$ by sampling M patches uniformly at random from the training data, performing whitening[4] on the patches followed by k-means clustering to get N centroids (see Figure 4). Unless noted otherwise, we use $N = 1024$. At inference time, given an input image, we transform the whitened patches of the image using $\phi_{\text{hard}}/\phi_{\text{full}}$. The embeddings of the patches are then concatenated together to get the embedding of the image (see Figure 1). Then, we learn a linear classifier on top of the embedding of the input images. We choose $k = 0.25 \cdot N$ as the number of neighbors defining the embedding (see the definition of k-$\pi_{\text{dist}}(\mathbf{z}; D)$ in Section 3.3) [5].

In Sections 5.2 and 5.4 we use the constrained linear classifier described in Section 4.3. The effects of the different constraints are studied in Section 5.3. See Appendix C.3 for details on efficient implementation using tensor operations.

5.2. Results

In this part, we compare the performance of the following: 1) A linear network (see Section 4.3) over Φ_{hard} and Φ_{full}. 2) Same as (1), but with patches sampled randomly from a Gaussian distribution. 3) A vanilla one-hidden-layer CNN with the same architecture as the models described above.[6]

Table 1 shows the results of this experiment. Note that $\mathcal{A}_{Patch}$ with embedding Φ_{full} outperforms both the shallow CNN and the Φ_{hard} embedding. Additionally, using random patches is inferior to using clustered patches from the training data, which shows the importance of using data-dependent features. Note that $\mathcal{A}_{Patch}$ with Φ_{hard} is the version proposed in Thiry et al. (2021). See Appendix C.2 for a detailed comparison between our implementation and theirs. For the full parameters setting we use please refer to Appendix C.1.

5.3. The Effect of the Linear Function Constraints

We examine the effects of the constraints we impose on the linear function on top of the embedding Φ_{full} (see Section 4.3). Table 2 compares the performance and cost between the original non-constrained linear classifier and the low-rank constrained linear classifier described in Section 4.3. The constrained version performs very similar to the original non-constrained version, while being 75x smaller. Since the standard linear classifier is very large, for technical reasons we run it with a smaller dictionary size of $N = 256$.

We also investigate the effect of using avg-pooling and 1×1 conv. (bottleneck) which further reduce the computational burden and improve accuracy. See the comparison in Table 3. The version with both average-pooling and bottleneck reaches higher accuracy than the original version, while being 39x smaller. The results here use the same parameters setting as described in Appendix C.1 (i.e. $N = 1024$).

Table 3: Examining the effect of pooling and bottleneck. For the versions with average-pooling layer we also add a batch-normalization which empirically helped the optimization.

	Test Accuracy	#Parameters
Original	78.98% ($\pm$0.22%)	$8.2M$
AvgPool	80.42% ($\pm$0.13%)	$0.66M$
Bottleneck	80.38% ($\pm$0.30%)	$0.44M$
Both	**81.23%** ($\pm$0.15%)	**0.21M**

5.4. The Effect of Model Depth

Note that the procedure we describe in Section 3.3 can also be applied to intermediate features of our learned model, allowing us to train a "deep" model. We train it

[4] See Appendix C.4 for details on how we perform whitening.

[5] See Appendix C.5 for different values of N and k.

[6] We use a learned convolution layer with kernel-size 5×5 and 1024 channels followed by a ReLU activation function, 4×4 average-pooling with stride 4, batch-normalization, 1×1 convolution with 32 output channels, and finally a linear layer.

in a layer-wise fashion, as follows. Denote our training-set by $S \subseteq \mathbb{R}^{H_0 \times W_0 \times C_0}$. Assume we already trained layers $\mathcal{N}_0, \ldots, \mathcal{N}_{r-1}$ and we now train layer $\mathcal{N}_r$. Since $\mathcal{N}_{r-1}$ is a linear network on top of the input embedding, removing the last linear layer of the network results in a new embedding of the input space, which we denote by $E_{r-1} : \mathbb{R}^{H_0 \times W_0 \times C_0} \to \mathbb{R}^{H_{r-1} \times W_{r-1} \times C_{r-1}}$. We fix this embedding and do the same procedure as before - sample patches uniformly from $E_{r-1}(S)$, perform whitening, run k-means clustering on the whitened patches to obtain a patches-dictionary D_r, and train a (constrained) linear classifier on top of the $\Phi_{\text{full}}(E_{r-1}(S); D_r)$.

Observe that the performance of Φ_{full}, similarly to a standard CNN, improves with depth, although admittedly achieving much lower final accuracy compared to a deep CNN. In contrast, note that when using Φ_{hard} the performance deteriorate with depth, suggesting that this embedding might not be suitable for deeper models. Table 4 shows the benefit of depth for $\mathcal{A}_{Patch}$ with the embedding Φ_{hard} and Φ_{full}, compared to vanilla CNN and a CNN trained in a layer-wise fashion (same as $\mathcal{A}_{Patch}$) [7].

Table 4: Test accuracy across depth. CNN (lw) denotes the performance of a CNN trained layerwise.

	Depth 1	Depth 2	Depth 3	Depth 4
Φ_{full}	81.33%	82.55%	82.74%	82.88%
Φ_{hard}	78.30%	65.41%	56.22%	50.69%
CNN	80.16%	87.91%	89.51%	89.62%
CNN (lw)	80.22%	85.44%	85.33%	85.30%

5.5. Intrinsic Dimension Estimation

Our theoretical results show that the sample complexity of $\mathcal{A}_{Patch}$ depends on the covering number of the patch distribution. Here we aim to show that patches in natural images indeed have a relatively small covering-number. Observe that if a set has a small covering number, it should be possible to cover it with a small number of cluster centers.

We therefore perform the following experiment. We sample 1 million patches of size 5×5 from CIFAR-10 / ImageNet dataset, perform k-means clustering with $k = 2, \ldots, 1024$ and show the mean distance between a patch and its assigned centroid. We compare the original/whitened patches to the following baselines, which tend to imitate "unnatural" data - random Gaussian patches, and the patches after shuffling the pixels values. Since the norms of the whitened/random patches are in a different scale than the original patches, we normalize all patches to unit-vectors. The results, shown in Figure 5, suggest that the patches are closer to their centroids than the shuffled patches, and the whitening operator increases the distances dramatically (although still smaller than the random Gaussian patches).

[7] For readability, STD is omitted. All runs have STD of $0.1\% - 0.3\%$, except in the deep versions of Φ_{hard} where training is not stable and STD is $5\% - 10\%$.

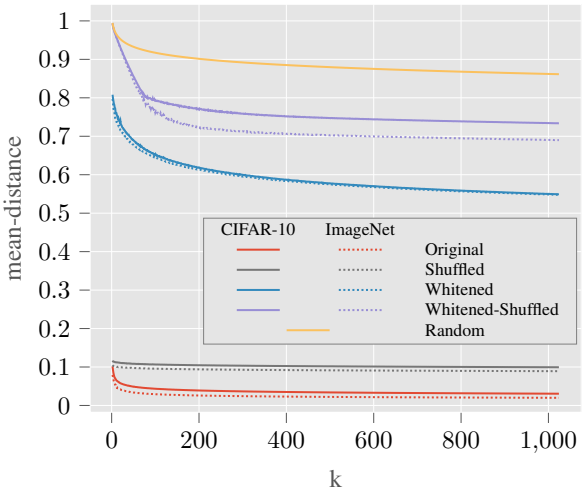

Figure 5: Mean distance between patches and centroids.

6. Discussion and Future Work

CNNs are a key component in deep learning, and in particular in machine vision. Since they are hard to learn in the worst case, it is important to understand what makes them learnable in practice. Naturally, the answer must involve the statistical structure of real data these networks are trained on. Here we take a step in this direction by relating the statistics of image patches to the learnability of neural nets. Our focus is on the statistics of patches, but it will be interesting to extend it to inter-patch correlations.

Our work also highlights the potential of methods that learn representations in an unsupervised manner. It is thus related to many recent works on self-supervised learning for images. It will be interesting to provide guarantees as we have here for these methods.

Finally, in Section 4.2 we show that sample complexity is related to an agreement between the prediction function and the image statistics. We believe such bounds can be made tighter, and leave it for future work.

Acknowledgments

This project was partially funded by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant ERC HOLI 819080). AB is supported by a Google PhD fellowship.

References

Arora, S., Cohen, N., and Hazan, E. On the optimization of deep networks: Implicit acceleration by overparameterization. In International Conference on Machine Learning, pp. 244–253. PMLR, 2018.

Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In International Conference on Machine Learning, pp. 322–332, 2019.

Awasthi, P., Tang, A., and Vijayaraghavan, A. Efficient algorithms for learning depth-2 neural networks with general relu activations. arXiv preprint arXiv:2107.10209, 2021.

Bartlett, P. L., Kulkarni, S. R., and Posner, S. E. Covering numbers for real-valued function classes. IEEE transactions on information theory, 43(5):1721–1724, 1997.

Basri, R. and Jacobs, D. W. Efficient representation of low-dimensional manifolds using deep networks. 2017.

Brutzkus, A. and Globerson, A. Globally optimal gradient descent for a convnet with gaussian inputs. In International Conference on Machine Learning, pp. 605–614, 2017.

Brutzkus, A. and Globerson, A. An optimization and generalization analysis for max-pooling networks. arXiv preprint arXiv:2002.09781, 2020.

Buchanan, S., Gilboa, D., and Wright, J. Deep networks and the multiple manifold problem. In International Conference on Learning Representations, 2020.

Cao, Y. and Gu, Q. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In Advances in Neural Information Processing Systems, pp. 10836–10846, 2019.

Chen, M., Jiang, H., Liao, W., and Zhao, T. Nonparametric regression on low-dimensional manifolds using deep relu networks: Function approximation and statistical recovery. arXiv preprint arXiv:1908.01842, 2019.

Cloninger, A. and Klock, T. A deep network construction that adapts to intrinsic dimensionality beyond the domain. Neural Networks, 2021.

Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

Daniely, A. and Malach, E. Learning parities with neural networks. Advances in Neural Information Processing Systems, 33, 2020.

Daniely, A. and Shalev-Shwartz, S. Complexity theoretic limitations on learning dnf's. In Conference on Learning Theory, pp. 815–830. PMLR, 2016.

Daniely, A. and Vardi, G. Hardness of learning neural networks with natural weights. Advances in Neural Information Processing Systems, 33, 2020.

Daniely, A. and Vardi, G. From local pseudorandom generators to hardness of learning. arXiv preprint arXiv:2101.08303, 2021.

Diakonikolas, I., Kane, D. M., Kontonis, V., and Zarifis, N. Algorithms and sq lower bounds for pac learning one-hidden-layer relu networks. In Conference on Learning Theory, pp. 1514–1539. PMLR, 2020.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.

Du, S., Lee, J., Tian, Y., Singh, A., and Poczos, B. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima. In International Conference on Machine Learning, pp. 1339–1348. PMLR, 2018a.

Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. Gradient descent finds global minima of deep neural networks. In International Conference on Machine Learning, pp. 1675–1685, 2019.

Du, S. S. and Goel, S. Improved learning of one-hidden-layer convolutional neural networks with overlaps. arXiv preprint arXiv:1805.07798, 2018.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. International Conference on Learning Representations, 2018b.

Falconer, K. Fractal geometry: mathematical foundations and applications. John Wiley & Sons, 2004.

Ge, R., Kuditipudi, R., Li, Z., and Wang, X. Learning two-layer neural networks with symmetric inputs. arXiv preprint arXiv:1810.06793, 2018.

Ghorbani, B., Mei, S., Misiakiewicz, T., and Montanari, A. When do neural networks outperform kernel methods? Journal of Statistical Mechanics: Theory and Experiment, 2021(12):124009, 2021.

Goldt, S., Mézard, M., Krzakala, F., and Zdeborová, L. Modeling the influence of data structure on learning in neural networks: The hidden manifold model. Physical Review X, 10(4):041044, 2020.

Gonzalez, T. F. Clustering to minimize the maximum intercluster distance. Theoretical computer science, 38: 293–306, 1985.

Hamm, T. and Steinwart, I. Adaptive learning rates for support vector machines working on data with low intrinsic dimension. arXiv preprint arXiv:2003.06202, 2020.

HaoChen, J. Z., Wei, C., Gaidon, A., and Ma, T. Provable guarantees for self-supervised deep learning with spectral contrastive loss. arXiv preprint arXiv:2106.04156, 2021.

Hardt, M. and Ma, T. Identity matters in deep learning. arXiv preprint arXiv:1611.04231, 2016.

Haussler, D. Sphere packing numbers for subsets of the boolean n-cube with bounded vapnik-chervonenkis dimension. Journal of Combinatorial Theory, Series A, 69 (2):217–232, 1995.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. In Advances in neural information processing systems, pp. 8571–8580, 2018.

Ji, Z. and Telgarsky, M. Gradient descent aligns the layers of deep linear networks. ICLR, 2019a.

Ji, Z. and Telgarsky, M. Polylogarithmic width suffices for gradient descent to achieve arbitrarily small test error with shallow relu networks. In International Conference on Learning Representations, 2019b.

Kamath, P., Montasser, O., and Srebro, N. Approximate is good enough: Probabilistic variants of dimensional and margin complexity. In Conference on Learning Theory, pp. 2236–2262. PMLR, 2020.

Kawaguchi, K. Deep learning without poor local minima. In Advances In Neural Information Processing Systems, pp. 586–594, 2016.

Klivans, A. R. and Sherstov, A. Cryptographic hardness results for learning intersections of halfspaces. In Proc. 47 IEEE Symp. on Foundations of Computer Science. Citeseer, 2006.

Klivans, A. R. and Sherstov, A. A. Cryptographic hardness for learning intersections of halfspaces. Journal of Computer and System Sciences, 75(1):2–12, 2009.

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

Malach, E. and Shalev-Shwartz, S. A provably correct algorithm for deep learning that actually works. arXiv preprint arXiv:1803.09522, 2018.

Malach, E., Kamath, P., Abbe, E., and Srebro, N. Quantifying the benefit of using differentiable learning over tangent kernels. arXiv preprint arXiv:2103.01210, 2021.

Montúfar, G., Pascanu, R., Cho, K., and Bengio, Y. On the number of linear regions of deep neural networks. arXiv preprint arXiv:1402.1869, 2014.

Oymak, S. and Soltanolkotabi, M. End-to-end learning of a convolutional neural network via deep tensor decomposition. arXiv preprint arXiv:1805.06523, 2018.

Pascanu, R., Montufar, G., and Bengio, Y. On the number of response regions of deep feed forward networks with piece-wise linear activations. arXiv preprint arXiv:1312.6098, 2013.

Rahimi, A. and Recht, B. Weighted sums of random kitchen sinks: replacing minimization with randomization in learning. In Nips, pp. 1313–1320. Citeseer, 2008.

Rahimi, A., Recht, B., et al. Random features for large-scale kernel machines. In NIPS, volume 3, pp. 5. Citeseer, 2007.

Schmidt-Hieber, J. Deep relu network approximation of functions on a manifold. arXiv preprint arXiv:1908.00695, 2019.

Shalev-Shwartz, S. and Ben-David, S. Understanding machine learning: From theory to algorithms. Cambridge university press, 2014.

Shawe-Taylor, J., Cristianini, N., et al. Kernel methods for pattern analysis. Cambridge university press, 2004.

Song, M. J., Zadik, I., and Bruna, J. On the cryptographic hardness of learning single periodic neurons. arXiv preprint arXiv:2106.10744, 2021.

Soudry, D., Hoffer, E., Nacson, M. S., Gunasekar, S., and Srebro, N. The implicit bias of gradient descent on separable data. The Journal of Machine Learning Research, 19(1):2822–2878, 2018.

Thiry, L., Arbel, M., Belilovsky, E., and Oyallon, E. The unreasonable effectiveness of patches in deep convolutional kernels methods. arXiv preprint arXiv:2101.07528, 2021.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Keysers, D., Uszkoreit, J., Lucic, M., et al. Mlp-mixer: An all-mlp architecture for vision. arXiv preprint arXiv:2105.01601, 2021.

Trockman, A. and Kolter, J. Z. Patches are all you need? arXiv preprint arXiv:2201.09792, 2022.

Vershynin, R. High-dimensional probability. An Introduction with Applications, 2017.

Vinnikov, A. and Shalev-Shwartz, S. K-means recovers ica filters when independent components are sparse. In International Conference on Machine Learning, pp. 712–720. PMLR, 2014.

Wang, T., Buchanan, S., Gilboa, D., and Wright, J. Deep networks provably classify data on curves. Advances in Neural Information Processing Systems, 34, 2021.

Williamson, D. P. and Shmoys, D. B. The design of approximation algorithms. Cambridge university press, 2011.

Zhou, D.-X. The covering number in learning theory. Journal of Complexity, 18(3):739–767, 2002.

A. Clustering Algorithm

Below is a pseudo-code of the farthest-first clustering algorithm from (Gonzalez, 1985), which we use in our theoretical analysis:

Algorithm 1 Clustering

Input: Set of patches P_u, $N > 0$.
Pick an arbitrary $\mathbf{z} \in P_u$.
Set $D = \{\mathbf{z}\}$.
for $i = 2, ..., N$ **do**:
  Find $\mathbf{v} \in P_u$ which maximizes $\mathrm{dist}(\mathbf{v}, D)$ (where $\mathrm{dist}(\mathbf{v}, D) := \min\{\mathrm{dist}(\mathbf{v}, \mathbf{u}) \mid \mathbf{u} \in D\}$)
  $D \leftarrow D \cup \{\mathbf{v}\}$
return D.

B. Proofs

B.1. Proof of Lemma 3.3

Proof. Observe that:

$$\begin{aligned}
|F_{\mathbf{W},\mathbf{u}}(\mathbf{x}) - F_{\mathbf{W},\mathbf{u}}(\mathbf{x}')| &= \left| \sum_i \left\langle \mathbf{u}^{(i)}, \sigma(\mathbf{W}\mathbf{x}[i]) \right\rangle - \sum_i \left\langle \mathbf{u}^{(i)}, \sigma(\mathbf{W}\mathbf{x}'[i]) \right\rangle \right| \\
&\leq \sum_i \left\| \mathbf{u}^{(i)} \right\| \|\sigma(\mathbf{W}\mathbf{x}[i]) - \sigma(\mathbf{W}\mathbf{x}'[i])\| \\
&\leq \sum_i \left\| \mathbf{u}^{(i)} \right\| \|\mathbf{W}(\mathbf{x}[i] - \mathbf{x}'[i])\| \leq \sum_i \left\| \mathbf{u}^{(i)} \right\| \|\mathbf{W}\|_2 \max_i \|\mathbf{x}[i] - \mathbf{x}'[i]\|
\end{aligned}$$

where the first inequality follows from the triangle inequality and Cauchy–Schwarz inequality, and the second inequality follows from the fact that σ is 1-Lipschitz. Note that $\|\cdot\|_2$ with respect to matrices (here used on $\mathbf{W}$) refers to the operator-norm. □

B.2. Proof of Theorem 4.1

The result of Theorem 4.1 follows immediately from the following Theorem:

Theorem B.1. *Let $\epsilon, \epsilon', \delta, \delta' \in (0, 1)$. Let $\mathcal{I}$ be some distribution satisfying Assumption 3.4 with Lipschitz constant L_0. Fix $r = \frac{1}{24L_0}$ and let $N_0 = N_{\ell_2}(P_{\mathcal{I}}, r)$. Assume that $m_u \geq \frac{N_0}{\epsilon'} \log\left(\frac{N_0}{\delta'}\right)$. Then, there exists some universal constant $\alpha > 0$, s.t. running algorithm $\mathcal{A}_{Patch}$ with Φ_{hard} on the ℓ_2 metric, a dictionary of size $N = N_0$, and number of labeled samples m s.t.*

$$\alpha \frac{nN_0 \log(1/\epsilon) + \log(1/\delta)}{\epsilon} \leq m \leq \frac{\delta}{2\epsilon'}$$

returns w.p.[8] $\geq 1 - \delta' - 2\delta$ *a hypothesis h satisfying $\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}}[h(\mathbf{x}) \neq y] \leq \epsilon + \epsilon'$.*

Proof. First of all, we show that since we have a large sample of unlabeled images, with high probability the patches in our sample will be a $(2r)$-covering of the non-negligible part of the patches distribution.

Indeed, let C be an r-covering of $P_{\mathcal{I}}$ of minimal size, namely $|C| = N_0$. Let $C' \subseteq C$ be the subset of balls that have mass $\geq \frac{\epsilon'}{N_0}$, i.e.:

$$C' = \left\{ \mathbf{v} \in C \ : \ \mathbb{P}_{x\sim\mathcal{I}}\left[\exists i \text{ s.t.} \mathbf{x}[i] \in \mathcal{B}_{\ell_2}(\mathbf{v}, r)\right] \geq \frac{\epsilon'}{N_0} \right\}$$

We first show that with probability $\geq 1 - \delta'$, for every $\mathbf{c} \in C'$ there is patch in P_u (the patches of the unlabeled images) in

[8] Over the randomness of S_u and S.

the ball $\mathcal{B}_{\ell_2}(\mathbf{c}, r,)$, namely $\mathcal{B}_{\ell_2}(\mathbf{c}, r) \cap P_u \neq \emptyset$. Indeed, fix some $\mathbf{c} \in C'$, and by definition of C':

$$\mathbb{P}\left[\mathcal{B}_{\ell_2}(\mathbf{c}, r) \cap P_u = \emptyset\right] \leq \left(1 - \frac{\epsilon'}{N_0}\right)^{m_u} \leq \exp\left(-\frac{m_u \epsilon'}{N_0}\right) \leq \frac{\delta'}{N_0}$$

and the required follows from the union bound.

Now, assume that for every $\mathbf{c} \in C'$ there exists $\mathbf{v} \in P_u$ such that $\mathbf{v} \in \mathcal{B}_{\ell_2}(\mathbf{c}, r)$. Therefore, we also have $\mathcal{B}_{\ell_2}(\mathbf{c}, r) \subseteq \mathcal{B}_{\ell_2}(\mathbf{v}, 2r)$, and so:

$$\cup_{\mathbf{c} \in C'} \mathcal{B}_{\ell_2}(\mathbf{c}, r) \subseteq \cup_{\mathbf{v} \in P_u} \mathcal{B}_{\ell_2}(\mathbf{v}, 2r) \tag{5}$$

Now, we show that the dictionary of patches (D) that was obtained using the clustering procedure on the patches in our sampled unlabeled data, also covers the non-negligible part of the patches distribution (although with a slightly larger radius than the entire set of patches P_u, i.e. $6r$ instead of $2r$).

Indeed, since $P_u \subseteq P_{\mathcal{I}}$, there exists a set of patches A of size $N = N_{\ell_2}(P_{\mathcal{I}}, r)$ such that:

$$P_u \subseteq \cup_{\mathbf{v} \in A} \mathcal{B}_{\ell_2}(\mathbf{v}, 2r) \tag{6}$$

The clustering algorithm guarantees that (Gonzalez, 1985),

$$\max_{\mathbf{v} \in P_u} \min_{\mathbf{c} \in D} \|\mathbf{v} - \mathbf{c}\| \leq 2 \max_{\mathbf{v} \in P_u} \min_{\mathbf{c} \in A} \|\mathbf{v} - \mathbf{c}\| \leq 4r \tag{7}$$

where the last inequality holds because A is a $(2r)$-covering of P_u (see Eq. (6)). It follows that $P_u \subseteq \cup_{\mathbf{v} \in D} \mathcal{B}_{\ell_2}(\mathbf{v}, 4r)$ which implies that $\cup_{\mathbf{v} \in P_u} \mathcal{B}_{\ell_2}(\mathbf{v}, 2r) \subseteq \cup_{\mathbf{v} \in D} \mathcal{B}_{\ell_2}(\mathbf{v}, 6r)$. Then, by Eq. (5) we get:

$$\cup_{\mathbf{v} \in C'} \mathcal{B}_{\ell_2}(\mathbf{v}, r) \subseteq \cup_{\mathbf{v} \in D} \mathcal{B}_{\ell_2}(\mathbf{v}, 6r) \tag{8}$$

Now, we'll define a vector $\hat{w} \in \mathbb{R}^{Nn}$ that will be used as a linear-function on top of the embedding Φ_{hard}, which will be equal to the target labeling CNN (on "most" of the input data distribution, i.e. on input images that don't contain patches from negligible balls in the patches distribution).

Fix a target CNN $F_{\mathbf{W},\mathbf{U}}(\mathbf{x}) = \sum_{i=1}^{n} \left\langle \mathbf{u}^{(i)}, \sigma(\mathbf{W}\mathbf{x}[i]) \right\rangle$ satisfying Assumption 3.4. Denote by f the function $f(\mathbf{v}) = \sigma(\mathbf{W}\mathbf{v})$ (which is the target function *on the patches* - the target convolution layer only, without the linear "readout" layer). Define the vector

$$\hat{\mathbf{w}} = \left(\left\langle \mathbf{u}^{(1)}, f(\mathbf{v}_1) \right\rangle, ..., \left\langle \mathbf{u}^{(1)}, f(\mathbf{v}_N) \right\rangle, ..., \left\langle \mathbf{u}^{(n)}, f(\mathbf{v}_1) \right\rangle, ..., \left\langle \mathbf{u}^{(n)}, f(\mathbf{v}_N) \right\rangle \right) \in \mathbb{R}^{Nn}$$

i.e., $\hat{\mathbf{w}}_{(i-1)N+j} = \left\langle \mathbf{u}^{(i)}, f(\mathbf{v}_j) \right\rangle$ for all $1 \leq i \leq n$ and $1 \leq j \leq N$. Let $\hat{F}(\mathbf{x}) = \langle \hat{\mathbf{w}}, \Phi_{\text{hard}}(\mathbf{x}; D) \rangle$.

For some $\mathbf{x} \in \mathcal{X}$, we say that $\mathbf{x}$ is *good* if for all $1 \leq i \leq n$, $\mathbf{x}[i] \in \cup_{\mathbf{v} \in C'} \mathcal{B}_{\ell_2}(\mathbf{v}, r)$. In words, an image x is good if all of its patches lie in non-negligible balls of the patches distribution. We say that the training set S is good if for all $\mathbf{x} \in S$, $\mathbf{x}$ is good.

By the definition of C', the probability that $\mathbf{x}$ is not good is at most $\sum_{\mathbf{v} \in C \setminus C'} \mathbb{P}_{\mathbf{x} \sim \mathcal{I}} \left[\exists i, \mathbf{x}[i] \in \mathcal{B}_{\ell_2}(\mathbf{v}, r)\right] \leq \epsilon'$. Let $\xi_j \in \{0, 1\}$ be the random variable indicating whether the j-th example in S is not good. Observe that $\mathbb{E}\left[\sum_{j=1}^{m} \xi_j\right] = \sum_{j=1}^{m} \mathbb{E}\left[\xi_j\right] = m\epsilon'$. Therefore, by Markov's inequality we have:

$$\mathbb{P}\left[S \text{ is not good }\right] = \mathbb{P}\left[\exists j \in [m] \; s.t. \; \xi_j = 1\right] = \mathbb{P}\left[\sum_{j=1}^{m} \xi_j \geq 1\right] \leq \mathbb{E}\left[\sum_{j=1}^{m} \xi_j\right] = m\epsilon' < \delta$$

Therefore, the probability that S is good is at least $1 - \delta$.

For some query patch $\mathbf{z}$, we denote by $\pi_{\text{dist}}(\mathbf{z}; D)$ the closest patch to $\mathbf{z}$ in D:

$$\pi_{\text{dist}}(\mathbf{z}; D) = \arg\min_{\mathbf{v} \in D} \text{dist}(\mathbf{v}, \mathbf{z})$$

So, for $k = 1$ we have k-$\pi_{\text{dist}}(\mathbf{z}; D) = \{\pi_{\text{dist}}(\mathbf{z}; D)\}$. For some $\mathbf{x} \in \mathcal{X}$, define $\mathbf{x}' \in \mathcal{X}$ to be the image where every patch in it is $\mathbf{x}$'s patch nearest-neighbor in D, i.e. for all $i \in [n]$ $\mathbf{x}'[i] = \pi_{\ell_2}(\mathbf{x}[i]; D)$. Observe that we have:

$$\hat{F}(\mathbf{x}) = \langle \hat{\mathbf{w}}, \Phi_{\text{hard}}(\mathbf{x}; D) \rangle = \sum_i \left\langle \mathbf{u}^{(i)}, \sigma(\mathbf{W}\pi_{\ell_2}(\mathbf{x}[i]; D)) \right\rangle = \sum_i \left\langle \mathbf{u}^{(i)}, \sigma(\mathbf{W}\mathbf{x}'[i]) \right\rangle = F_{\mathbf{W},\mathbf{U}}(\mathbf{x}')$$

Now, fix some good $\mathbf{x} \in \mathcal{X}$. Since every patch in $\mathbf{x}$ is in some non-negligible ball in the patches-distribution, and these balls are covered by our patches-dictionary D, every patch in $\mathbf{x}$ is close (up to $6r$) to some patch in D. Since the predicted function $\hat{F}$ on $\mathbf{x}$ equals to the target function $F_{\mathbf{W},\mathbf{U}}$ on an $\mathbf{x}'$, and $\mathbf{x}$ and $\mathbf{x}'$ are close to one another, the Lipshitzness of the target function $F_{\mathbf{W},\mathbf{U}}$ guarantees the outputs will be similar.

Indeed, from what we showed for all i we have $\|\mathbf{x}[i] - \pi_{\ell_2}(\mathbf{x}[i]; D)\| \leq 6r$ and therefore:

$$\left| F_{\mathbf{W},\mathbf{U}}(\mathbf{x}) - \hat{F}(\mathbf{x}) \right| = |F_{\mathbf{W},\mathbf{U}}(\mathbf{x}) - F_{\mathbf{W},\mathbf{U}}(\mathbf{x}')| \leq L_0 \cdot \max_i \|\mathbf{x}[i] - \pi_{\ell_2}(\mathbf{x}[i]; D)\| \leq 6L_0 r \leq \frac{1}{4}$$

For any $(\mathbf{x}, y) \sim \mathcal{I}$ where $\mathbf{x}$ is good, assumption 3.4 guarantees $yF_{\mathbf{W},\mathbf{U}}(\mathbf{x}) \geq 1$. If $y = +1$ it must be that $\hat{F}(\mathbf{x}) \geq \frac{3}{4}$ and if $y = -1$ it must be that $\hat{F}(\mathbf{x}) \leq -\frac{3}{4}$ so in any case $y\hat{F}(\mathbf{x}) \geq \frac{3}{4}$.

Overall we got that for every $(\mathbf{x}, y) \sim \mathcal{I}$ where $\mathbf{x}$ is good, $y\hat{F}(\mathbf{x}) \geq \frac{1}{2}$.

For now we consider two events: (1) Event where for every $\mathbf{c} \in C'$ we have $\mathcal{B}_{\ell_2}(\mathbf{c}, r) \cap P_u \neq \emptyset$ which occurs with probability at least $1 - \delta'$ and (2) Event that S is good which holds with probability at least $1 - \delta$. If the first two events happen, then the predictor $\hat{F}$ has zero error on S (since S is good). So, by standard VC generalization bounds (e.g., via the Fundamental Theorem and bounds on the VC dimension of linear classifiers from Shalev-Shwartz & Ben-David (2014)), with probability at least $1 - \delta' - 2\delta$ over the randomness of S and S_u:

$$\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I},\, \mathbf{x} \text{ is good}} \left[y \left\langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \right\rangle < 0 \right] \leq \epsilon$$

Therefore,

$$\begin{aligned}\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}} \left[y \left\langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \right\rangle < 0 \right] &= \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I},\, \mathbf{x} \text{ is good}} \left[y \left\langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \right\rangle < 0 \right] \\ &\quad + \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I},\, \mathbf{x} \text{ is not good}} \left[y \left\langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \right\rangle < 0 \right] \leq \epsilon + \epsilon'\end{aligned}$$

which concludes the proof.

□

B.3. Proof of Theorem 4.6

To prove Theorem 4.6 we use the following Lemma, stating that with a large enough sample from unlabeled data there exists a linear separator for the a non-negligible part of data distribution. Later on, we'll require a large enough sample from labeled data which will enable learning this linear separator.

Lemma B.2. *Let $\epsilon, \delta, \in (0, 1/4)$. Let $\mathcal{I}$ be some distribution satisfying the assumptions of Theorem 4.6, with some target $F_{\mathbf{W},\mathbf{U}}$. Then, w.p. at least $1 - \delta$ over sampling $S_u \sim \mathcal{I}$ of size $m_u \geq \frac{N_0}{\epsilon} \log\left(\frac{N_0}{\delta}\right)$, there exists some $\hat{\mathbf{w}}$ s.t. $\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}} \left[y \left\langle \hat{\mathbf{w}}, \Phi_{\text{full}}(\mathbf{x}) \right\rangle \geq 1 \right] \geq 1 - \epsilon$.*

Proof. First of all, we show that every "important" ball in the patches distribution contains some patch in our patches dictionary (important in the sense that it contains some patch from our sampled data).

Well, let C be an r-covering of $P_{\mathcal{I}}$ of size N_0 that respects $f(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$. Let $C' \subseteq C$ be the subset of balls that have mass $\geq \frac{\epsilon}{N_0}$, as defined in the proof of Theorem 4.1. Similarly, we have that with probability $\geq 1 - \delta$, for every $\mathbf{c} \in C'$ there exists some $\mathbf{v} \in P_u$ s.t. $\mathbf{v} \in \mathcal{B}_{\text{dist}}(\mathbf{c}, r)$. Denote by $C_u \subseteq C$ as follows:

$$C_u := \{\mathbf{c} \in C \ : \ \exists \mathbf{v} \in P_u \text{ s.t. } \mathbf{v} \in \mathcal{B}_{\text{dist}}(\mathbf{c}, r)\}$$

That is, C_u is the set of balls in the cover C from which there is a point in the sample of patches P_u. So, we showed that w.p. at least $1-\delta$ we have $C' \subseteq C_u$. Now, observe that by the guarantees of (Gonzalez, 1985), since $|C_u| \leq |C| = N_0$ it holds that:

$$\max_{\mathbf{v}\in P_u} \min_{\mathbf{v}'\in D} \text{dist}(\mathbf{v},\mathbf{v}') \leq 2 \max_{\mathbf{v}\in P_u} \min_{\mathbf{c}\in C_u} \text{dist}(\mathbf{v},\mathbf{c}) \leq 2r \tag{9}$$

Where the last inequality is by definition of C_u.

Claim: For every $\mathbf{v} \in P_{\mathcal{I}}$ and for every $\mathbf{c} \in C$, if $\mathbf{v} \notin \mathcal{B}_{\text{dist}}(\mathbf{c}, r)$ then $\text{dist}(\mathbf{v},\mathbf{c}) > 3r$.

Proof. The intuition is as follows. From the definition of the covering that "respects" the linear regions, it must contain ball that are distant from each other. Therefore, if a patch is not contained in some ball, it must be quite far from it, since the patch must be contained in some other ball which is far away. Formally, let $\mathbf{v} \in P_{\mathcal{I}}$ and $\mathbf{c} \in C$ s.t. $\mathbf{v} \notin \mathcal{B}_{\text{dist}}(\mathbf{c}, r)$. By definition of C, there exists $\mathbf{c}' \in C$ s.t. $\mathbf{v} \in \mathcal{B}_{\text{dist}}(\mathbf{c}', r)$. Therefore, $\mathbf{c} \neq \mathbf{c}'$ and so by the Definition 4.5, it holds that $\text{dist}(\mathbf{c},\mathbf{c}') > 4r$. Therefore, we have:

$$4r < \text{dist}(\mathbf{c},\mathbf{c}') \leq \text{dist}(\mathbf{v},\mathbf{c}) + \text{dist}(\mathbf{v},\mathbf{c}') \leq \text{dist}(\mathbf{v},\mathbf{c}) + r$$

□

Claim: For every $\mathbf{c} \in C_u$, there is $\mathbf{v} \in D$ s.t. $\mathbf{v} \in \mathcal{B}_{\text{dist}}(\mathbf{c}, r)$.

Proof. The intuition is that if there exists a ball that doesn't contain any patch from D, from the previous claim it must be far away from every patch in D, contradicting the guarantee from the clustering procedure. Formally, assume there is some $\mathbf{c} \in C_u$ such that for all $\mathbf{v} \in D$ it holds that $\mathbf{v} \notin \mathcal{B}_{\text{dist}}(\mathbf{c}, r)$. By definition of C_u, there is some $\mathbf{v_c} \in P_u$ s.t. $\text{dist}(\mathbf{c}, \mathbf{v_c}) \leq r$. By the previous claim, for all $\mathbf{v} \in D$ it holds that

$$3r < \text{dist}(\mathbf{c},\mathbf{v}) \leq \text{dist}(\mathbf{c},\mathbf{v_c}) + \text{dist}(\mathbf{v},\mathbf{v_c}) \leq \text{dist}(\mathbf{v},\mathbf{v_c}) + r$$

Therefore:

$$\max_{\mathbf{v}'\in P_u} \min_{\mathbf{v}\in D} \text{dist}(\mathbf{v},\mathbf{v}') \geq \min_{\mathbf{v}\in D} \text{dist}(\mathbf{v},\mathbf{v_c}) > 2r$$

contradicting Eq. (9). □

Now, for every $\mathbf{c} \in C_u$, from Definition 4.5 there is a linear region $H_{\mathbf{c}}$ of f such that $\mathcal{B}_{\text{dist}}(\mathbf{c}, r) \subseteq H_{\mathbf{c}}$. In other words, there exists $\mathbf{W_c} \in \mathbb{R}^{l\times d_P}$ such that $f(\mathbf{v}) = \mathbf{W_c v}$ for all $\mathbf{v} \in \mathcal{B}_{\text{dist}}(\mathbf{c}, r)$.[9] Since C_u covers P_u and $D \subseteq P_u$, for every $\mathbf{v} \in D$ there is some $\mathbf{c}(\mathbf{v}) \in C_u$ s.t. $\mathbf{v} \in \mathcal{B}_{\text{dist}}(\mathbf{c}(\mathbf{v}), r)$.

Now, we'll define the vector $\hat{w} \in \mathbb{R}^{Nnd_P}$ which will be used as a linear-function on top of the embedding Φ_{full} to be equal to the target CNN on "good" $\mathbf{x}$'s. Denote $D = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$, and define the vector $\hat{w}$ as follows:

$$\hat{\mathbf{w}} = \left[\left(\mathbf{u}^{(1)}\right)^\top \mathbf{W}_{\mathbf{c}(\mathbf{v}_1)}, \ldots, \left(\mathbf{u}^{(1)}\right)^\top \mathbf{W}_{\mathbf{c}(\mathbf{v}_N)}, \ldots, \left(\mathbf{u}^{(n)}\right)^\top \mathbf{W}_{\mathbf{c}(\mathbf{v}_1)}, \ldots, \left(\mathbf{u}^{(n)}\right)^\top \mathbf{W}_{\mathbf{c}(\mathbf{v}_N)} \right]$$

i.e., for all $1 \leq i \leq n$ and $1 \leq j \leq N$, $\hat{\mathbf{w}}_{(i-1)N+j}$ is a block of size d_P defined as $\hat{\mathbf{w}}_{(i-1)N+j} = \left(\mathbf{u}^{(i)}\right)^\top \mathbf{W}_{\mathbf{c}(\mathbf{v}_j)}$.

Observe that:

$$\begin{aligned}
\langle \hat{\mathbf{w}}, \Phi_{\text{full}}(\mathbf{x}) \rangle &= \sum_{i=1}^{n} (\mathbf{u}^{(i)})^\top \sum_{j=1}^{N} \mathbf{W}_{\mathbf{c}(\mathbf{v}_j)} \phi^{(j)}_{\text{full}}(\mathbf{x}[i]; D) \\
&= \sum_{i=1}^{n} \left\langle \mathbf{u}^{(i)}, \mathbf{W}_{\mathbf{c}(\pi_{\text{dist}}(\mathbf{x}[i];D))} \mathbf{x}[i] \right\rangle
\end{aligned}$$

For some $\mathbf{x} \in \mathcal{X}$, we call $\mathbf{x}$ *good* as in the proof of Theorem 4.1. Now, for all *good* $\mathbf{x}$, for every i denote by $\mathbf{c}(\mathbf{x}[i]) \in C'$ the center of the ball containing $\mathbf{x}[i]$, i.e. the vector $\mathbf{c}(\mathbf{x}[i]) \in C'$ satisfying $\mathbf{x}[i] \in \mathcal{B}_{\text{dist}}(\mathbf{c}(\mathbf{x}[i]), r) \subseteq H_{\mathbf{c}(\mathbf{x}[i])}$. Since the ball is contained in some linear region, i.e. $\mathcal{B}_{\text{dist}}(\mathbf{c}(\mathbf{x}[i]), r) \subseteq H_{\mathbf{c}(\mathbf{x}[i])}$, we get that $f(\mathbf{x}[i]) = \mathbf{W}_{\mathbf{c}(\mathbf{x}[i])}\mathbf{x}[i]$.

[9] There is no bias because the first Convolutional layer has no bias

Claim: Denote by $\mathbf{v}_\pi^i$ and nearest-neighbor of $\mathbf{x}[i]$ in the patches-dictionary D, i.e. $\mathbf{v}_\pi^i = \pi_{\text{dist}}(\mathbf{x}[i]; D)$. Then, if $C' \subseteq C_u$, we have that for all good $\mathbf{x}$ and all i, $\mathbf{c}(\mathbf{x}[i]) = \mathbf{c}(\mathbf{v}_\pi^i)$.

In words, the claim states that every patch in every good $\mathbf{x}$ is contained in the same ball as its nearest-neighbor in D.

Proof. Intuitively, the claim is true because if the two patches were not in the same ball, they were distant from each other (from the previous claims). This will contradict what we previously showed, that each patch is quite close to its nearest-neighbor in the patches dictionary.

Formally, we have $\mathbf{c}(\mathbf{x}[i]) \in C' \subseteq C_u$, and from the previous claim it holds that there is $\mathbf{v} \in D$ s.t. $\mathbf{v} \in \mathcal{B}_{\text{dist}}(\mathbf{c}(\mathbf{x}[i]), r)$. Therefore,

$$\text{dist}(\mathbf{x}[i], \mathbf{v}_\pi^i) \leq \text{dist}(\mathbf{x}[i], \mathbf{v}) \leq \text{dist}(\mathbf{x}[i], \mathbf{c}(\mathbf{x}[i])) + \text{dist}(\mathbf{c}(\mathbf{x}[i]), \mathbf{v}) \leq 2r$$

Note that

$$\text{dist}(\mathbf{c}(\mathbf{x}[i]), \mathbf{c}(\mathbf{v}_\pi^i)) \leq \text{dist}(\mathbf{c}(\mathbf{x}[i]), \mathbf{x}[i]) + \text{dist}(\mathbf{x}[i], \mathbf{v}_\pi^i) + \text{dist}(\mathbf{v}_\pi^i, \mathbf{c}(\mathbf{v}_\pi^i))$$

and if $\mathbf{c}(\mathbf{v}_\pi^i) \neq \mathbf{c}(\mathbf{x}[i])$, by Definition 4.5 we have $\text{dist}(\mathbf{c}(\mathbf{x}[i]), \mathbf{c}(\mathbf{v}_\pi^i)) > 4r$ and therefore

$$\text{dist}(\mathbf{x}[i], \mathbf{v}_\pi^i) \geq \text{dist}(\mathbf{c}(\mathbf{v}_\pi^i), \mathbf{c}(\mathbf{x}[i])) - \text{dist}(\mathbf{x}[i], \mathbf{c}(\mathbf{x}[i])) - \text{dist}(\mathbf{v}_\pi^i, \mathbf{c}(\mathbf{v}_\pi^i)) > 2r$$

in contradiction to what we showed, and therefore the required follows. □

So, if $C' \subseteq C_u$ we have for all *good* $\mathbf{x}$:

$$\begin{aligned} F_{\mathbf{W},\mathbf{U}}(\mathbf{x}) &= \sum_{i=1}^{n} \left\langle \mathbf{u}^{(i)}, f(\mathbf{x}[i]) \right\rangle = \sum_{i=1}^{n} \left\langle \mathbf{u}^{(i)}, \mathbf{W}_{\mathbf{c}(\mathbf{x}[i])} \mathbf{x}[i] \right\rangle \\ &= \sum_{i=1}^{n} \left(\mathbf{u}^{(i)}\right)^\top \mathbf{W}_{\mathbf{c}(\mathbf{v}_\pi^i)} \mathbf{x}[i] = \langle \hat{\mathbf{w}}, \Phi_{\text{full}}(\mathbf{x}) \rangle \end{aligned}$$

And the required follows from the fact that $\mathbb{P}_{\mathbf{x}\sim\mathcal{I}}[\mathbf{x} \text{ is not good}] \leq \epsilon$, as shown in the proof of Theorem 4.1. □

Proof of Theorem 4.6. Fix $\epsilon' = \delta/m$ and $\delta' = \delta$. From Lemma B.2, w.p. $1 - \delta'$ over sampling S_u, there exists some $\hat{\mathbf{w}}$ s.t. for all *good* $\mathbf{x}$ it holds that $F_{\mathbf{W},\mathbf{U}}(\mathbf{x}) = \langle \hat{\mathbf{w}}, \Phi_{\text{full}}(\mathbf{x}) \rangle$. Recall that, as in Theorem 4.1, we say that S is *good* if $\mathbf{x}$ is *good* for all $\mathbf{x} \in S$. Similarly, we get that by choice of ϵ' we have w.p. $\geq 1 - \delta$, that the sample S is *good*.

So, if S is *good* and $C' \subseteq C_u$, there exists some $\hat{\mathbf{w}}$ that separates S with margin 1, and from what we showed this happens w.p. $1 - \delta - \delta'$. Using standard VC bounds, w.p. $1 - \delta' - 2\delta$ it holds that

$$\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I},\, \mathbf{x} \text{ is good}} [y \langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \rangle < 0] \leq \epsilon$$

Therefore,

$$\begin{aligned} \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}} [y \langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \rangle < 0] &= \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I},\, \mathbf{x} \text{ is good}} [y \langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \rangle < 0] \\ &\quad + \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I},\, \mathbf{x} \text{ is not good}} [y \langle \bar{\mathbf{w}}, \Phi(\mathbf{x}; D) \rangle < 0] \leq \epsilon + \epsilon' \end{aligned}$$

which concludes the proof.

□

B.4. Proof of Theorem 4.3

Assume that $m < nN/2$. Denote $\Delta = 4n/L$. Let $Z = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(N)}\} \subseteq \mathbb{R}^{d_P}$ a set of N points s.t. $\mathbf{z}^{(j)} = (j\Delta, 1, 0, \ldots, 0)$. For every $i \in [n], j \in [N]$, define $\mathbf{x}_{i,j} \in \mathbb{R}^{d_I}$ as $\mathbf{x}_{i,j}[i] = \mathbf{z}^{(j)}$ and for all $i' \neq i$ it holds that $\mathbf{x}_{i,j}[i'] = 0$ (i.e., the i-th patch of $\mathbf{x}_{i,j}$ is $\mathbf{z}^{(j)}$ and the rest of the patches are zero).

Claim: For any $\mathbf{y} \in \{\pm 1\}^N$, there exists some function $f_\mathbf{y} : \mathbb{R}^{d_P} \to \mathbb{R}$ s.t.

- For all j it holds that $y_j f_{\mathbf{y}}(\mathbf{z}^{(j)}) \geq 1$.
- $f_{\mathbf{y}}$ is $(2/\Delta)$-Lipschitz.
- There exist some $\mathbf{W}_{\mathbf{y}} \in \mathbb{R}^{l \times d_P}$ and $\mathbf{u}_{\mathbf{y}} \in \mathbb{R}^l$ s.t. $f_{\mathbf{y}}(\mathbf{z}) = \langle \mathbf{u}_{\mathbf{y}}, \sigma(\mathbf{W}_{\mathbf{y}}\mathbf{z}) \rangle$

Proof. Denote $\psi(x) = \sigma(x+1) - 2\sigma(x) + \sigma(x-1)$. Observe that the following are immediate:

- $\psi(x) = 0$ for $x \notin (-1, 1)$.
- $\psi(0) = 1$.
- ψ is 1-Lipschitz.

Now, for all j, denote $\Psi_j(\mathbf{z}) = \psi\left(\frac{2}{\Delta}\mathbf{z}_1 - 2j\mathbf{z}_2\right) = \psi\left(\frac{2}{\Delta}\mathbf{z}_1 - 2j\right)$, and observe that $\Psi_j(\mathbf{z}^{(j)}) = 1$, $\Psi_j(\mathbf{z}^{(i)}) = 0$ for all $j \neq i$ and Ψ_j is $\frac{2}{\Delta}$-Lipschitz. Now, define:

$$f_{\mathbf{y}}(\mathbf{z}) = \sum_{j=1}^{N} y_j \Psi_j(\mathbf{z})$$

and observe that this function satisfies the required. □

Claim: For all $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n)} \in \{\pm 1\}^N$, the function $F_{\mathbf{Y}}(\mathbf{x}) = \sum_{i=1}^{n} f_{\mathbf{y}^{(i)}}(\mathbf{x}[i]) = \sum_{i=1}^{n} \left\langle \mathbf{u}_{\mathbf{y}^{(i)}}, \sigma(\mathbf{W}_{\mathbf{y}^{(i)}}\mathbf{x}[i]) \right\rangle$ is L-Patch-Lipschitz.

Proof. Fix some $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^{d_I}$. Since every function $f_{\mathbf{y}^{(i)}}$ is $\frac{2}{\Delta}$-Lipschitz we have that:

$$|F_{\mathbf{Y}}(\mathbf{x}) - F_{\mathbf{Y}}(\mathbf{x}')| \leq \sum_{i=1}^{n} \left| f_{\mathbf{y}^{(i)}}(\mathbf{x}[i]) - f_{\mathbf{y}^{(i)}}(\mathbf{x}'[i]) \right| \leq \frac{2}{\Delta} \sum_{i=1}^{n} \|\mathbf{x}[i] - \mathbf{x}'[i]\| \leq L \cdot \max_j \|\mathbf{x}[i] - \mathbf{x}'[i]\|$$

□

Claim: It holds that $N_{\ell_2}(Z, 1/L) = N$.

Proof. Since $|Z| = N$, it clearly holds that $N_{\ell_2}(Z, 1/L) \leq N$. Now, observe that for any $\mathbf{z}, \mathbf{z}'$ s.t. $\mathbf{z} \neq \mathbf{z}'$ it holds that $\|\mathbf{z} - \mathbf{z}'\| \geq 4/L$, and so for any ball $\mathcal{B}_{\ell_2}(\mathbf{c}, 1/L)$ it cannot hold that $\mathbf{z}, \mathbf{z}' \in \mathcal{B}_{\ell_2}(\mathbf{c}, 1/L)$. Therefore, every $\mathbf{z} \in$ is covered by a unique ball, and therefore any cover is of size at least N. □

So, for any choice of $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n)} \in \{\pm 1\}^N$, let $\mathcal{I}_{\mathbf{Y}}$ be the distribution defined as follows: w.p. $1 - \epsilon$, take $\left(\mathbf{x}_{1,1}, y_1^{(1)}\right)$, and with probability ϵ take $\left(\mathbf{x}_{i,j}, y_j^{(i)}\right)$ where $i \sim [n], j \sim [N]$ uniformly. Observe that by the previous claims $\mathcal{I}_{\mathbf{Y}}$ satisfies Assumption 3.4 and also $N_{\ell_2}(P_{\mathcal{I}_{\mathbf{y}}}, 1/L) = N$.

Let S be the sample seen by the algorithm $\mathcal{A}$, and $\mathcal{A}(S)$ be the hypothesis returned by $\mathcal{A}$ upon seeing the sample S. Let $\overline{S}$ be the samples not seen by the algorithm. We also denote $\mathbf{y}(S)$ the coordinates of $\mathbf{y}$ that appear in S, and $\mathbf{y}(\overline{S})$ the coordinates of $\mathbf{y}$ that do not appear in S. So, we have:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathbf{Y}} \mathop{\mathbb{E}}_{S} \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}_{\mathbf{Y}}} \left[\mathcal{A}(S)(\mathbf{x}) \neq y | \mathbf{x} \neq \mathbf{x}_{1,1}\right] &\geq \mathop{\mathbb{E}}_{\mathbf{Y}} \mathop{\mathbb{E}}_{S} \frac{1}{nN} \sum_{(\mathbf{x},y)\in\overline{S}} \mathbf{1}\{\mathcal{A}(S)(\mathbf{x}) \neq y\} \\
&\geq \frac{\mathbb{E}_S\left|\overline{S}\right|}{2nN}
\end{aligned}$$

Now, for some sample $S = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$, denote by M the number of unique samples in S, and we have:

$$M = \sum_{i\in[n],j\in[N]} \mathbf{1}\{\mathbf{x}_{i,j} \in S\} \leq 1 + \sum_{(i,j)\neq(1,1)} \mathbf{1}\{x_{i,j} \in S\}$$
$$\leq 1 + \sum_{(i,j)\neq(1,1)} \sum_{t=1}^{m} \mathbf{1}\{\mathbf{x}^{(t)} = \mathbf{x}_{i,j}\} = 1 + \sum_{t=1}^{m} \mathbf{1}\{\mathbf{x}^{(t)} \neq \mathbf{x}_{1,1}\}$$

So, we have:

$$\mathop{\mathbb{E}}_{S} \left|\overline{S}\right| = \mathop{\mathbb{E}}_{S}(nN - M) \geq nN - 1 - \mathop{\mathbb{E}}_{S} \sum_{t=1}^{m} \mathbf{1}\{\mathbf{x}^{(t)} \neq \mathbf{x}_{1,1}\} = nN - m\epsilon - 1$$

Therefore, if $m < \frac{nN-2}{2\epsilon}$ there exists a distribution $\mathcal{I}_{\mathbf{Y}}$ such that

$$\mathop{\mathbb{E}}_{S} \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}_{\mathbf{Y}}} \left[\mathcal{A}(S)(\mathbf{x}) \neq y | \mathbf{x} \neq \mathbf{x}_{1,1}\right] \geq \frac{1}{4}$$

Now, using Lemma B.1 in (Shalev-Shwartz & Ben-David, 2014) we get that w.p. at least $1/8$ we have

$$\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}_{\mathbf{Y}}} \left[\mathcal{A}(S)(\mathbf{x}) \neq y\right] \geq \epsilon \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{I}_{\mathbf{Y}}} \left[\mathcal{A}(S)(\mathbf{x}) \neq y | \mathbf{x} \neq \mathbf{x}_{1,1}\right] \geq \frac{\epsilon}{8}$$

C. Experimental Details

C.1. Parameters setting

We sample uniformly at random $262,144$ patches of size 5×5 from the training-data, run k-means clustering to get $N = 1,024$ patches which are used as the patches-dictionary. We use $k = 0.25 \cdot N$ as the number of neighbours. We do not use padding when calculating the embedding, so an input image of shape $32 \times 32 \times 3$ is transformed to $28 \times 28 \times 1024$. We then use average-pooling with size 4×4 and stride 4 to get $7 \times 7 \times 1024$, followed by batch-normalization and then 1×1 convolution (a.k.a. "bottleneck") which outputs $7 \times 7 \times 32$. Finally, a linear layer on top of that predicts the desired output.

Note that in section 5.4 we use a slightly different parameters setting - we use a smaller pooling (2×2 instead of 4×4), and it's being used only in the first layer (the deeper layers don't use pooling at all). The patch-size of the first layer is the same (5×5) but in the deeper layers we use a smaller patch-sizes (3×3 instead of 4×4) which proved to be helpful. The number of neighbors in the deeper layer was lowered to $k = 128$ which proved to be helpful as well.

We train for 200 epochs using SGD with momentum 0.9 and batch-size 64. We set 0.003 as the initial learning-rate, and decay it by a factor of 0.1 at epochs 100 and 150. We use standard data augmentations during training (random horizontal flip and random crop), but no augmentations are used when sampling the patches for the dictionary.

Each experiment was launched 5 times (with different random seeds) and the reported values are the mean and std across the 5 runs [10].

C.2. Comparison to previous works

We use k-means clustering for the unsupervised stage, where Coates et al. (2011) examined more techniques - auto-encoders, Boltzmann machines and Gaussian mixtures. They reported k-means achieves the best performance, so we focused on this method only. Thiry et al. (2021) discarded the unsupervised phase at all and obtained the patches dictionary simply by sampling patches uniformly at random from the training data. We chose to keep the clustering phase since we found it increased the performance a bit and it does not increase the training or inference time as its done only once during the initialization of the model.

When creating the embedding vector for each patch, Coates et al. (2011) experimented with a "softer" version - instead of the hard-assignment of 0/1 indicating the neighbourhood, assign a number which contains some information about the

[10] For readability, the standard-deviations were removed from Table 4, all of them are between 0.1% and 0.3% except the deep version of Φ_{hard} which were unstable with std of 5% - 10%.

distances between the patches. This results in a vector which is less sparse, with about half of the coordinates being non-zero, and they reported it works better. We follow the idea of Thiry et al. (2021) using the hard assignment with a largest amount of neighbors ($25\%/40\%$ of the patches-dictionary), which also results in an embedding vector that is less sparse, and it works slightly better than the softer version (according to our experiments).

In order to reduce the spatial dimension, Coates et al. (2011) split the image to 4 quadrants and summed over each one (i.e., adaptive sum-pooling to 2×2). Thiry et al. (2021) used average-pooling with overlapping kernels (e.g., 5×5 with stride 3) followed by adaptive average-pooling to 6×6. We conducted multiple experiments with different parameters and found the differences to be minor. We picked the parameters which gave the best accuracy, which were simply average-pooling with kernel 4×4 and stride 4.

Coates et al. (2011) uses a linear layer for predicting the classes' scores, given the (spatially-pooled) features. Thiry et al. (2021) improved it by first reducing the number of channels with a 1×1 convolution to 128, essentially replacing the linear layer with a (shallow) linear network. We follow the same idea, and found that the number of intermediate channels can be reduced even further to 32 without performance degradation.

Thiry et al. (2021) introduced "modern" techniques that were not used by Coates et al. (2011), like batch-normalization and data-augmentations (i.e., random crop and flip). We use the same techniques which proved to be useful.

Thiry et al. (2021) doubled the patches dictionary by adding the originally sampled patches multiplied by minus one. In practice, (by observing their publicly released code implementation) they had two separate "branches" of (Embedding $\rightarrow$ AvgPool $\rightarrow$ BatchNorm $\rightarrow$ Bottleneck), one operating on the original dictionary and one operating on the negative patches, and the two outputs are being summed up element-wise before feeding them to the final linear layer. We chose to keep the architecture simpler and didn't do it, since the performance gain was minor.

C.3. Efficient implementation using tensor operations

We implement our model (using the constrained linear classifier suggested in section 4.3) efficiently using tensor operations:

1. **Calculate k-nearest-neighbors mask**
   Each input image of shape $H \times W \times C$ is transformed into a tensor of shape $H' \times W' \times N$, where the ij entry is a k-hot vector containing 1 in the indices of the k nearest neighbors to the patch centered at ij.
   We implement the calculation of the distances between the whitened patches in the input image and the whitened patches in our dictionary using tensor operations as well, as described in Appendix B in Thiry et al. (2021).

2. **Convolution**
   Using a learnable convolution layer operating on the input image, we get output tensor of shape $H' \times W' \times N$. This layer calculates the linear functions $\mathbf{z} \mapsto \langle \mathbf{w}^{(q)}, \mathbf{z} \rangle + b^{(q)}$ (for every $q \in [N]$, i.e. for each patch in our dictionary), and then puts the results in the desired indices (i.e. the result of the q-th linear function on the patch centered at $x[i, j]$ is in the index i, j, q).

3. **Element-wise multiplication**
   We multiply the tensors from the previous two steps and get a tensor of shape $H' \times W' \times N$ which contains the outputs of the linear classifiers corresponding to the k nearest patches in their corresponding locations (and zero elsewhere).

4. **Linear layers**
   We (possibly) add more linear layers such like average-pooling, batch-normalization or 1×1 convolution. The effect of such linear layers are discussed in Table 3.

5. **Final linear classifier**
   The result of the previous stage is flattened and fed to a linear classifier, predicting the desired output.

C.4. Whitening

The whitening operator is calculated as follows. We create a matrix $X \in \mathbb{R}^{n \times d_P}$ containing all of the patches in the training-data. We calculate the covariance matrix $\Sigma := \frac{1}{n} X^T X$, perform EVD to get $\Sigma = EDE^T$, and then the PCA-whitening operator is defined as $W_{PCA} = (\lambda I + D)^{-1/2} E^T$ (where λ is the whitening regularization factor, we use $\lambda = 0.001$). Since whitened data will stay whitened after any rotation, we can also use ZCA-whitening which is defined as

$W_{ZCA} = E(\lambda I + D)^{-1/2} E^T = (\lambda I + \Sigma)^{-1/2}$ and results in whitened data that is as close as possible to the original data. Figure 6 shows the patches before and after the ZCA-whitening operator. Using whitened patches instead of the original patches is extremely important for the performance of the model - accuracy increase from 72% to 81%.

Another interesting observation is that the learned kernels of a vanilla CNN turn out to be quite similar to the whitened patches from the data (see Figure 6). We show the patches that had the highest effect on the final output of the network, by measuring the norms of their weights in the bottleneck layer. Interestingly, there is some similarity between the learned kernels and our patches. This might hint that k-means clustering over whitened patches and trained ConvNets result in similar convolutional filters (a similar phenomenon was suggested in Vinnikov & Shalev-Shwartz (2014)).

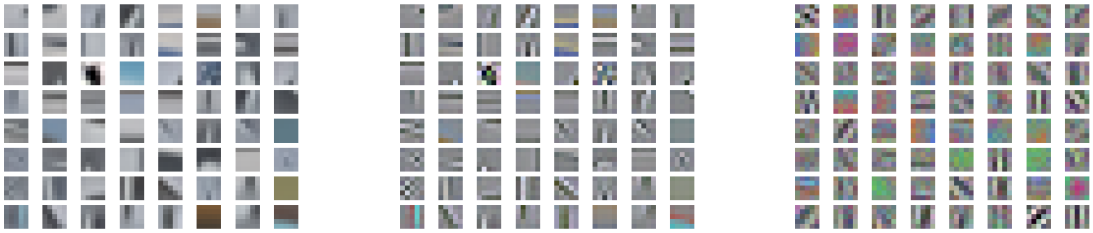

Figure 6: (Left) original patches. (Middle) the same patches after ZCA-whitening operator. (Right) Learned kernels in a vanilla CNN.

C.5. The effects of the different parameters in our model

In this section we examine the affect of different parameters for the algorithm $\mathcal{A}_{Patch}$, equipped with our suggested embedding Φ_{full}. Figure 7 (left) shows the accuracy for different values of k. Note that using $k = 1024$ (which is equal to the patches-dictionary size) results in a model achieving only 39% accuracy, which make sense because the information about the neighbourhood of each patch is completely lost.

Figure 7 (right) shows the accuracy for different sizes of the patches-dictionary, which increase to almost 84% with a dictionary of size $16,384$. Note that all runs are using the same parameters setting as in C.1, in particular the number of neighbors k is 25% of the dictionary size.

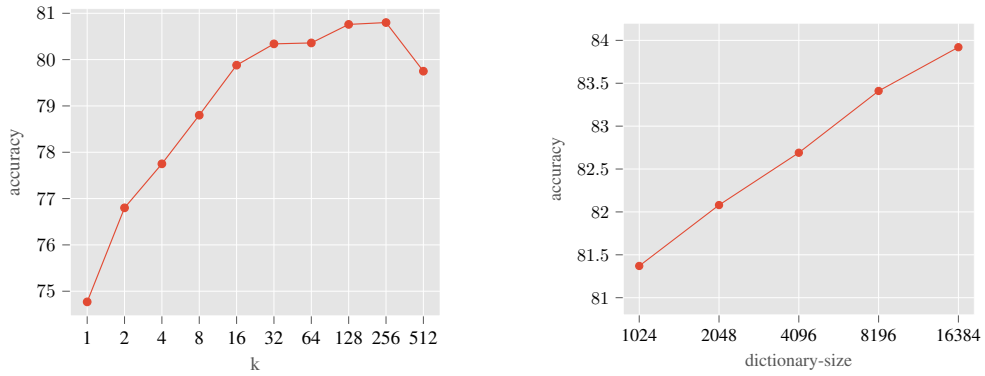

Figure 7: (Left) accuracy per k (number of neighbors), where the patches-dictionary size is $1,024$. (Right) accuracy per dictionary-size.