GRADIENTS AS FEATURES FOR DEEP REPRESENTATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We address the challenging problem of deep representation learning–the efficient adaption of a pre-trained deep network to different tasks. Specifically, we propose to explore gradient-based features. These features are gradients of the model parameters with respect to a task-specific loss given an input sample. Our key innovation is the design of a linear model that incorporates both gradient features and the activation of the network. We show that our model provides a local linear approximation to a underlying deep model, and discuss important theoretical insight. Moreover, we present an efficient algorithm for the training and inference of our model without computing the actual gradients. Our method is evaluated across a number of representation learning tasks on several datasets and using different network architectures. We demonstrate strong results in all settings. And our results are well-aligned with our theoretical insight.

1 INTRODUCTION

Despite tremendous success of deep models, training deep neural networks requires a massive amount of labeled data and computing resources. The recent development of representation learning holds great promises for improving data efficiency of training, and enables an easy adaption to different tasks using a same feature representation. These features can be learned via either unsupervised learning using deep generative models (Kingma & Welling, 2013; Dumoulin et al., 2016), or self-supervised learning with "pretext" tasks and pseudo labels (Noroozi & Favaro, 2016; Zhang et al., 2016; Gidaris et al., 2018), or transfer learning from another large-scale dataset (Yosinski et al., 2014; Oquab et al., 2014; Girshick et al., 2014). After learning, the the activation of the deep network are considered as generic features. By leveraging these features, simple classifiers, e.g., linear models, can be build for different tasks. However, given sufficient amount of training data, the performance of representation learning methods lack behind fully-supervised deep models.

As a step to bridge this gap, we propose to make use of gradient-based features from a pre-trained network, i.e., gradients of the model parameters relative to a task-specific loss given an input sample. Our key intuition is that these per-sample gradients contain task-relevant discriminative information. More importantly, we design a novel linear model that accounts for both gradient-based and activation-based features. The design of our linear model stems from the recent advances in the theoretical analysis of deep models. Specifically, our gradient-based features are inspired by the neural tangent kernel (Jacot et al., 2018; Lee et al., 2019; Arora et al., 2019b) modified for finite-width networks. Therefore, our model provides a local approximation of fine-tuning a underlying deep model. And the accuracy of rthe approximation is controlled by the semantic gap between the representation learning and the target task. Finally, we derive an efficient and scalable algorithm for training our linear model with gradient-based features.

To evaluate our model, we focus on visual representation learning in this paper, although our model can be easily modified for natural language processing or speech recognition. To this end, we consider a number of learning tasks in vision, including unsupervised, self-supervised and transfer learning. Our method was evaluated across tasks, datasets and architectures and compared against a set of baseline methods. We observe empirically that our model with gradient-based features outperforms the traditional activation-based features by a significant margin in all settings. Moreover, our results compares favorably against those produced by fine-tuning of network parameters.

Our main contributions are summarized as follows.

- We propose a novel representation learning method. At the core of our method lies in a linear model that builds on gradients of model parameters as the feature representation.
- From a theoretical perspective, we show that our linear model provides a local approximation of fine-tuning a underlying deep model. From a practical perspective, we devise an efficient and scalable algorithm for the training and inference of our method.
- We demonstrate strong results of our method across various representation learning tasks, different network architectures and several datasets. And our empirical results are well-aligned to our theoretical insight.

2 Related Work

Representation Learning. Learning good representation of data without expensive supervision remains a challenging problem. Representation learning using deep models has been recently explored. For example, different types of deep latent variable models (Kingma & Welling, 2013; Higgins et al., 2017; Berthelot et al., 2018; Dumoulin et al., 2016; Donahue et al., 2016; Dinh et al., 2016; Kingma & Dhariwal, 2018; Grathwohl et al., 2018) were considered for representation learning. These models were designed to fit to the distribution of data, yet their intermediate responses were found useful for discriminative tasks. Another example is self-supervised learning. This paradigm seeks to learn from a discriminative pretext task whose supervision comes almost for free. These pretext tasks for images include predicting rotation angles (Gidaris et al., 2018), solving jigsaw puzzles (Noroozi & Favaro, 2016) and colorizing grayscale images (Zhang et al., 2016). Finally, the idea of transfer learning hinges on the assumption that feature maps learned from a large and generic dataset can be shared across closely related tasks and datasets (Girshick et al., 2014; Sharif Razavian et al., 2014; Oquab et al., 2014). The most successful models for transfer learning so far are those pre-trained on the ImageNet classification task (Yosinski et al., 2014).

As opposed to proposing new representation learning tasks, our work primarily studies how to get the most out of the existing tasks. Hence, our method is broadly applicable – it offers a generic framework that can be readily combined with any representation learning paradigm.

Gradients of Deep Networks. Our method makes use of the Jacobian matrix of a deep network as feature representation for a downstream task. Gradient information is traditionally employed for visualizing and interpreting convolutional networks (Simonyan et al., 2013), and more recently for generating adversarial samples (Szegedy et al., 2013), crafting defense strategies (Goodfellow et al., 2014), facilitating network compression (Sinha et al., 2018), and boosting multi-task and meta learning (Sinha et al., 2018; Achille et al., 2019).

Our work draws inspiration from Fisher vectors (FVs) (Jaakkola & Haussler, 1999)–gradient-based features from a probabilistic model (e.g. GMM). FVs have demonstrated its success for visual recognition using hand-crafted features (Perronnin & Dance, 2007). More recently, FVs have shown promising results with deep models, first as an ingredient of a hybrid system (Perronnin & Larlus, 2015), and then as task embeddings for meta-learning (Achille et al., 2019). Our method differs from the FV approaches in two folds. First, it is not built around a probabilistic model, hence has distinct theoretical motivations as we describe later. Second, our method enjoys *exact* gradient computation with respect to network parameters and allows scalable training, whereas Perronnin & Larlus (2015) extracts FVs from a probabilistic module posterior to the network, and Achille et al. (2019) employs heuristics in their method to aggressively approximate the computation of FVs.

Neural Tangent Kernel (NTK) for Wide Networks. Jacot et al. (2018) established the connection between deep networks and kernel methods by introducing the neural tangent kernel (NTK). Lee et al. (2019) further showed that a network evolves as a linear model in the infinite width limit when trained on certain losses under gradient descent. Similar ideas have been used to analyze wide deep neural networks, e.g., (Arora et al., 2019b;a; Li & Liang, 2018; Allen-Zhu et al., 2019a; Du et al., 2019; Allen-Zhu et al., 2019b; Cao & Gu, 2019; Mei et al., 2019). Our method is, to our best knowledge, the first attempt to port the theory into the regime of practical networks. In the case of binary classification, our linear model reduces to a kernel machine equipped with NTK. Instead of assuming random initialization of network parameters as all the prior works do, we for the first time evaluate the implication of pre-training on the linear approximation theory.

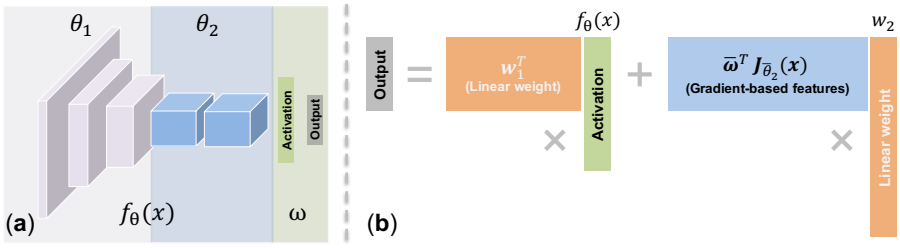

Figure 1: (**a**) An illustration of our parameterization. We consider a deep network $F(\boldsymbol{x};\theta,\boldsymbol{\omega}) \triangleq \boldsymbol{\omega}^T f_\theta(\boldsymbol{x})$ that consists of a ConvNet f_θ with its parameters $\theta \triangleq (\theta_1,\theta_2)$ and linear classifiers $\boldsymbol{\omega}$. (**b**) An overview of our proposed model. Our model takes the activation $f_\theta(\boldsymbol{x})$ and the gradients $J_{\bar{\theta}_2}(\boldsymbol{x})$ as inputs (see illustration in (a)), and learns linear weights $\boldsymbol{w}_1$ and w_2 for prediction.

3 Gradient-based Features for Representation Learning

We consider a feed-forward deep neural network $F(\boldsymbol{x};\theta,\boldsymbol{\omega}) \triangleq \boldsymbol{\omega}^T f_\theta(\boldsymbol{x})$ that consists of a convolutional backbone $f(\boldsymbol{x};\theta) \triangleq f_\theta(\boldsymbol{x})$ with its vectorized parameters θ and a linear model defined by $\boldsymbol{\omega}$ (*italic* for vectors and **bold** for matrix). Specifically, f_θ encodes the input $\boldsymbol{x}$ into a vector representation $f_\theta(\boldsymbol{x}) \in R^d$. $\boldsymbol{\omega} \in R^{d\times c}$ are thus linear classifiers that map a feature vector into c output dimensions. For this work, we focus on convolutional networks (ConvNets) for classification tasks. With trivial modifications, our method can easily extend beyond ConvNets and classification, e.g., for a recurrent network as the backbone for a regression task.

Following the setting of representation learning, we assume that a *pre-trained* $f_{\bar{\theta}}$ is given with $\bar{\theta}$ as the learned weights. The term representation learning refers to a set of learning methods that do not make use of discriminative signals from the task of interest. For example, f can be the encoder of a deep generative model ((Kingma & Welling, 2013; Dumoulin et al., 2016; Donahue et al., 2016)), or a ConvNet learned by using proxy tasks (self-supervised learning) ((Goyal et al., 2019; Kolesnikov et al., 2019)) or from another large-scale labeled dataset such as ImageNet ((Deng et al., 2009)). Given a target task, it is a common practice to regard $f_{\bar{\theta}}(\boldsymbol{x})$ as a fixed feature extractor (activation-based features) and train a set of linear classifiers, given by

$$g_{\bar{\boldsymbol{\omega}}}(\boldsymbol{x}) = \bar{\boldsymbol{\omega}}^T f_{\bar{\theta}}(\boldsymbol{x}), \tag{1}$$

We omit the bias term for clarity. Note that $\bar{\boldsymbol{\omega}}$ and $\bar{\theta}$ are instantiations of $\boldsymbol{\omega}$ and θ, where $\bar{\boldsymbol{\omega}}$ is the solution of the linear model and $\bar{\theta}$ is given by representation learning. Based on this setup, we now describe our method, discuss its theoretic implications and present an efficient training scheme.

3.1 Method Outline

Our method assumes a partition of $\theta \triangleq (\theta_1,\theta_2)$, where θ_1 and θ_2 parameterize the bottom and top layers of the ConvNet f (see Figure 1(a) for an illustration). Importantly, we propose to use *gradient*-based features $\nabla_{\bar{\theta}_2} F_{\bar{\theta},\bar{\omega}}(\boldsymbol{x}) = \boldsymbol{\omega}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x})$ in addition to *activation*-based features $f_{\bar{\theta}}(\boldsymbol{x})$. Specifically, $\boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x}) \in R^{d\times|\theta_2|}$ is the Jacobian matrix of $f_{\bar{\theta}}$ w.r.t. the pre-trained parameters $\bar{\theta}_2$ from the *top* layers of f. Formally, given the features $(f_{\bar{\theta}}(\boldsymbol{x}), \boldsymbol{\omega}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x}))$ for $\boldsymbol{x}$, our linear model $\hat{g}$, hereby considered as a classifier for concreteness, takes the form

$$\hat{g}_{\boldsymbol{w}_1,w_2}(\boldsymbol{x}) = \boldsymbol{w}_1^T f_{\bar{\theta}}(\boldsymbol{x}) + \bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x}) w_2 = g_{\boldsymbol{w}_1}(\boldsymbol{x}) + \bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x}) w_2, \tag{2}$$

where $\boldsymbol{w}_1 \in R^{d\times c}$ are liner classifiers initialized from $\bar{\boldsymbol{\omega}}$, $w_2 \in R^{|\theta_2|}$ are shared linear weights for gradient features, and $|\theta_2|$ is the size of the parameters θ_2. Both $\boldsymbol{w}_1$ and w_2 are our model parameters that need to learned from a target task. An overview of model is shown in Figure 1(b).

Our model subsumes the the linear model in Eq. (1) as the first term, and includes a second term that is linear in the gradient-based features. We note that this extra linear term is different from traditional linear classifiers as in Eq. (1). In this case, the gradient-based features forms a matrix and the linear weight w_2 is multiplied to each row of the feature matrix. Therefore, w_2 is shared for all output dimensions. Similar to linear classifiers, the output of $\hat{g}$ is further normalized by a softmax function and trained with a cross-entropy loss using labeled data from the target dataset.

Conceptually, our method can be summarized into three steps.

- **Pre-train the ConvNet $f_{\bar{\theta}}$.** This is accomplished by substituting in any existing representation learning algorithm.
- **Train linear classifiers $\bar{\boldsymbol{\omega}}$ using $f_{\bar{\theta}}(\boldsymbol{x})$.** This is a standard step in representation learning.
- **Learn the linear model $\hat{g}_{\boldsymbol{w}_1,w_2}(\boldsymbol{x})$.** A linear model of special form (in Eq. (2)) is learned using gradient-based and activation-based features. Note that our features are obtained when $\theta = \bar{\theta}$ is kept fixed, hence *requires no extra tuning of the parameters $\bar{\theta}$.*

3.2 Theoretical Insight

The key insight is that our model provides a local linear approximation to $F(\boldsymbol{x};\theta_2,\boldsymbol{\omega})$. This approximation comes from Eq. (2)–the crux of our approach. Importantly, our linear model is mathematically well motivated –it can be interpreted as the 1st-order Taylor expansion of $F_{\theta,\boldsymbol{\omega}}$ w.r.t. its parameters $(\theta_2,\boldsymbol{\omega})$ around the point of $(\bar{\theta}_2,\bar{\boldsymbol{\omega}})$. Specifically, we note that

$$\begin{aligned} F_{\theta,\boldsymbol{\omega}}(\boldsymbol{x}) &\approx \bar{\boldsymbol{\omega}}^T f_{\bar{\theta}}(\boldsymbol{x}) + \bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x})(\theta_2 - \bar{\theta}_2) + (\boldsymbol{\omega} - \bar{\boldsymbol{\omega}})^T f_{\bar{\theta}}(\boldsymbol{x}) \\ &= \boldsymbol{\omega}^T f_{\bar{\theta}}(\boldsymbol{x}) + \bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x})(\theta_2 - \bar{\theta}_2) \\ &= \hat{g}_{\boldsymbol{\omega},\theta_2-\bar{\theta}_2}(\boldsymbol{x}). \end{aligned} \tag{3}$$

With $\boldsymbol{\omega} = \boldsymbol{w}_1$ and $\theta_2 - \bar{\theta}_2 = w_2$, Eq. (2) provides a linear approximation of the deep model $F_{\theta_2,\boldsymbol{\omega}}(\boldsymbol{x})$ around the initialization $(\bar{\theta}_2,\bar{\boldsymbol{\omega}})$. $F_{\theta_2,\boldsymbol{\omega}}$ can be considered as fine-tuning both θ_2 and $\boldsymbol{\omega}$ for the target task. Our key intuition is that given a sufficiently good base network, our model will provide a linear approximation to the underlying model $F_{\theta_2,\boldsymbol{\omega}}$, and our training approximates fine-tuning $F_{\theta_2,\boldsymbol{\omega}}$.

The quality of the linear approximation can be theoretically analyzed when the base network $\bar{\boldsymbol{\omega}}^T f_{\bar{\theta}}(\boldsymbol{x})$ is sufficiently wide and at random initialization. This has been done via the recent neural tangent kernel approach (Jacot et al., 2018; Lee et al., 2019; Arora et al., 2019b) or some related ideas (Arora et al., 2019a; Li & Liang, 2018; Allen-Zhu et al., 2019a; Du et al., 2019; Allen-Zhu et al., 2019b; Cao & Gu, 2019; Mei et al., 2019). In fact, these studies are the theoretical inspiration for our approach. However, while their approximation was developed for networks with infinite or sufficiently large width at random initialization, we apply the linear approximation on pre-trained models of practical sizes. We argue that such an approximation is useful in practice for the following two critical and natural reasons:

The network f from representation learning provides a strong starting point. Thus, the pre-trained network parameter $\bar{\theta}$ is *close* to a good solution for the downstream task. Note that the key for a good linear approximation is that the output of the network is stable w.r.t. small changes in the network parameter and activation. In the existing analysis, this is proved under the conditions of large width and random base networks (see, e.g., Section 7 in (Allen-Zhu et al., 2019b) or Section B.1 in (Cao & Gu, 2019)). The pre-trained base network also has such stability properties, which are supported by empirical observations. For example, the pre-trained network has similar predictions for a significant fraction of data in the downstream task as a fine-tuned network.

The network width required for the linearization to hold decreases as data becomes more structured. An assumption made in existing analysis is that the network is sufficiently or even infinitely wide compared to the size of the dataset, so that the approximation can hold for *any* dataset. We argue that this is not necessary in practice, since the practical datasets are well-structured, and as long as the trained network is sufficiently wide compared to the effective complexity determined by the structure of the data, then the approximation can hold (Li & Liang, 2018; Allen-Zhu et al., 2019a). Our approach thus takes advantage of the bottom layers to reduce data complexity in the hope that linearization of the top (and often the widest) layers can be sufficiently accurate.

Compared to fine-tuning $F_{\theta_2,\boldsymbol{\omega}}$, our method only need to learn a linear classifier, which is efficient and straightforward. In particular, it is efficient for training and inference using our scalable training technique described below, while achieving much better performance than using activation-based features. In the most interesting setting where the task for pre-training are similar to the target tasks, the linear approximation works well, and our method achieves comparable or even better performance than fine-tuning; see the experimental section for details.

3.3 SCALABLE TRAINING

Moving beyond the theoretic aspects, a practical challenge of our method lies in the scalable training of $\hat{g}$. A naïve approach requires evaluating and storing $\bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x})$ for all $\boldsymbol{x}$, which is computationally expensive and can become infeasible as the output dimension c and the number of parameters $|\theta_2|$ grow. Inspired by Pearlmutter (1994), we design an efficient training and inference scheme for $\hat{g}$. Thanks to this scheme, the complexity of training our model using gradient-based features is on the same magnitude as training a linear classifier on activation-based features.

Central to our scalable approach is the inexpensive evaluation of the Jacobian-vector product (JVP) $\bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x}) w_2$, whose size is the same as the output dimension c. First, we note that

$$F_{\bar{\theta}+r w_2, \bar{\omega}}(\boldsymbol{x}) = F_{\bar{\theta}, \bar{\omega}}(\boldsymbol{x}) + r \bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x}) w_2 + o(r^2) \tag{4}$$

by 1st-order Taylor expansion around a scalar $r = 0$. Rearrange and take the limit of r to 0, we get

$$\bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x}) w_2 = \lim_{r \to 0} \frac{F_{\bar{\theta}+r w_2, \bar{\omega}}(\boldsymbol{x}) - F_{\bar{\theta}, \bar{\omega}}(\boldsymbol{x})}{r} = \left. \frac{\partial}{\partial r} \right|_{r=0} F_{\bar{\theta}+r w_2, \bar{\omega}}(\boldsymbol{x}), \tag{5}$$

which can be conveniently evaluated via forward-mode automatic differentiation.

More precisely, let us consider the basic building block of f–convolutional layers. These layers are defined as a *linear* function $h(\boldsymbol{z}_c; \boldsymbol{w}_c, b_c) = \boldsymbol{w}_c^T \boldsymbol{z}_c + b_c$, where $\boldsymbol{w}_c$ and b_c, which are part of θ, are the weight and bias respectively. And $\boldsymbol{z}_c$ is the input to the layer. We denote the counterparts of $\boldsymbol{w}_c$ and b_c in w_2 as $\tilde{\boldsymbol{w}}_c$ and $\tilde{b}_c$, i.e., $\tilde{\boldsymbol{w}}_c$ and $\tilde{b}_c$ are the linear weights applied to $\boldsymbol{w}_c$ and b_c. It can thus be shown that

$$\frac{\partial h(\boldsymbol{z}_c; \boldsymbol{w}_c + r\tilde{\boldsymbol{w}}_c, b_c + r\tilde{b}_c)}{\partial r} = h(\boldsymbol{z}_c; \tilde{\boldsymbol{w}}_c, \tilde{b}_c) + h(\frac{\partial \boldsymbol{z}_c}{\partial r}; \boldsymbol{w}_c, 0), \tag{6}$$

where $\frac{\partial \boldsymbol{w}_c}{\partial r}$ is the JVP coming from the upstream layer.

When a nonlinearity is encountered, we have, using the ReLU function as an example,

$$\frac{\partial ReLU(\boldsymbol{z}_c)}{\partial r} = \frac{\partial \boldsymbol{z}_c}{\partial r} \odot \mathbf{1}_{\mathrm{z}_c \geq 0}, \tag{7}$$

where $\odot$ is the element-wise product, and $\mathbf{1}$ is the element-wise indicator function and $\boldsymbol{z}_c$ is the input to the layer. Other activation functions as well as average/max pooling layers can be handled in the same spirit. For batch normalization, we fold them into their corresponding convolutions.

Importantly, Eq. (6) and (7) provide an efficient approach to compute the desired JVP in Eq. (5) by successively evaluating a set of JVPs *on the fly*. This process starts with the seed $\frac{\partial z_0}{\partial r} = 0$, where z_0 is the output of the last layer in f parameterized by θ_1 and can be pre-computed. And $\bar{\boldsymbol{\omega}}^T \boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x}) w_2$ can be computed along with the standard forward propagation through f. Moreover, during the training of $\hat{g}$, its parameters $\boldsymbol{w}_1$ and w_2 can be updated via standard back-propagation. In summary, our approach only requires a *single* forward pass through the fixed f for evaluating $\hat{g}$, and a *single* backward pass for updating the parameters $\boldsymbol{w}_1$ and w_2.

Complexity Analysis. We further discuss the complexity of our method in training and inference, and contrast our method to the fine-tuning of network parameters θ_2. Our forward pass, as demonstrated by Eq. (6) and (7), is a chain of linear operations intertwined by element-wise multiplications. And the second term in Eq. (6) forms the "main stream" of computation, while the first, "branch" term merges into the main stream at every layer of the ConvNet f. The same reasoning holds for the backward pass. Overall, our method requires twice as many linear operations as fine-tuning θ_2 of the ConvNet. Note, however, that half of the linear operations by our method are slightly cheaper due to removal of the bias term. Moreover, in the special case where θ_2 only includes the very top layer of f, our method carries out the same number of operations as fine-tuning since the second term in Eq. (6) can be dropped. For memory consumption, our method requires to store an additional "copy" (linear weights) of the model parameters in comparison to fine-tuning. As the size of θ_2 is small, this minor increase of computing and memory cost put our method on the same page as fine-tuning.

4 EXPERIMENTS

We now describe our experiments and results. Our main results are organized into two parts. First, we conduct an ablation study of our method on CIFAR-10 dataset (Krizhevsky et al., 2009). Our

Table 1: **Ablation study on CIFAR-10.** We evaluate how the choice of θ_1, θ_2 and ω influence the effectiveness of our method. All results are reported using top-1 classification accuracy. The ConvNet comes from the encoder of BiGAN, with the last 3 conv layers of width 256, 256, and 512.

			θ_2: conv5	θ_2: conv4-5	θ_2: conv3-5
Random θ_1	random θ_2	random ω	63.05	62.74	63.26
		pre-trained ω	64.61	65.00	64.85
	pre-trained θ_2	random ω	63.62	63.33	63.10
		pre-trained ω	64.38	64.31	64.13
Pre-trained θ_1	random θ_2	random ω	67.48	65.24	57.41
		pre-trained ω	69.11	67.10	62.62
	pre-trained θ_2	random ω	68.89	69.83	70.30
		pre-trained ω	**70.21**	**70.74**	**71.30**
Baseline (linear logistic regression)				63.38	
Fine-tuned θ_2 and ω			72.27	74.16	77.14
Fine-tuned θ_2 and ω + Ours			72.26	74.28	76.84

study is to dissect how different approaches of computing the gradient features influence their representational power. Moreover, we evaluate our method on three representation learning tasks: unsupervised learning using deep generative models, self-supervised learning using a pretext task, and transfer learning from large-scale datasets. We report results on several datasets and network architectures and demonstrate the strength of our method.

Implementation Details. In the rest our experiments, we use the same settings for training and evaluating our models and baselines methods, unless otherwise noticed. Concretely, we adopt the NTK parametrization (Jacot et al., 2018) for θ_2 and fold batch normalization into their preceding convolutional layers, piror to our training. For training of our method, we use a batch size of 128, learning rate of 0.001, and train until convergence using the Adam (Kingma & Ba, 2014). No weight decay or dropout is used for our method. For inference, we evaluate on a single center crop of the image for all datasets. We often refer to "baseline" as a linear classifier (logistic regression) trained on top of the last activation from the network.

4.1 Ablation study

In this study, we probe the design of our model on a set of ablations on CIFAR-10 by using a pre-trained encoder ConvNet from BiGAN (Dumoulin et al., 2016) (trained on CIFAR-10). Specifically, our gradient feature $\boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x})$ is a function of three parameters θ_1, θ_2 and ω, which can take on either random or pre-trained values. We probe the representational power of $J_{\bar{\theta}_2}(\boldsymbol{x})$ by feeding it all possible configurations of the three parameters. Moreover, we compare our results to (1) a linear classifier baseline, (2) the fine-tuning of the network and (3) our method starting from a fine-tuned model. Our results, summarized in Table 1, highlight that *the success of our method crucially depends on pre-training all three parameters*. We now provide a detailed discussion of our findings.

Random vs. pre-trained $\bar{\theta}$. Pre-training f plays a central role in the materialization of our theoretical insight. As is evidenced by our results, pre-trained θ_1 and θ_2 each injects substantial amount of information into $J_{\bar{\theta}_2}(\boldsymbol{x})$. Computing $J_{\bar{\theta}_2}(\boldsymbol{x})$ from random θ_2 leads to visible performance drop, while our method becomes virtually indistinguishable from the baseline when θ_1 is set to be random.

Random vs. pre-trained $\bar{\omega}$. We obtain better results by first training the baseline classifier (Eq. (2)), feeding its learned weights $\bar{\omega}$ into the gradient operator $\nabla_{\bar{\theta}_2} F$ for computing $\boldsymbol{J}(\mathbf{x})$, and then seamlessly transitioning to training our proposed classifier (Eq. (1)) by adding on the gradient term. Hence, the best training procedure for our model can be effectively broken down into a "pre-training" phase and a "residual learning" phase.

Optimal size of θ_2. As the size of $\boldsymbol{J}_{\bar{\theta}_2}(\boldsymbol{x})$ grows, our model's performance improves slightly albeit at the expense of extra computational overhead. Fortunately, our results suggest that it suffices to set the very top layer as θ_2 to enjoy a reasonably large performance gain.

Our method vs. fine-tuning of θ_2 **and** ω. We highlight two observations hint by our theoretical insight. First, our model, initialized with pre-trained parameters, stays close to the fine-tuned network in terms of performance, when the gradient features come from a top layer with moderate width. Second, we observe an enlarging performance gap between our model and the fine-tuned network

Table 2: **Unsupervised Learning Results**: Evaluation of our linear model using ConvNet f pre-trained with generative learning. We use top-1 classification accuracy as our evaluation metric. The top-3 conv layers (i.e. conv3-5) of our BiGAN encoder have widths 128, 256 and 512. The top-3 conv layers (i.e. conv6-8) of our VAE encoder have widths 256, 256 and 512. We refer the reader to the original papers for details of the ConvNet architectures.

Generative models / Transfer methods			**CIFAR-10**	**CIFAR-100**	**SVHN**
BiGAN	Baseline		63.38	34.45	82.24
	Our method	θ_2: conv5	69.74	37.35	90.87
		θ_2: conv4-5	70.29	**38.03**	91.15
		θ_2: conv3-5	**71.03**	37.92	**91.38**
	Fine-tuned net	θ_2 conv5	72.32	39.89	93.51
		θ_2: conv4-5	74.40	42.47	94.59
		θ_2: conv3-5	75.55	45.07	95.88
VAE	Baseline		50.67	26.41	81.63
	Our method	θ_2: conv8	61.19	33.90	91.18
		θ_2: conv7-8	63.52	36.18	92.25
		θ_2: conv6-8	**64.77**	**37.10**	**92.69**
	Fine-tuned net	θ_2 conv8	65.14	38.41	93.53
		θ_2: conv7-8	70.06	43.10	95.66
		θ_2: conv6-8	74.61	46.45	96.55

as we gather gradient features from more narrower layers towards the bottom. Furthermore, using gradient features from a fine-tuned parameters stays almost the same as the fine-tuned network.

Remarks. Our ablation studies shows that the best practice for our method is to start with pre-trained $\bar{\theta}_2$ (from representation learning) and $\bar{\omega}$ (linear classifier for the target task). Moreover, the performance of our method increases slightly as the size of θ_2 grows, i.e., linearization of more layers in f. This performance boost, however, comes with an increased computational cost, as well as an increased performance gap to the fine-tuned model. A small sized θ_2, e.g., from the parameters of last few convolutional layers seems to be sufficient.

4.2 Results on Representation Learning

We present results of our method on three different representation learning tasks: unsupervised learning, self-supervised learning and transfer learning. For all experiments in this section, we contrast at least three sets of results: a baseline linear classifier on activation-based features, our proposed linear classifier that makes use of gradient-based features w.r.t. θ_2, and a network whose θ_2 and ω is fine-tuned for the target dataset. We compare our model against the baseline to demonstrate the advantage of our gradient-based features, and against the fine-tuned network (our theoretical upper bound) to illustrate the various factors that support or break down our theoretical insights.

Unsupervised Learning using Deep Generative Models. We consider BiGAN and VAE training as the representation-learning tasks and use their encoders as the pre-trained ConvNet f. Our BiGAN and VAE models strictly follow the architecture and training setup from (Dumoulin et al., 2016) and (Berthelot et al., 2018). We average-pool the output of the ConvNet for activation-based features, and compute our gradient-based features from one, two or all of the top-3 conv layers. Training of our linear model follows the two-step process as described before. We train on the `train` split of CIFAR-10/100 and the `extra` split of SVHN, and report the top-1 classification accuracy on the `test` splits of both datasets.

Results. We summarize our results in Table 2. Our models consistently outperform the baseline across three datasets with relative improvement over 10%. Moreover, we observe good agreement of performance between our models and the fine-tuned networks. In the BiGAN case, performance of our method saturates with gradient-based features only from the very top layer.

Self-supervised Learning. We experiment with a ResNet50 pre-trained on the Jigsaw pretext task available from (Goyal et al., 2019). We refer the reader to Noroozi & Favaro (2016) for technical details. We average-pool the output of the ConvNet for activation-based features, and compute our gradient-based features from the three residual blocks in the last stage. Unlike Goyal et al. (2019) which uses linear SVM as their baseline classifier, we use a standard linear logistic regressor. We train on the `trainval` split of VOC07 and the `train` split of COCO2014 for image classification, and report the mean average precision (mAP) scores on their `test` and `val` splits respectively.

Table 3: **Self-supervise Learning Results:** Evaluation of our method using ConvNet f pre-trained via self-supervised learning. We use mean average precision (mAP) score as our evaluation metric. Our f is a ResNet-50 pre-trained on the Jigsaw pretext task. Our gradient-based features are computed w.r.t. to three bottleneck residual blocks in the last stage. The conv layers in each block have widths 2048, 512 and 2048. Unless specified, the cited experiments are AlexNet-based.

Self-supervising tasks / Transfer methods			**VOC07**	**COCO2014**
Jigsaw (ResNet50)	Baseline	Logistic regression	56.31	39.31
		Linear SVM	57.2	41.1
	Ours (θ_2: layer5 block1-3)		**62.83**	**46.35**
	Fine-tuned θ_2 (θ_2: layer5 block1-3)		72.09	59.84
Zhang et al. (2016) (ResNet50)	Linear SVM		52.3	n/a
Zhang et al. (2016)	Linear SVM		51.6	n/a
Agrawal et al. (2015)	Logistic regressor		31.2	n/a
Wang & Gupta (2015)	Logistic regressor		29.4	n/a
Doersch et al. (2015)	Logistic regressor		44.7	n/a

Results. We summarize our experimental results in Table 3. Again, we observe a large performance boost in comparison to the linear classifier baseline (over 5% on VOC07 and COCO2014). Our method also outperforms a large set of self-supervised learning methods. We note that the gap between our method and the fine-tuned network is quite large in this setting (more than 15%). We conjecture that this is due to the large semantic gap between the representation learning (jigsaw on ImageNet) and the target tasks (classification on VOC07 and COCO2014).

Table 4: **Transfer Learning Results:** Evaluation of our method using ImageNet pre-trained models f. We report mean average precision (mAP). Our f is the first 5 conv layers of an ImageNet pre-trained AlexNet. Our gradient-based features are from the last 2 conv layers (both widths are 256).

Pre-training datasets / Transfer methods		**VOC07**	**COCO2014**
ImageNet	Baseline	67.82	44.62
	Our method (θ_2: conv4-5)	**69.60**	**49.28**
	Fine-tuned net (θ_2: conv4-5)	68.85	48.80

Transfer Learning from ImageNet We now report results of transfer learning from ImageNet pre-traine models. Specifically, we start with the pyTorch distribution of ImageNet pre-trained AlexNet (Paszke et al., 2017). We remove the fully connected layers so that the remaining conv layers become our f. To obtain activation-based features, we downsample the output of f to form a 1024-dimensional feature vector. Our gradient features are computed w.r.t. the last two conv layers of f (i.e. conv4-5), both of width 256. For fine-tuning θ_2 of the network, we use the stochastic gradient descent (SGD) optimizer and apply a weight decay of 0.0005 and a momentum of 0.9. We remark that performance of the fine-tuned model gets worse otherwise due to overfitting. Our train/test splits and evaluation metric remain the same as the previous section.

Results. Our results are presented in Table 4. Our method again demonstrates strong performance on both datasets, with a notable 2% and 5% improvements on VOC07 and COCO2014 datasets in comparison to the baseline. More importantly, our method also slightly outperforms the fine-tuned network. This is a very interesting result. We argue that it is due the significant overlapping between the pre-training and target tasks. It would be interesting to identify other representation learning scenarios where our method are particularly beneficial.

5 Conclusion

In this paper, we presented a novel method for deep representation learning. Specifically, given a pre-trained model, we explored the per-sample gradients of the model parameters relative to a task-specific loss, and constructed a linear model that combines gradients of model parameters and the activation of the model. We showed that our model can be very efficient in training and inference, and provides a local linear approximation to a underlying deep model. Through a set of experiments, we demonstrated that these gradient-based features are highly discriminative for the target task, and our method can significantly improve the baseline learning of representation learning across tasks, datasets and network architectures.

REFERENCES

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. *arXiv preprint arXiv:1902.03545*, 2019.

Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in Neural Information Processing Systems*, 2019a.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pp. 242–252, 2019b.

Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pp. 322–332, 2019a.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019b.

David Berthelot, Colin Raffel, Aurko Roy, and Ian Goodfellow. Understanding and improving interpolation in autoencoders via an adversarial regularizer. *arXiv preprint arXiv:1807.07543*, 2018.

Yuan Cao and Quanquan Gu. A generalization theory of gradient descent for learning over-parameterized deep relu networks. *arXiv preprint arXiv:1902.01384*, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1422–1430, 2015.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pp. 1675–1685, 2019.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Priya Goyal, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra. Scaling and benchmarking self-supervised visual representation learning. *arXiv preprint arXiv:1905.01235*, 2019.

Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2(5):6, 2017.

Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in neural information processing systems*, pp. 487–493, 1999.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10215–10224, 2018.

Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. *arXiv preprint arXiv:1901.09005*, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *arXiv preprint arXiv:1902.06720*, 2019.

Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, pp. 8157–8166, 2018.

Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit. *arXiv preprint arXiv:1902.06015*, 2019.

Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pp. 69–84. Springer, 2016.

Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1717–1724, 2014.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Florent Perronnin and Christopher Dance. Fisher kernels on visual vocabularies for image categorization. In *2007 IEEE conference on computer vision and pattern recognition*, pp. 1–8. IEEE, 2007.

Florent Perronnin and Diane Larlus. Fisher vectors meet neural networks: A hybrid classification architecture. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3743–3752, 2015.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Ayan Sinha, Zhao Chen, Vijay Badrinarayanan, and Andrew Rabinovich. Gradient adversarial training of neural networks. *arXiv preprint arXiv:1806.08028*, 2018.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2015.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328, 2014.

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pp. 649–666. Springer, 2016.