LP++: A Surprisingly Strong Linear Probe for Few-Shot CLIP

Yunshi Huang[1*] Fereshteh Shakeri[1*]
Jose Dolz[1] Malik Boudiaf[1] Houda Bahig[2] Ismail Ben Ayed[1]

[1]ÉTS Montréal, [2]Université de Montréal

Abstract

In a recent, strongly emergent literature on few-shot CLIP adaptation, Linear Probe (LP) has been often reported as a weak baseline. This has motivated intensive research building convoluted prompt learning or feature adaptation strategies. In this work, we propose and examine from convex-optimization perspectives a generalization of the standard LP baseline, in which the linear classifier weights are learnable functions of the text embedding, with class-wise multipliers blending image and text knowledge. As our objective function depends on two types of variables, i.e., the class visual prototypes and the learnable blending parameters, we propose a computationally efficient block coordinate Majorize-Minimize (MM) descent algorithm. In our full-batch MM optimizer, which we coin LP++, step sizes are implicit, unlike standard gradient descent practices where learning rates are intensively searched over validation sets. By examining the mathematical properties of our loss (e.g., Lipschitz gradient continuity), we build majorizing functions yielding data-driven learning rates and derive approximations of the loss's minima, which provide data-informed initialization of the variables. Our image-language objective function, along with these non-trivial optimization insights and ingredients, yields, surprisingly, highly competitive few-shot CLIP performances. Furthermore, LP++ operates in black-box, relaxes intensive validation searches for the optimization hyper-parameters, and runs orders-of-magnitudes faster than state-of-the-art few-shot CLIP adaptation methods. Our code is available at: https://github.com/FereshteShakeri/FewShot-CLIP-Strong-Baseline.git.

1. Introduction

Recently, there has been a growing popularity in multimodal learning methods, which process and merge information from diverse modalities. In particular, large-scale vision-language models (VLMs), such as CLIP [23] and ALIGN [15], have attracted wide attention and made substantial progress in computer vision, showing promising generalization capabilities in various downstream tasks. Unlike conventional task-specific models that are trained with a predetermined set of labels, these so-called *foundation* models learn to align images with text, in an open-vocabulary fashion. They train, via contrastive learning, vision and text embeddings jointly using a large-scale amount of image-text pairs collected over the internet, thereby leveraging the rich semantic knowledge inherent to language (e.g., concept hierarchies). For a given downstream image classification task, vision-language embeddings enable *zero-shot* predictions, without re-training, using textual descriptions of the classes (a.k.a *prompts*). For instance, for a given class k, a textual description of the class,

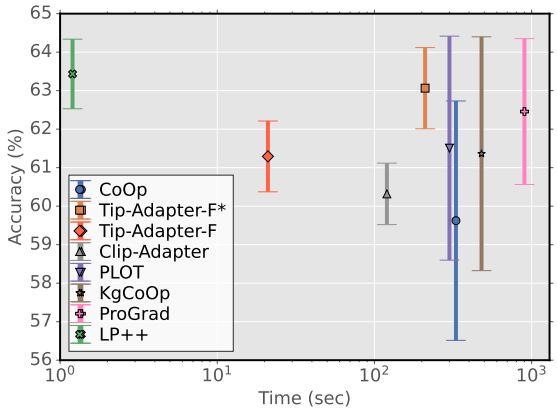

Figure 1. Comparison of LP++ with state-of-the-art few-shot CLIP methods in the 1-shot setting across 11 datasets. We compute the mean accuracy and standard deviation using 10 random tasks for each dataset. The error bars indicate the average standard deviation over all 11 datasets. The x-axis represents the run time for one task, averaged over the 11 datasets. Tip-Adapter-F and Tip-Adapter-F* are two re-implementations of Tip-Adapter-F [28], with fixed and grid-search hyper-parameters, respectively (implementation details provided in Sec. 3.2).

*Equal contribution. Correspondence to yunshi.huang@etsmtl.ca, fereshteh.shakeri.1@etsmtl.net

which we denote $\boldsymbol{z}_k$, could be "a photo of a [class$_k$]", where [class$_k$] is the class name. Thus, the zero-shot class prediction for a query image $\boldsymbol{x}$ is obtained from the cosine similarity between the l_2-normalized vision-encoded embeddings, $\boldsymbol{f} = \boldsymbol{\theta}_v(\boldsymbol{x})$, and the text-encoded ones, $\boldsymbol{t}_k = \boldsymbol{\theta}_t(\boldsymbol{z}_k)$: $\hat{k} = \arg\max_k \boldsymbol{f}^t \boldsymbol{t}_k$, where t denotes the transpose[1].

Motivated by the observation that the choice of input prompts $\boldsymbol{z}_k$ may affect the zero-shot predictions, and following on from the strong recent emergence of *prompt learning* research in the NLP community [14, 16, 24], the popular work in [29] pioneered *context optimization* (CoOp) for vision-language models. CoOp models input text $\boldsymbol{z}_k$ as learnable continuous vectors, e.g., in the form $\boldsymbol{z}_k = (\boldsymbol{z}_k^1, \ldots, \boldsymbol{z}_k^M, [\text{class}_k])$, where $(\boldsymbol{z}_k^l)_{1 \leq l \leq M}$ are *learnable* text tokens, [class$_k$] is a fixed token corresponding to the word embedding vector of the name of the k^{th} class, and M is a hyper-parameter. These learnable vectors are fine-tuned as task-specific prompts using few-shot training examples and a standard supervised classification loss. More specifically, in this few-shot setting, we assume access to a set consisting of a few labeled samples for each target class, often referred to as the *support set*. Let $\boldsymbol{f}_i = \boldsymbol{\theta}_v(\boldsymbol{x}_i)$ denote the vision embedding of support image i, and y_{ik} its one-hot encoded label, i.e., $y_{ik} = 1$ if image $\boldsymbol{x}_i$ belongs to class k and 0 otherwise. Expressing the text embeddings as $\boldsymbol{t}_k = \boldsymbol{\theta}_t(\boldsymbol{z}_k^1, \ldots, \boldsymbol{z}_k^M, [\text{class}_k])$, CoOp fine-tunes text tokens $(\boldsymbol{z}_k^l)_{1 \leq l \leq M}$ by minimizing the cross-entropy (CE) loss, with N labeled support samples and K classes[2]:

$$-\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \ln p_{ik} \tag{1}$$

where the softmax predictions p_{ik} and the logits (class scores) l_{ik} are given by:

$$p_{ik} = \frac{\exp(l_{ik})}{\sum_{j=1}^{K} \exp(l_{ij})}; \quad l_{ik} = \boldsymbol{f}_i^t \boldsymbol{t}_k$$

Although recent, the pioneering idea of CoOp has triggered a quite abundant literature on prompt learning for few-shot vision-language models, with numerous, more convoluted extensions, e.g. [4, 27, 30], to list a few. For instance, PLOT [4] followed up by learning multiple prompts, to describe the characteristics of each class, via minimizing an optimal-transport distance. KgCoOp [27] improves CoOp's performance when dealing with unseen classes, via minimizing the discrepancy between the text embeddings generated by the learned prompts and hand-crafted ones. While CoOp directly updates the context vectors using the CE loss, ProGrad [30] aligns the few-shot downstream knowledge with the large-scale general knowledge, thus mitigating the overfitting of the few-shot samples.

Prompt learning methods have brought significant improvements over zero-shot classification, but they come at the price of heavy computational and memory load, as they require gradient back-propagation through the entire text encoder. Furthermore, they assume knowledge of the text encoder. These aspects may impede their deployment in low-resource and black-box, privacy-preserving scenarios, which are of wide interest in practice. Indeed, in NLP, there is currently an emerging literature on fast few-shot adaptation of black-box models [6], strongly motivated by the fact that large-scale foundation models (e.g., the GPT family, Anthropic's Claude or Google's PaLM) are only available through APIs and their pre-trained weights are not shared. Finally, by evaluating prompt learning methods over larger numbers of sampled support sets in our experiments (Fig. 1), we observed that they exhibit large variation in performances. This could be explained by the fact that, through the text encoder, they learn prompts that are ''too specialized" for a given image support set.

While prompt learning alters the textual inputs, another category of approaches, referred to as *adapters*, focused on transforming the pre-training features of the visual or language encoders, e.g., [9, 28]. These adapters are *non-linear* transformations, for instance, in the form of multi-layer modules, added to the encoder's bottleneck. They learn additional transformations, yielding logits of the form:

$$l_{ik} = \boldsymbol{\theta}_a(\boldsymbol{f}_i, \boldsymbol{t}_k) \tag{2}$$

The adapter's learnable parameters, $\boldsymbol{\theta}_a$, are fine-tuned over a few-shot task by optimizing the cross-entropy loss, similarly to (1) but with logits l_{ik} expressed as functions of $\boldsymbol{\theta}_a$. For instance, the popular CLIP-Adapter [9] integrated a multi-layered perceptron to modify the features, along with residual connections, which enable blending with the original pre-trained features. Tip-Adapter [28] added a non-linear, quadratic-complexity module, which evaluates the pairwise similarities between the features of the support sets, and blends the ensuing class scores with the original textual features. This category of approaches mitigates some of the limitations of prompt-learning methods as they result in few-shot adaptation that has significantly lower computation and memory loads. However, as shown in Fig. 1 and in our experiments, their performances seem to depend strongly on some key hyper-parameters that have to be adjusted carefully on each downstream task, e.g. those that control the blending between the vision and language features. Therefore, to perform competitively, they incur an additional computation overhead to the adaptation phase, due to intensive (e.g., grid) search of the hyper-parameters over task-dedicated validation sets.

In the above-mentioned, strongly emergent literature on

[1] For l_2-normalized feature embeddings, the dot product corresponds to the cosine similarity.

[2] The number of labeled support samples per class, $S = \frac{N}{K}$, is small, typically in $\{1, 2, 4, ..., 16\}$.

few-shot CLIP adaptation, *linear probe* (LP) [23] has been often reported as a very weak baseline. For instance, in the 1-shot setting, it scores near 20% lower than the zero-shot predictions averaged over 11 benchmarks (Table 1). Initially evaluated in [23], LP is a linear classifier on the vision-encoded features. Specifically, it optimizes the CE loss (1) w.r.t the last-layer weights of the vision encoder (i.e., the class prototypes), which we will denote $(\boldsymbol{w}_k)_{1\leq k\leq K}$ in the rest of the paper, with the logits given by: $l_{ik} = \boldsymbol{f}_i^t \boldsymbol{w}_k$. *A clear deficiency in this standard LP baseline is that it omits completely the language knowledge of CLIP, i.e.,* $(\boldsymbol{t}_k)_{1\leq k\leq K}$.

In this work, we propose and examine from convex-optimization perspectives a generalization of the standard LP baseline. Specifically, we extend the logits in the CE loss in (1), so that they become learnable functions of the text embedding:

$$l_{ik} = \boldsymbol{f}_i^t(\boldsymbol{w}_k + \alpha_k \boldsymbol{t}_k)$$

with $(\alpha_k)_{1\leq k\leq K}$ trainable class-wise parameters blending image and text knowledge. As our objective function depends on two types of variables, i.e., the visual class prototypes $(\boldsymbol{w}_k)_{1\leq k\leq K}$ and blending parameters $(\alpha_k)_{1\leq k\leq K}$, we propose a computationally efficient Block Majorize-Minimize (BMM) procedure. In our full-batch MM optimizer, which we coin LP++, step sizes are implicit in the definition of the majorizing functions, unlike standard gradient descent practices where learning rates are intensively searched over validation sets. Moreover, we examine the mathematical properties of our objective, i.e., (i) Lipschitz gradient continuity and (ii) decomposition into convex functions having closed-form optima. This enabled us to build majorizing functions yielding data-driven learning rates, and to derive approximations of the objective-function minima, which yield data-informed initializations of the variables. Our image-language objective function, along with these non-trivial optimization insights and ingredients, yield, surprisingly, highly competitive few-shot CLIP performances (Fig. 1). Furthermore, LP++ operates in black-box, relaxes intensive validation searches for the optimization of hyper-parameters, and runs orders-of-magnitudes faster than state-of-the-art few-shot CLIP methods (Table 4). For instance, for 16-shot ImageNet adaptation, it takes seconds on a single NVIDIA RTX A600 GPU.

2. Formulation of LP++

Following the notations introduced in the previous section, we propose to minimize the following CE objective function w.r.t visual class prototypes $\mathbf{w} = (\boldsymbol{w}_k)_{1\leq k\leq K}$ and class-wise blending parameters $\boldsymbol{\alpha} = (\alpha_k)_{1\leq k\leq K}$:

$$L(\mathbf{w}, \boldsymbol{\alpha}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \ln p_{ik}(\mathbf{w}, \boldsymbol{\alpha}) \qquad (3)$$

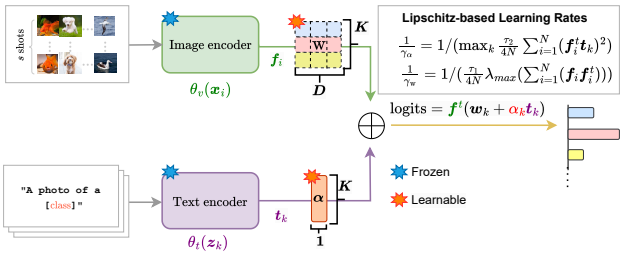

Figure 2. **Visualization of LP++.**

where softmax probability outputs p_{ik} are now given by:

$$p_{ik}(\mathbf{w}, \boldsymbol{\alpha}) = \frac{\exp\left(\boldsymbol{f}_i^t(\boldsymbol{w}_k + \alpha_k \boldsymbol{t}_k)\right)}{\sum_{j=1}^{K}\exp\left(\boldsymbol{f}_i^t(\boldsymbol{w}_j + \alpha_j \boldsymbol{t}_j)\right)} \qquad (4)$$

and $\boldsymbol{t}_k$ are the fixed pre-trained embeddings of the text templates used in zero-shot CLIP [23]. Clearly, the objective function defined by (3) and (4) could be viewed as a generalization of the vision-encoder CE loss used in the standard LP [23]. Indeed, the latter corresponds to setting $\alpha_k = 0\,\forall k$, i.e., no text knowledge. As we will see in our experimental ablation over different objective functions (Table 2), introducing the text knowledge ($\alpha_k > 0\,\forall k$) has a substantial effect on performances. Also, making α_k learnable (rather than fixed) leads to a further significant impact. Indeed, we hypothesize that the optimal blending of the text and visual knowledge is task dependent, which motivates learning it from the context of the support set.

2.1. Block coordinate Majorize-Minimize descent

Majorize-Minimize (MM) [18] is a very general optimization principle, which includes different classes of standard optimizers such as gradient descent, concave-convex procedures and expectation-maximization. Let $\mathbf{v} = (\mathbf{w}, \boldsymbol{\alpha}) \in \mathbb{R}^{K(D+1)}$ denote the overall vector of variables in our case, with D being the dimension of the feature embeddings. At each iteration, the MM procedure updates the variable as the minimum of a *majorizing* function, i.e., an upper bound on the original objective, which is tight at the current iteration j: $L(\mathbf{v}) \leq M(\mathbf{v}, \mathbf{v}^j)$ and $L(\mathbf{v}^j) = M(\mathbf{v}^j, \mathbf{v}^j)$. Thus, update step $\mathbf{v}^{j+1} = \min_{\mathbf{v}} M(\mathbf{v}, \mathbf{v}^j)$ guarantees that the original objective does not increase at each iteration[3]: $L(\mathbf{v}^{j+1}) \leq M(\mathbf{v}^{j+1}, \mathbf{v}^j) \leq M(\mathbf{v}^j, \mathbf{v}^j) = L(\mathbf{v}^j)$. Therefore, in MM algorithms, step sizes are *implicit* in the definition of the majorizing function, unlike standard gradient-descent practices, in which the step sizes (*a.k.a* learning rates) are intensively searched over validation sets, via running the optimizer several times.

In this work, we exploit the Lipschitz-gradient continuity of our convex objective in (3), i.e., bounds on the

[3]This assumes, of course, that minimizing $M(\mathbf{v}, \mathbf{v}^j)$ over $\mathbf{v}$ could be solved to global optimality and is easier than the original problem.

maximum eigen values of the Hessian matrices (Prop. 1), thereby building majorizing functions with data-driven, task-specific step sizes. This removes the need for validation searches for the optimization hyper-parameters, reducing the computational load for fine-tuning (Table 2), while yielding performances on par with the best learning rates found with validation (Fig. 4). Also, interestingly, the Lipschitz-based learning rates computed from our derivation in Prop. 1 are orders-of-magnitude larger than those typically used in deep learning, yielding steeper decreases towards the minimum. Before giving proper majorizing functions for our convex, gradient-Lipschitz function in (3), let us first point to the following results, well-known in convex optimization [3]. While these results are text-book knowledge in optimization, they enable to connect the general MM principle to gradient descent, motivating the data-driven, task-specific step sizes we derive in Prop. 1 and the block-coordinate MM optimizer we propose in Alg. 1.

Lemma 2.1. *([3, p. 268]) Assume $L(\mathbf{v})$ is a twice-differentiable function, which has a Lipschitz continuous gradient, i.e., there exists a strictly positive Lipschitz constant γ such that $\nabla^2 L(\mathbf{v}) \preceq \gamma \mathbf{I}$, with $\mathbf{I}$ the identity matrix. Then, the following quadratic bound is a majorizing function for L at iteration j:*

$$M(\mathbf{v}, \mathbf{v}^j) = L(\mathbf{v}^j) + \nabla L(\mathbf{v}^j)^t(\mathbf{v} - \mathbf{v}^j) + \frac{\gamma}{2}\|\mathbf{v} - \mathbf{v}^j\|^2 \quad (5)$$

Furthermore, a specific gradient step, with learning rate $\frac{1}{\gamma}$ minimizes M, i.e., $\mathbf{v}^{j+1} = \mathbf{v}^j - \frac{1}{\gamma}\nabla L(\mathbf{v}^j) = \arg\min_{\mathbf{v}} M(\mathbf{v}, \mathbf{v}^j)$, and guarantees that objective L decreases by at least $\frac{1}{2\gamma}\|\nabla L(\mathbf{v})\|^2$:

$$L(\mathbf{v}^{j+1}) \leq L(\mathbf{v}^j) - \frac{1}{2\gamma}\|\nabla L(\mathbf{v})\|^2 \quad (6)$$

Moreover, the following Theorem, which follows from Lemma 2.1, establishes the *sublinear convergence* of the MM procedure using bound (5), i.e., a convergence rate of $O(1/J)$, J being the total number of iterations.

Theorem 2.2. *([3, p. 267]) For convex, twice-differentiable function $L(\mathbf{v})$, which has a γ-Lipschitz gradient, performing J updates $\mathbf{v}^{j+1} = \mathbf{v}^j - \frac{1}{\gamma}\nabla L(\mathbf{v}^j)$, starting from initialization $\mathbf{v}^0$, will yield a solution that satisfies:*

$$\|L(\mathbf{v}^J) - L(\mathbf{v}^*)\| \leq \frac{\gamma}{2J}\|\mathbf{v}^0 - \mathbf{v}^*\| \quad (7)$$

where $L(\mathbf{v}^*)$ is the optimal value.

For completeness, we provide the proofs of these well-known results in the supplemental material. Clearly, Lemma 2.1 and Theorem 2.2 prescribe a learning rate of $\frac{1}{\gamma}$ for a function that has a γ-Lipschitz gradient. One valid Lipschitz constant would be the maximum eigen value of the Hessian of our objective in (3), which provides a majorizing function of the form in Eq. (5) and data-driven learning rates. However, a naive spectral decomposition of the Hessian matrices (to obtain the maximum eigen value) could be computationally intensive. For instance, for ImageNet, the Hessian of our objective is of size $K(D+1) \times K(D+1) \approx 1M \times 1M$, as $D = 1024$ and $K = 1000$. In Prop. 1, we derive approximate global and block-wise Lipschitz constants that can be computed efficiently (i.e., evaluating the maximum eigen value of a single $D \times D$ matrix).

Block-coordinate updates Our procedure provided in Alg. 1 belongs to the family of Block Majorize-Minimize (BMM) methods, well studied in the optimization community [13]. To minimize a multi-block objective, as in our case where the blocks correspond to variables $\mathbf{w}$ and $\boldsymbol{\alpha}$, we minimize one or many successive majorizing functions of the objective in each block, with the other block fixed, in a cyclic order:

$$L(\mathbf{w}^j, \boldsymbol{\alpha}) + \nabla L_{\mathbf{w}}(\mathbf{w}^j)^t(\mathbf{w} - \mathbf{w}^j) + \frac{\gamma_{\mathbf{w}}}{2}\|\mathbf{w} - \mathbf{w}^j\|^2 \quad (8)$$

$$L(\mathbf{w}, \boldsymbol{\alpha}^j) + \nabla L_{\boldsymbol{\alpha}}(\boldsymbol{\alpha}^j)^t(\boldsymbol{\alpha} - \boldsymbol{\alpha}^j) + \frac{\gamma_{\boldsymbol{\alpha}}}{2}\|\boldsymbol{\alpha} - \boldsymbol{\alpha}^j\|^2 \quad (9)$$

where in (8), block $\boldsymbol{\alpha}$ is fixed and, in (9), $\mathbf{w}$ is fixed. $\nabla L_{\mathbf{w}}$ and $\nabla L_{\boldsymbol{\alpha}}$ denote block-wise gradients, and $(\gamma_{\mathbf{w}}, \gamma_{\boldsymbol{\alpha}})$ are the *block Lipschitz constants*. Accommodating different choices of the block-cycling strategies and majorizing functions, BMM includes a breadth of optimizers as particular cases, such the well-known block coordinate gradient descent (BCGD) [1] and its projection-based variant. Indeed, the so-called Gauss-Seidel cycling [13] alternates steps (8) and (9), which corresponds to the BCGD method. One could also performs many successive steps in one block, as we do in Alg. 1, which corresponds to the so-called Essentially-Cyclic[4] strategy [13]. Importantly, for a fairly large spectrum of choices of the cycling strategies, BMM enjoys the same *sublinear convergence* property as MM for convex objectives (with each block-wise update decreasing the objective), provided that the majorizing functions are strongly convex; see Theorem 3.1 in [13]. This is the case for (8) and (9). In our experiments, we observed that this block-wise variant performs better than a single-block MM; see Table 2. This might be explained by the fact that each block of variables has a dedicated step size. In the supplemental material, we provide results for different block-cycling strategies.

Global and block-coordinatewise Lipschitz constants In the following, we derive approximate global and block-

[4]Essentially-Cyclic means that there is a period during which each block is updated at least once.

coordinatewise Lipschitz constants for our objective function in (3), which could be evaluated efficiently. We deploy these in Alg. 1, to compute data-driven, block-wise learning rates for updating visual prototypes $\mathbf{w}$ and blending parameters $\boldsymbol{\alpha}$.

Proposition 1. *Considering the blocks of variables $\mathbf{w}$ and $\boldsymbol{\alpha}$, the gradient of our objective L in (3) is block-coordinatewise Lipschitz continuous. For $\tau_1 \geq 2$, it has the following block Lipschitz constant for the set of variables in $\mathbf{w}$:*

$$\gamma_{\mathbf{w}} = \frac{\tau_1}{4N} \lambda_{max} \left(\sum_{i=1}^{N} (\boldsymbol{f}_i \boldsymbol{f}_i^t) \right) \tag{10}$$

where $\lambda_{max}(\boldsymbol{A})$ denotes the maximum eigenvalue of matrix $\boldsymbol{A}$. Furthermore, for $\tau_1 \geq 1$, the expression in Eq. (10) provides a tighter but approximate block Lipschitz constant. Similarly, for $\tau_2 \geq 1$, we have following approximate block Lipschitz constant for the variables in $\boldsymbol{\alpha}$:

$$\gamma_{\boldsymbol{\alpha}} = \max_k \frac{\tau_2}{4N} \sum_{i=1}^{N} (\boldsymbol{f}_i^t \boldsymbol{t}_k)^2 \tag{11}$$

Finally, for $\tau \geq 2$, the following expression provides an approximate global Lipschitz constant for objective (3), i.e., w.r.t all variables $\mathbf{v} = (\mathbf{w}, \boldsymbol{\alpha})$:

$$\gamma = \tau \max(\gamma_{\mathbf{w}}, \gamma_{\boldsymbol{\alpha}}) \tag{12}$$

Proof. The details are deferred to the supplemental material. The main ingredients of the proof are based on the Gershgorin circle theorem and the variational characterization of the maximum eigenvalue, following the min-max theorem, also referred to as the variational principle. □

2.2. Initialization of the variables

In the following Prop. 2, we derive approximations of the minima of our objective function (3), which yield a data-informed initialization of the variables. Indeed, the expressions we obtain in Eqs. (13) and (15) suggest initial guesses for variables $\mathbf{w}$ and $\boldsymbol{\alpha}$. Interestingly, and as will be confirmed by our experiments (Table 3), such an initialization yields substantially lower values of the minimized loss than a random initialization. Furthermore, surprisingly, using this initialization for a *training-free* prediction yields better performances than the training-free version of the recent Tip-Adapter-F approach [28]. The details of our training-free version are provided in the supplemental material.

Proposition 2. *The cross-entropy in Eq. (3) could be written as the sum of two convex functions, i.e., $L = g_1 + g_2$, such that, $\forall k \in [1, \dots, K]$, the minimum of g_1 w.r.t $\boldsymbol{w}_k$ is co-linear to the hard mean vector of features within class k:*

$$\arg\min_{\boldsymbol{w}_k} g_1 = \frac{1}{\lambda N} \sum_{i=1}^{N} y_{ik} \boldsymbol{f}_i \propto \frac{\sum_{i=1}^{N} y_{ik} \boldsymbol{f}_i}{\sum_{i=1}^{N} y_{ik}} \tag{13}$$

and the minimum of g_2 w.r.t $\boldsymbol{w}_k$ is co-linear to the soft mean vector of features within class k:

$$\arg\min_{\boldsymbol{w}_k} g_2 = \frac{1}{\lambda N} \sum_{i=1}^{N} p_{ik} \boldsymbol{f}_i \propto \frac{\sum_{i=1}^{N} p_{ik} \boldsymbol{f}_i}{\sum_{i=1}^{N} p_{ik}} \tag{14}$$

where $\lambda \leq \min_k \lambda_{min}(\boldsymbol{A}_k)$, $\boldsymbol{A}_k = \frac{1}{N} \sum_{i=1}^{N} (p_{ik} - p_{ik}^2) \boldsymbol{f}_i \boldsymbol{f}_i^t$ and $\lambda_{min}(\boldsymbol{A})$ denotes the smallest eigenvalue of matrix $\boldsymbol{A}$.

Proof. We defer the details, including the full expressions of convex functions g_1 and g_2, to the supplemental material. □

Similarly to the development in Prop. 2, one could decompose (3) as the sum of two convex functions, i.e., $L = h_1 + h_2$, such that, $\forall k \in [1, \dots, K]$, the minima of h_1 and h_2 w.r.t α_k could be written, up to a multiplicative positive factor, as the hard and soft means of the cosine similarities between the image and text embeddings:

$$\arg\min_{\alpha_k} h_1 = \frac{1}{\beta N} \sum_{i=1}^{N} y_{ik} \boldsymbol{f}_i^t \boldsymbol{t}_k \tag{15}$$

$$\arg\min_{\alpha_k} h_2 = \frac{1}{\beta N} \sum_{i=1}^{N} p_{ik} \boldsymbol{f}_i^t \boldsymbol{t}_k \tag{16}$$

Here, $\beta = \min_k \frac{1}{N} \sum_{i=1}^{N} (p_{ik} - p_{ik}^2) (\boldsymbol{f}_i^t \boldsymbol{t}_k)^2$, and h_1 (respectively h_2) has the same expression as g_2 (respectively g_1), except that term $\frac{\lambda}{2} \sum_{k=1}^{K} \|\boldsymbol{w}_k\|^2$ is replaced by $\frac{\beta}{2} \|\boldsymbol{\alpha}\|^2$; see the supplemental material for the expressions of g_1 and g_2.

Algorithm 1: Block coordinate MM ($\mathbf{w}$, $\boldsymbol{\alpha}$)

iter$_{\mathbf{w}}$ = 10; iter$_{\boldsymbol{\alpha}}$ = 1; $\tau_1 = 1$; $\tau_2 = 16$; $\lambda = \frac{1}{N}$; $\beta = \frac{1}{250K}$
Initialize $\mathbf{w}^{0,0}$ // Using (13) for each k
Initialize $\boldsymbol{\alpha}^{0,0}$ // Using (15) for each k
for $j = 0, 1, \dots$ **do**
  for $l_1 = 0, 1, \dots,$ iter$_{\mathbf{w}}$ **do**
    $\mathbf{w}^{j,l_1+1} = \mathbf{w}^{j,l_1} - \frac{1}{\gamma_{\mathbf{w}}} \nabla L_{\mathbf{w}}(\mathbf{w}^{j,l_1}, \boldsymbol{\alpha}^{j,0})$
    // $\frac{1}{\gamma_{\mathbf{w}}}$ from Eq. (10)
  $\mathbf{w}^{j+1,0} = \mathbf{w}^{j,\text{iter}_{\mathbf{w}}}$
  for $l_2 = 0, 1, \dots,$ iter$_{\boldsymbol{\alpha}}$ **do**
    $\boldsymbol{\alpha}^{j,l_2+1} = \boldsymbol{\alpha}^{j,l_2} - \frac{1}{\gamma_{\boldsymbol{\alpha}}} \nabla L_{\boldsymbol{\alpha}}(\mathbf{w}^{j+1,0}, \boldsymbol{\alpha}^{j,l_2})$
    // $\frac{1}{\gamma_{\boldsymbol{\alpha}}}$ from Eq. (11)
  $\boldsymbol{\alpha}^{j+1,0} = \boldsymbol{\alpha}^{j,\text{iter}_{\boldsymbol{\alpha}}}$

Number of shots (S)		1	2	4	8	16
	Zero-shot CLIP ICML'21[23]			58.89		
Prompt-Learning	CoOp IJCV'22[29]	59.62 ± 3.11	63.80 ± 2.32	67.23 ± 1.64	71.30 ± 0.86	74.06 ± 0.55
	PLOT ICLR'23[4]	61.51 ± 2.91	65.67 ± 2.06	68.39 ± 1.17	71.96 ± 0.70	74.35 ± 0.66
	KgCoOp CVPR'23[27]	61.36 ± 3.04	63.23 ± 2.06	65.73 ± 1.15	67.50 ± 1.11	69.01 ± 0.79
	ProGrad ICCV'23[30]	62.46 ± 1.89	65.88 ± 1.46	68.52 ± 1.15	71.82 ± 0.11	73.95 ± 0.68
CLIP-based Adapters	CLIP-Adapter IJCV'23[9]	60.32 ± 0.80	61.93 ± 0.93	65.12 ± 0.80	69.20 ± 0.56	72.57 ± 0.54
	Tip-Adapter-F ECCV'22[28]	61.29 ± 0.92	62.94 ± 0.75	66.02 ± 0.80	69.88 ± 0.51	73.82 ± 0.55
	Tip-Adapter-F* ECCV'22[28]	63.06 ± 1.05	**66.47** ± 0.65	68.71 ± 0.96	71.78 ± 1.00	74.37 ± 0.35
Linear-Probing	Standard LP ICML'21[23]	36.10 ± 1.43	46.99 ± 1.29	56.72 ± 1.20	64.66 ± 0.55	70.56 ± 0.44
	LP++	**63.43** ± 0.90	66.20 ± 0.72	**69.16** ± 0.79	**72.04** ± 0.46	**74.42** ± 0.45

Table 1. **Comparison to state-of-the-art methods.** Average classification accuracy (%) on 11 benchmarks, with standard derivation over 10 sampled support sets for each dataset. The best values are highlighted in bold.

3. Experiments

3.1. Datasets and Implementation details

Following the CLIP-based few-shot adaptation literature [27, 28], we conduct the main experiments on 11 public classification data sets: Caltech101 [8], ImageNet [7], DTD [5], OxfordPets [22], Flowers102 [20], StandfordCars [17], Food101 [2], FGVCAircraft [19], SUN397 [26], EuroSAT [11] and UCF101 [25]. We follow standard practices [23] and consider $S = \{1, 2, 4, 8, 16\}$ shots for model adaptation, which are randomly sampled for each data set.

Towards a fair validation set. Apparently, prior works on this problem [28] have resorted to a large set of validation samples to adjust their hyper-parameters. For the sake of fairness, we tune the hyper-parameters across all the methods based on a small validation set, which contains as many samples (i.e., shots) as the training set. Furthermore, to avoid the potential overfitting on the few training samples, we adopt an early stopping strategy on this validation set.

General Setting. While the existing works evaluate methods based on either a single or three random tasks (support sets) [28, 29], we found that, for some datasets, the chosen support samples may not be representative of the class, leading to large standard deviations in low-shot scenarios (see Fig. 1). To ensure fair comparisons, we evaluate all the methods by averaging their classification accuracies over 10 randomly sampled tasks, for each dataset. In all the experiments, we employ ResNet-50 [10] as the visual encoder for the CLIP backbone. *It is important to note that, for our BMM procedure in Alg. 1, the optimizer hyper-parameters remain fixed across all the datasets*. We use the validation set only to find the best model, via a single run of our BMM procedure with a fixed number of variable updates, i.e., 300 gradient updates including all the blocks of variables.

3.2. Baselines

We benchmark the proposed LP++ against relevant state-of-the-art methods in the few-shot adaptation of CLIP-based models. We first resort to zero-shot CLIP as the standard baseline, which only leverages the knowledge learned by the pre-trained CLIP model. Also, we includes the standard LP baseline, whose implementation is done following [23, 29]. More concretely, this baseline optimizes the standard cross-entropy loss, which corresponds to $\alpha_k > 0 \, \forall k$ in our generalization in (3), using the L-BFGS [21] optimizer[5]. It also includes an l_2-regularizer, whose balancing weight is set based on the validation set.

CLIP-based adapters. We benchmark LP++ against two popular adapter-based few-shot approaches: CLIP-adapter [9] and TIP-adapter [28]. As exposed earlier, several popular works follow unfair practices by resorting to a larger validation set, or even to the entire test set, to adjust their key hyperparameters –as well as their model selection criteria (i.e., epochs)– for each task. For the sake of fairness, we re-implement Tip-Adapter-F and report the results in 2 different settings. In the first setting (Tip-Adapter-F), we set the two crucial hyper-parameters of this method to 1, keeping them fixed during training, and adopt early stopping based on the validation set. In the second setting, referred to as Tip-Adapter-F*, we perform intensive grid-search on the validation set to find the best values for these hyperparameters at initialization, which incurs an additional time complexity burden compared to Tip-Adapter-F.

Prompt learning. We further compare the proposed LP++ to relevant prompt-learning methods, including CoOp[29] and more recent variants, such as PLOT[4], KgCoOp[27] and ProGrad[30]. We also apply early stopping here based on the performances on the validation set.

3.3. Results

Comparison to the state-of-the-art In Table 1, we present the quantitative results obtained by LP++ (Alg. 1)

[5]L-BFGS (Limited-memory BFGS) aims to find the minimum of objective function using the second order method. It estimates the Hessian matrix based on recent gradients only, enabling it to determine the steepest direction for achieving the optimal solution. Additionally, it is implemented with line searches to automatically determine the optimal step size.

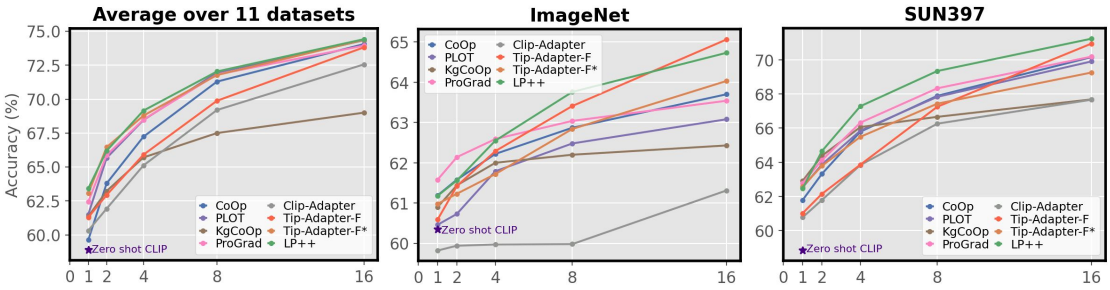

Figure 3. Quantitative performance of different adaptation methods on the 11 benchmarks (mean), as well as in two other datasets, averaged over 10 tasks (additional figures on the remaining 9 datasets can be found in Appendix, Sec. 16).

and the relevant literature in the task of efficient adaptation of VLMs. We report the average classification accuracy and standard deviation, across 11 classification benchmarks. From these results, one can make several observations. First, while the standard LP baseline largely underperforms the existing adaptation methods, our improved version, LP++, brings significant performance gains, particularly in the low-labeled data regimes. It is important to stress that the standard LP baseline integrates only the visual features extracted from CLIP, disregarding the text-encoder knowledge. This contrasts with the existing adapters, which leverage both image and text information. Therefore, these results evidence that **the potential of LP has been severely underestimated in the existing literature.** Second, when the model selection process is performed fairly, i.e., using a small validation set, their performances fall behind the proposed method (from 1% to 5%), despite being arguably more complex approaches. In particular, among the adapter-based strategies, only Tip-Adapter-F* yields performances on par with LP++, but at the cost of increasing computational load, due to an additional intensive grid-search over its hyper-parameters. If we look at prompt-learning methods, ProGrad and PLOT might be considered as competitors of LP++, particularly as the number of shots increases. Nevertheless, as already discussed, these methods are computationally inefficient compared to adapters, and do not enable black-box adaptation. The overall performance across all the methods is depicted in Fig. 3 for the average over the 11 benchmarks, as well as in two datasets, showing that LP++ typically outperforms existing methods under different few-shot scenarios. A more detailed analysis is deferred to the Appendix, Sec. 16.

Ablation on the loss functions and different optimization strategies. Table 2 reports the test accuracy and run time for different optimizers and loss functions, including the standard CE loss ($\alpha_k = 0\,\forall k$) and our loss with learnable blending parameters. Additionally, for LP++, we evaluate our loss with fixed blending parameters ($\alpha_k = 1\,\forall k$). Independently of the optimizer used, the main takeway from Table 2 is that introducing the text knowledge and making α_k learnable (rather than fixed) have a substantial impact on accuracy. As for evaluating the optimizers, and for a fair comparison, we use a fixed budget for the number of variable updates for all the optimizers (i.e., 300 updates). In the case of LP++, this corresponds to the total number of updates for all the blocks. Also, for all optimizers, we initialize $\mathbf{w}$ and $\boldsymbol{\alpha}$ following Eqs. (13) and (15). We first consider two popular optimizers, i.e., GD and ADAM, and deploy them in two different settings. First, we run each optimizer 7 times, with each run corresponding to a learning rate in the range $[10^{-4}, 10^{2}]$. Then, we record the best performances obtained on the validation set; see GD (optimum) and ADAM (optimum) in Table 2. This follows the standard practices in deep learning, i.e., searching for the learning rates over validation sets, which incurs additional computation overhead; see the time column in Table 2. Second, we run GD and ADAM with our data-driven learning rate, as prescribed by the approximate Lipschitz constant we derived in Eq. (12), with $\tau = 1$; see GD (our Lipschitz cst) and ADAM (our Lipschitz cst) in Table 2. Note that, in this case, GD corresponds to LP++ with a single block. Furthermore, we include the L-BFGS in these comparisons, with its initial learning rate set to 1. For L-BFGS, implementing a line search for the optimal step size also introduces an additional computational overhead. As highlighted by Table 2, our method removes the need for validation searches for the optimization hyper-parameters, thanks to its data-driven, task-specific step sizes, thereby reducing the computational load for fine-tuning. In the meanwhile, it yields performances on par with those obtained with the best learning rates found with the validation set. In Fig. 4, we plot the performances of single-block GD vs. the learning rates in the range $[10^{-4}, 10^{2}]$, for three datasets. We observe that our Lipschitz-based, task-specific step sizes match the optimal ones found on the validation set, although these vary among the datasets. Also, interestingly, these Lipschitz-based step sizes are orders-of-magnitude larger than those used in deep learning, which are, typically, within interval $[10^{-4}, 10^{-2}]$; see [28], for instance.

How to initialize the classifier? To justify empirically the

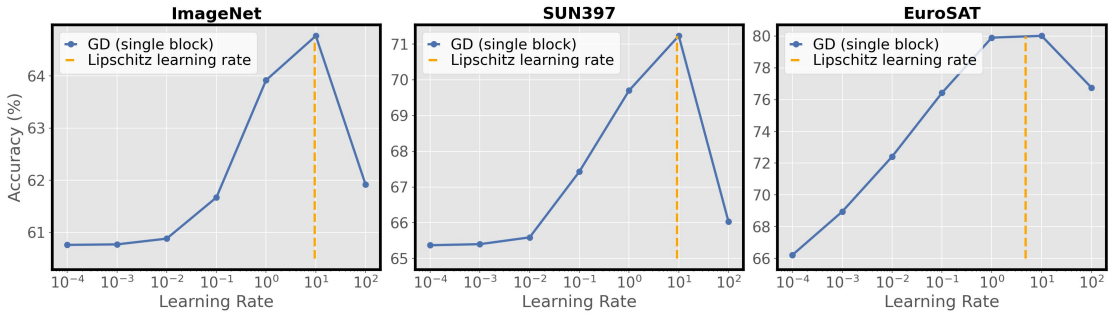

Figure 4. Single-block GD performance as a function of different values of the learning rates. The dotted vertical line shows the Lipschitz-based, data-driven learning rate.

Optimization Method	Standard loss ($\alpha_k = 0$)		Our loss		Time
	1 shot	16 shot	1 shot	16 shot	
LP++ (α_k is learnt)	35.55	69.75	**63.43**	74.42	0.78s
LP++ ($\alpha_k = 1$)	-	-	46.37	69.41	0.72s
GD (our Lipschitz cst)	35.55	69.75	63.04	74.28	0.86s
GD (optimum)	35.64	69.67	62.93	**74.55**	6.02s
ADAM (our Lipschitz cst)	26.22	64.53	25.44	64.62	0.98s
ADAM (optimum)	35.90	69.73	63.07	74.23	6.89s
L-BFGS [21]	34.54	67.44	62.09	72.82	6.73s

Table 2. Accuracy and run time for different optimizers and loss functions (average over 11 datasets). The running time is recorded for our loss (16 shots), and is averaged over 11 datasets. The best result is marked in bold, and the second best is underlined.

advantages brought by initializing the classifier weights following Eq. (13), and the blending parameters in Eq. (15), we evaluate objective (3) at the beginning of the training, as well as the training-free test accuracy of three methods: random initialization, and the training-free version of both the Tip-adapter-F method [28] (i.e., Tip-adapter) and our method (we provide more details on training-free LP++ in the appendix). The results in Table 3 confirm empirically the technical observations in Eqs. (13) and (15), which prescribe initial guesses for our problem's variables.

Number of shots (S)	**1**	**2**	**4**	**8**	**16**
Models	Initial Loss (L_0)				
Random Initialization	21.45	21.42	21.31	21.27	21.32
Proposed Initialization	1.60	1.54	1.47	1.35	1.21
Models	Initial test accuracy (Acc_0)				
Random Initialization	17.79	17.92	18.03	18.10	18.22
Tip-adapter[28]	59.28	59.72	60.55	62.09	64.29
Proposed Initialization	59.70	60.66	62.04	64.16	66.20

Table 3. Comparison of the initial loss and test accuracy in the *training-free* scenario: random initialization, Tip-adapter and the proposed initialization. The results are averaged over 11 datasets.

Computational overhead. As the literature on adapting VLMs is gaining popularity, it is essential to evaluate the extent to which novel methods are efficient. To do this, we report the overall computational overhead of the approaches studied in this work, which includes the time required for training and for finding the hyper-parameters, when applicable. We also indicate whether these methods enable black-box adaptation which, in our perspective, is a critical aspect in novel strategies aiming to address practical, real-world demands. The numerical values in Table 4 show that, in addition to yielding state-of-the-art performance (shown in previous sections), LP++ is the most efficient method (by several orders of magnitude), and does not require to access the internal representations of the pre-trained models.

Methods	Overall Time	BlackBox	# Parameters
CoOp[29]	$\sim$ 17h	✗	$K \times M \times D$
PLOT−2[4]	$\sim$ 10h	✗	$P \times K \times M \times D$
KgCoOp[27]	$\sim$ 4h	✗	$K \times M \times D$
ProGrad[30]	$\sim$ 20h	✗	$K \times M \times D$
Clip-Adapter[9]	$\sim$ 40min	✓	$2(D_1 \times D)$
Tip-adapter-F[28]	$\sim$ 6min	✓	$K \times S \times D$
Tip-adapter-F*[28]	$\sim$ 50min	✓	$K \times S \times D$
Standard LP[23]	3min	✓	$K \times D$
LP++	$\sim$ 2s	✓	$K(D+1)$

Table 4. Run time and suitability to black-box scenarios for different methods on 16-shot ImageNet. All the experiments are performed on a single NVIDIA RTX A6000 GPU, except for PLOT−2, which is evaluated on two A6000 GPUs. $D_1 = 256$, and $D = 1024$. The number of context tokens M is set to 16. For PLOT, $P = 4$ is the number of prompts.

4. Conclusion

We introduced LP++, a strong linear probe for few-shot CLIP adaptation. A specific modeling of the classifier weights, blending visual prototypes and text embeddings via learnable multipliers, along with convex-optimization ingredients, often overlooked in deep learning practices, led to the surprising results. While the findings of this work do not invalidate the promise of prompt learning and adaptation research, we believe LP++ could be used as a baseline to measure progress in these strongly emergent areas.

References

[1] Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization 23(4)*, 23(4):2037–2060, 2013. 4, 2

[2] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, pages 446–461, 2014. 6

[3] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–358, 2015. 4

[4] Guangyi Chen, Weiran Yao, Xiangchen Song, Xinyue Li, Yongming Rao, and Kun Zhang. Prompt learning with optimal transport for vision-language models. In *International Conference on Learning Representations*, 2023. 2, 6, 8, 7

[5] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. 6

[6] Pierre Colombo, Victor Pellegrain, Malik Boudiaf, Victor Storchan, Myriam Tami, Ismail Ben Ayed, Celine Hudelot, and Pablo Piantanida. Transductive learning for textual few-shot classification in api-based embedding models. In *Empirical Methods in Natural Language Processing*, 2023. 2

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 6

[8] Li Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *IEEE Workshop on Computer Vision and Pattern Recognition*, pages 178–178, 2004. 6

[9] Peng Gao, Shijie Geng, Renrui Zhang, Teli Ma, Rongyao Fang, Yongfeng Zhang, Hongsheng Li, and Yu Qiao. Clip-adapter: Better vision-language models with feature adapters. *International Journal of Computer Vision*, pages 1–15, 2023. 2, 6, 8, 7

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 6

[11] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019. 6

[12] Steffen Herbold. Autorank: A python package for automated ranking of classifiers. *Journal of Open Source Software*, 5 (48):2173, 2020. 4

[13] Mingyi Hong, XiangfengWang, Meisam Razaviyayn, and Zhi-Quan Luo. Iteration complexity analysis of block coordinate descent methods. *Mathematical Programming*, 163: 85–114, 2017. 4

[14] Z hong, D Friedman, and D Chen. Factual probing is [mask]: Learning vs. learning to recall. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2021. 2

[15] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning*, pages 4904–4916, 2021. 1

[16] Z Jiang, F Xu, J Araki, and G Neubig. How can we know what language models know. In *Association for Computational Linguistics*, 2020. 2

[17] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *International Conference on Computer Vision*, pages 554–561, 2013. 6

[18] Kenneth Lange, David R Hunter, and Ilsoon Yang. Optimization transfer using surrogate objective functions. *Journal of computational and graphical statistics*, 9(1):1–20, 2000. 3

[19] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. arXiv, 2013. 6

[20] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics and Image Processing*, pages 722–729, 2008. 6

[21] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980. 6, 8

[22] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505. IEEE, 2012. 6

[23] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763, 2021. 1, 3, 6, 8, 7

[24] T Shin, Logan R. L. IV Razeghi, Y, E Wallace, and S Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. 2020. 2

[25] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv, 2012. 6

[26] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3485–3492. IEEE, 2010. 6

[27] Hantao Yao, Rui Zhang, and Changsheng Xu. Visual-language prompt tuning with knowledge-guided context optimization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6757–6767, 2023. 2, 6, 8, 7

[28] Renrui Zhang, Wei Zhang, Rongyao Fang, Peng Gao, Kunchang Li, Jifeng Dai, Yu Qiao, and Hongsheng Li. Tip-adapter: Training-free adaption of clip for few-shot classification. In *European Conference on Computer Vision*, pages 493–510. Springer, 2022. 1, 2, 5, 6, 7, 8, 3, 4

[29] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *International Journal of Computer Vision*, 130(9):2337–2348, 2022. 2, 6, 8, 4, 7

[30] Beier Zhu, Yulei Niu, Yucheng Han, Yue Wu, and Hanwang Zhang. Prompt-aligned gradient for prompt tuning. In *International Conference on Computer Vision*, 2023. 2, 6, 8, 7