LoRA

Yuchen Yan
Australian National University

April 21, 2025

Overview

Problem statement - conditional language modeling objective

Fine tuning, the model is initialized to pre-trained weights $\Phi 0$ and updated to $\Phi_0 + \Delta\Phi$ by repeatedly following the gradient to maximize the conditional language modeling objective:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_\Phi\left(y_t \mid x, y_{<t}\right)\right)$$

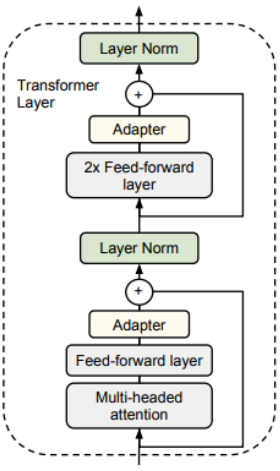
- $\mathcal{Z}$ the training dataset of input-output pairs (x, y)
- $y_{<t}$ all tokens before timestep t
- $P_\Phi\left(y_t \mid x, y_{<t}\right)$ the model's predicted probability of the token y_t, given the input x and previously generated tokens $y_{<t}$

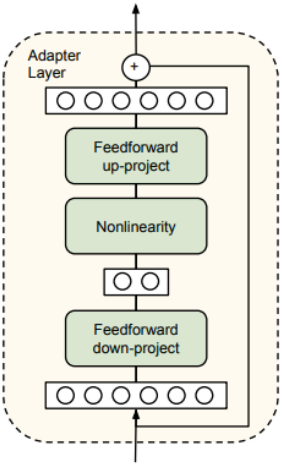
Parameter efficient approach

Adopt a more parameter-efficient approach, where the task-specific parameter increment $\Delta\Phi = \Delta\Phi(\Theta)$ is further encoded by a much smaller-sized set of parameters Θ with $|\Theta| \ll |\Phi_0|$. The task of finding $\Delta\Phi$ thus becomes optimizing over Θ :

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}\left(y_t \mid x, y_{<t}\right)\right)$$

Existing solutions-Adapter





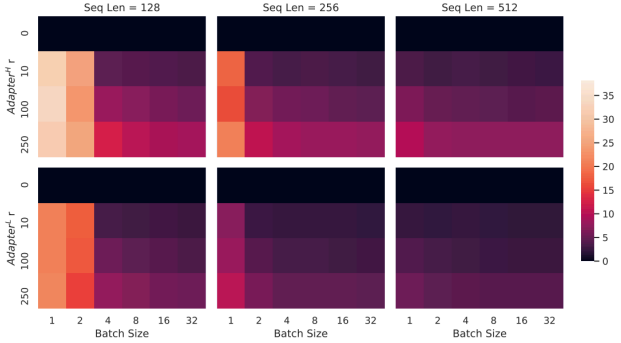
Two adapter layers per Transformer block with bottleneck architecture to limit the number of parameters

$$|\Theta| = 2md + d + m$$

md for projection, m for up-projection bias and d for down-projection bias

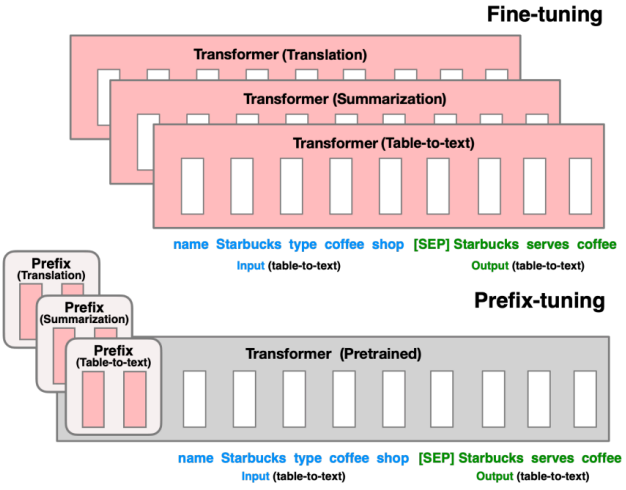
Existing solutions- Adapter Cont.

Batch Size Sequence Length $\|\Theta\|$	32 512 0.5M	16 256 11M	1 128 11M
Fine-Tune/LoRA	1449.4±0.8	338.0±0.6	19.8±2.7
$\text{Adapter}^{\text{L}}$	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
$\text{Adapter}^{\text{H}}$	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)



Adapters are added sequentially, without hardware parallelism (online inference scenario), a significant increase in latency

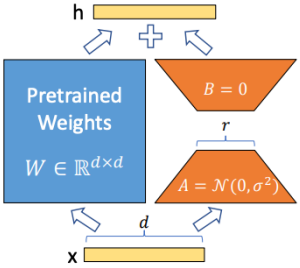
Existing solutions-Prefix tuning & Limitation



Prepends a sequence of *continuous task-specific* vectors to the input, only prefix is optimized.

Limitation difficult to optimize and adapted reduced sequence length for application

Low-rank-parametrized Update Matrices



Inspiration Aghajanyan et al. (2020) shows that the pre-trained language models have a low "instrisic dimension" and can still learn efficiently despite a random projection to a smaller subspace

Low-rank-parametrized Update Matrices

Define For a pre-trained weight matrix

$$W_0 \in \mathbb{R}^{d \times k}$$

constrains its update by representing the latter with a low-rank decomposition

$$W_0 + \Delta W = W_0 + BA,$$

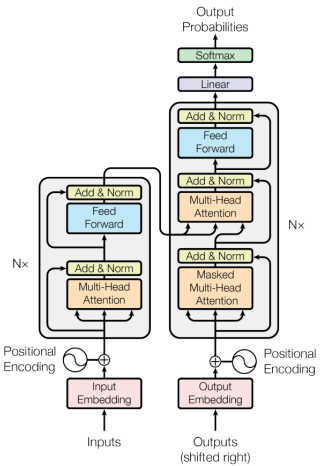
where $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$ and rank $r \ll \min(d, k)$.

Training W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. For $h = W_0 x$, modified forward pass yields:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

Initialization Gaussian for A & zero for B. Scale $\Delta W x$ by α / r where α is a constant in r.

Low-rank-parametrized Update Matrices



Application

LoRA is applicapable to any subset of weight matrices.

Transformer

four weight matrices in the self-attention module (W_q, W_k, W_v, W_o) and two in the MLP module.

Benefits

- reduction in memory and storage usage
- switch between tasks at a much lower cost by only swapping the LoRA weights

Limitation

Not straightforward to batch inputs to different tasks

LoRA — Baselines

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

General Language Understanding Evaluation (GLUE)

GLUE is a multitask benchmark and analysis platform for natural language understanding.

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

Table 1: Task descriptions and statistics. All tasks are single sentence or sentence pair classification, except STS-B, which is a regression task. MNLI has three classes; all other classification tasks have two. Test sets shown in bold use labels that have never been made public in any form.

Evaluation metrics — Matthews corr. (CoLA)

Matthews correlation coefficient (MCC) as a classification metric

$$\mathrm{MCC} = \frac{(\mathrm{TP} \cdot \mathrm{TN}) - (\mathrm{FP} \cdot \mathrm{FN})}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}$$

MCC $\in [-1, 1]$

1 $\rightarrow$ perfect prediction

-1 $\rightarrow$ complete disagreement

MCC value is symmetric, the order of the positive and negative classes doesn't matter.

The denominator computes the maximum possible value of the numerator.

Evaluation metrics — Pearson coefficient parameter (STS-B)

Pearson coefficient parameter as a metric for linear regression

$$r_{xy} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

where

- n is sample size
- x_i, y_i are the individual sample points indexed with i
- $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ (the sample mean); and analogously for $\bar{y}$.

LoRA — Baselines

Fine-Tuning (FT) the model is initialized to the pre-trained weights and biases, all model parameters undergo gradient updates. One baseline on GPT-2 adapts only last two layers.

Bias-only or BitFit only trains the bias vector.

Prefix-embedding tuning (PreEmbed) inserts special tokens among the input tokens. Use l_p (resp. l_i) denote the number of prefix (resp. infix) tokens. $|\Theta| = d_{\text{model}} \times (l_p + l_i)$.

Prefix-layer tuning (PreLayer) extension to prefix-embedding tuning. Learns the activations after every Transformer layer. $|\Theta| = L \times d_{\text{model}} \times (l_p + l_i)$, where L is the number of Transformer layers.

Evaluation metrics — Matthews corr. (CoLA)

Adapter tuning

- Adapter^H Two fully connected layers with biases with a nonlinearity in between.
- Adapter^L Adapter layer applied only after the MLP module and after a LayerNorm.
- Adapter^P similar to adapter^L
- Adapter^D drops some adapter layers for greater efficiency

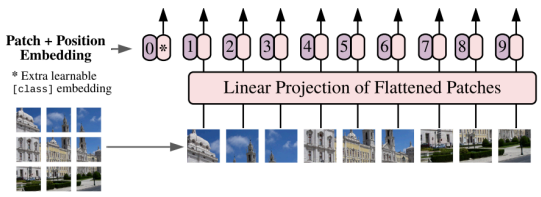
In all cases,

$$|\Theta| = \hat{L}_{\text{Adpt}} \times (2 \times d_{\text{model}} \times r + r + d_{\text{model}}) + 2 \times \hat{L}_{LN} \times d_{\text{model}}$$

where

- $\hat{L}_{\text{Adpt}}$ is the number of adapter layers,
- $\hat{L}_{LN}$ the number of trainable LayerNorms (e.g., in Adapter $^{\text{L}}$).

Image Encoder-Constructing Image Patches

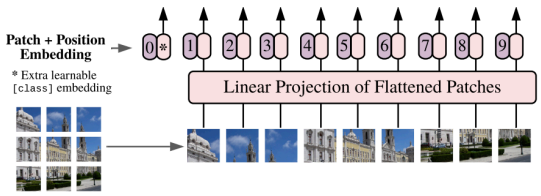


Reshape the image $x \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$

(H, W) : resolution of original image
C : number of channels
(P, P) : resolution of image patch
$N = HW/P^2$: number of patches

Example
224×224 image with 3 channels and 16×16 patches, $N = 196$

Image Encoder — Linear Projection



Flatten the patches and map to D dimensions using a trainable linear projection

$$\mathbf{z}_0 = \left[\mathbf{x}_{\text{class}}\,;\,\mathbf{x}_p^1\mathbf{E};\mathbf{x}_p^2\mathbf{E};\cdots;\mathbf{x}_p^N\mathbf{E}\right] + \mathbf{E}_{pos},$$

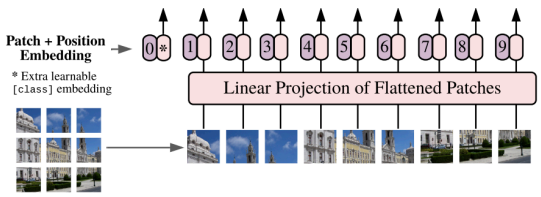
$$\mathbf{E} \in \mathbb{R}^{\left(P^2\cdot C\right)\times D},$$

$$\mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D}$$

Example

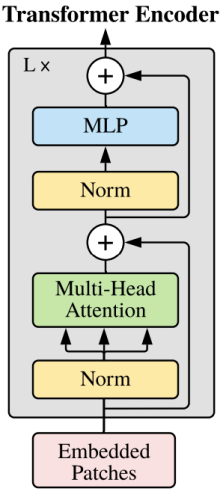
A 16×16 patch with 3 channels projected to 768 dimensions, $D = 768$

Image Encoder- [class] Token and Position Embeddings



- [class] token prepended to the sequence of embedded patches
  - interacts with all patches through self-attention, capturing a global summary of the image
  - acts as the final representation for classification
- Position embeddings $\mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D}$
  - added to the patch embedding to retain positional information
  - provide positional information

Self Attention - Layer Normalization



Normalize each embedding vector across the channel dimension

$$\mathrm{LN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}}{\hat{\boldsymbol{\sigma}}} + \boldsymbol{\beta}.$$

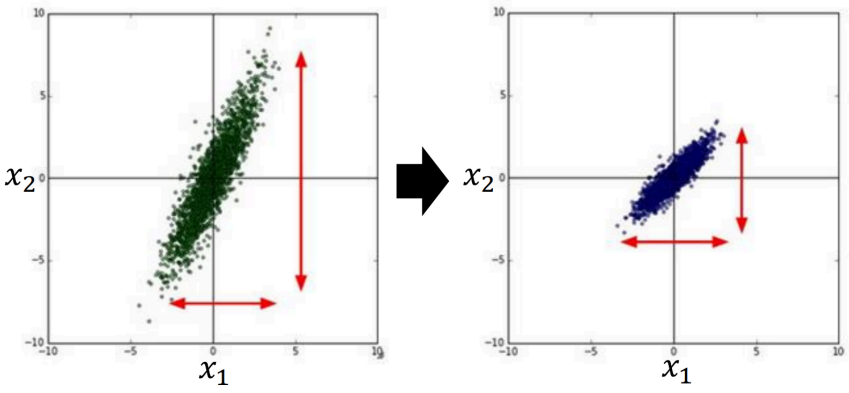
$$\hat{\boldsymbol{\mu}} = \frac{1}{d} \sum_{x^j \in \mathbf{x}} x^j$$

$$\hat{\boldsymbol{\sigma}}^2 = \frac{1}{d} \sum_{x^j \in \mathbf{x}} \left(x^j - \hat{\boldsymbol{\mu}}\right)^2 + \epsilon$$

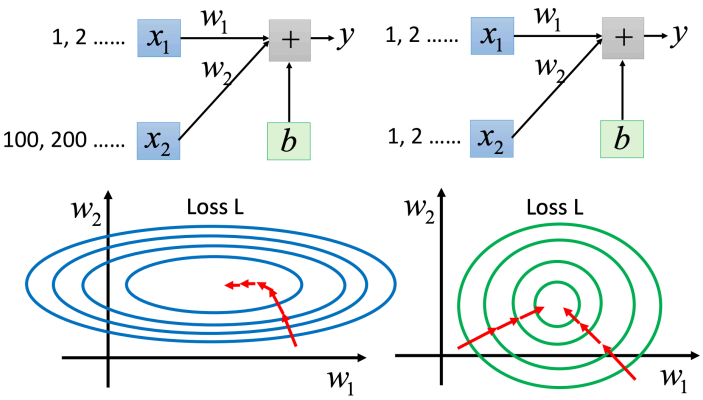
$\gamma, \beta \in \mathbb{R}^D$ learnable scale and shift parameters
ϵ small constant to avoid division by zero

Layer Normalization - Explanation

$$y = b + w_1 x_1 + w_2 x_2$$

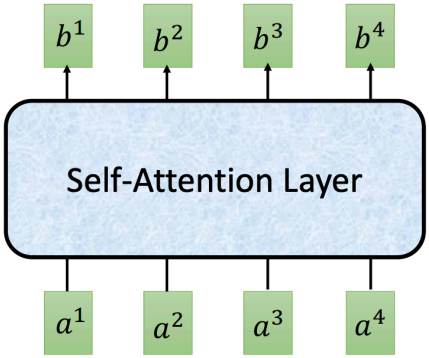


Layer Normalization - Explanation cont.



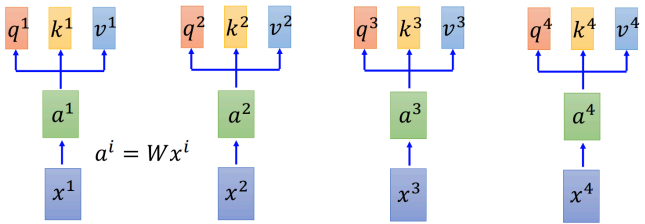
- Faster convergence
- Mitigating interval covariate shift
- Reducing the chance of exploding or vanishing gradients

Image Encoder - Self Attention



- b^i is obtained based on the whole input sequence
- b^1, b^2, b^3, b^4 can be parallelly computed

Image Encoder - Self Attention



- q : query (to math others)

$$q^i = W^q a^i$$

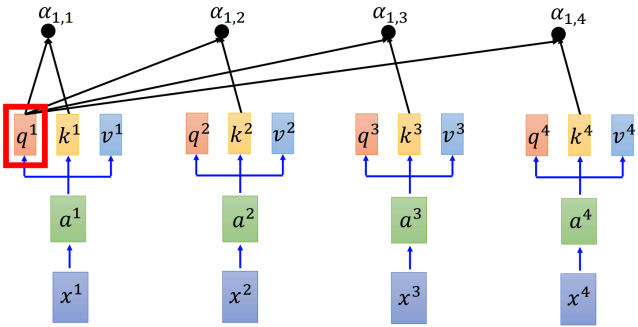
- k : key (to be matched)

$$k^i = W^k a^i$$

- v : information to be extracted

$$v^i = W^v a^i$$

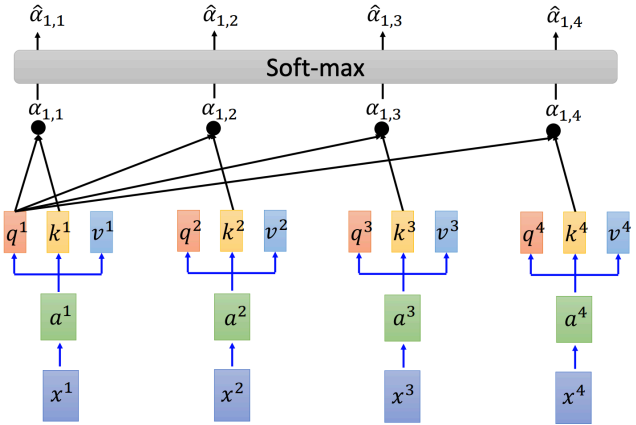
Image Encoder - Self Attention



Use each query to compute the dot product with all keys. The formula for computing the attention weights is called Scaled Dot-Product Attention:

$\alpha_{1,i} = q^i \cdot k^i / \sqrt{d}$, d dimension of q^i, k^i

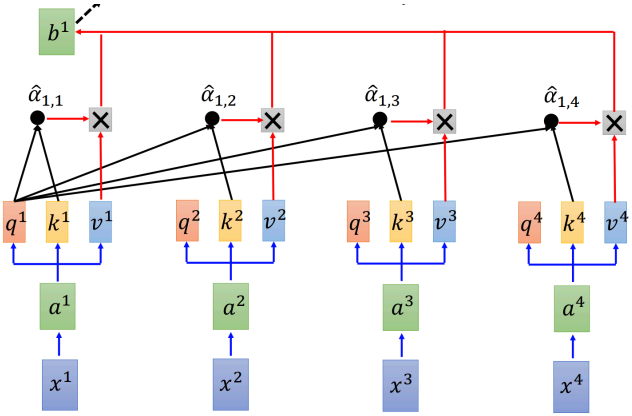
Image Encoder - Self Attention



Apply a soft-max function to obtain the normalized attention weights:

$$\hat{\alpha}_{1,i} = \exp(\alpha_{1,i}) / \sum_{j=1}^{N} \exp(\alpha_{1,j})$$

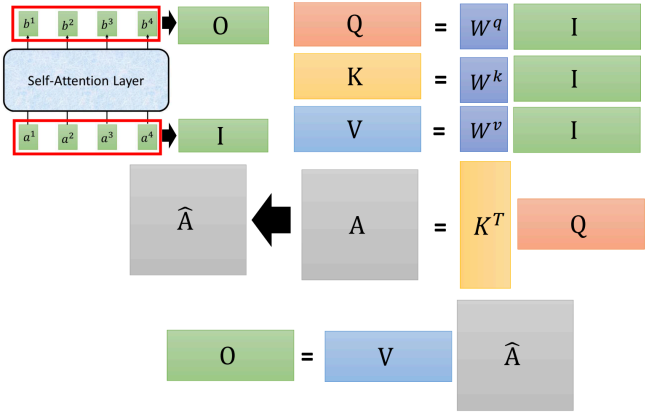
Image Encoder - Self Attention



The output b^i of the self-attention layer is the sum of the normalized attention weights and the value vectors:

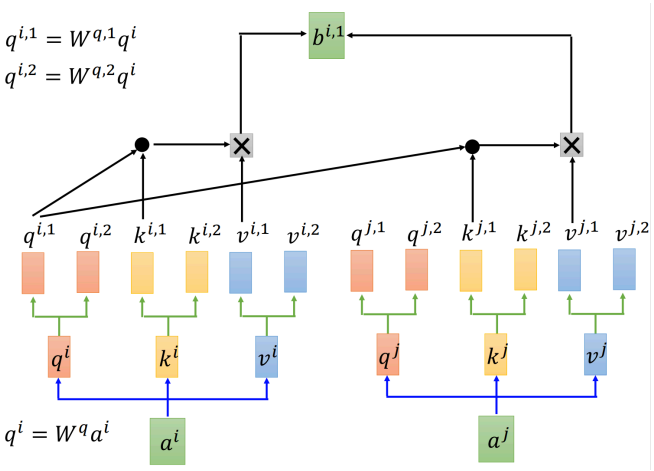
$$b^1 = \sum_{i=1}^{N} \hat{\alpha}_{1,i} v^i$$

Image Encoder - Self Attention conc.



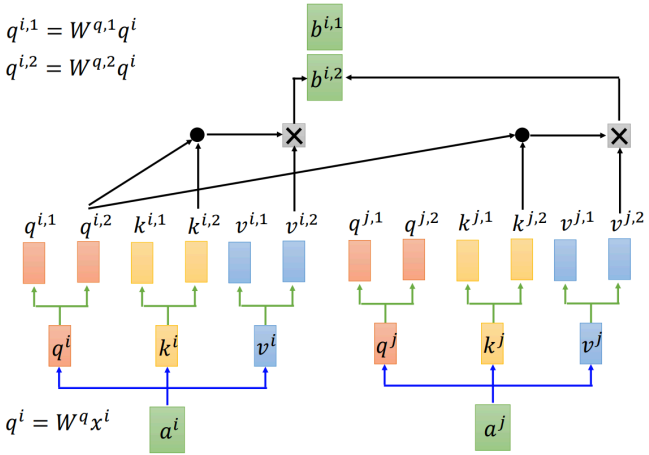
All these vector operations can concatenated into matrix multiplications. Therefore, GPU can be used to accelerate the computation.

Self Attention - Multihead Self Attention



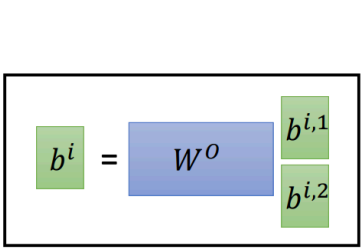
We have more than one W^q, W^k, and W^v to learn different linear projections. The output of each head is concatenated and projected to the final output dimension.

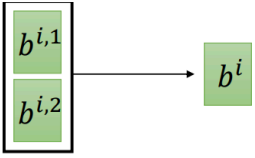
Self Attention - Multihead Self Attention



The calculation of $b^{i,1}$ only includes the multiplication of q^i and k^i in the first head.

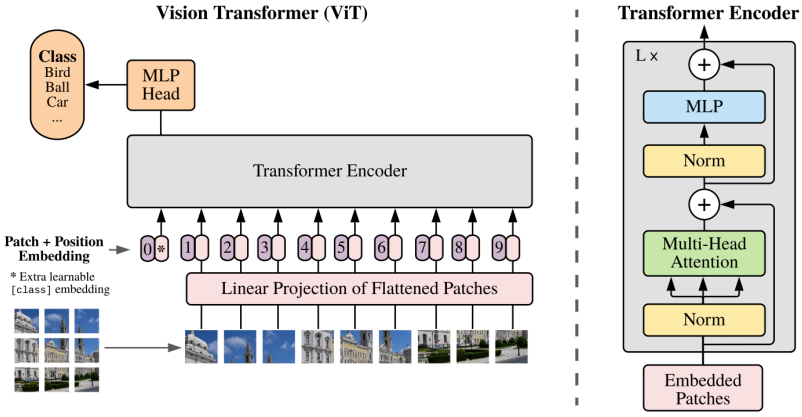
Self Attention - Multihead Self Attention





Two outputs of the multihead self-attention layer are concatenated by multiplying a W^0 matrix and projected to the final output dimension.

Image Encoder - Vision Transformer



Text Encoder - Modified Transformer

Quote

As a base size we use a 12-layer 512-wide model with 8 attention heads. The transformer operates on a ***lower-cased byte pair encoding (BPE) representation*** *of the text (Sennrich et al., 2015). The text sequence is bracketed with [SOS] and [EOS] tokens and the activations of the highest layer of the transformer at the [EOS] token are used as the feature representation of the text which is layer normalized and then linearly projected into the multi-modal embedding space. Masked self-attention was used in the text encoder to preserve the ability to add language modeling as an auxiliary objective, though exploration of this is left as future work.*

Subword-based Tokenization - BPE

Byte-Pair Encoding (BPE) is a simple form of data compression algorithm in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur in that data.

Motivation

Transformer has limited capacity to process long sequences, BPE is used to reduce the vocabulary size and the length of the input sequence

[*SOS*] **and** [*EOS*] special tokens added to the beginning and end of the text, indicating the start and end of the text.

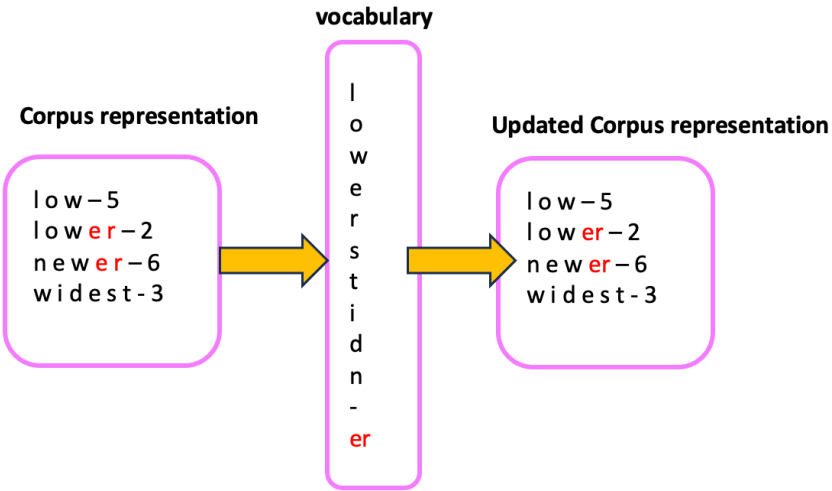
Subword-based Tokenization - BPE

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

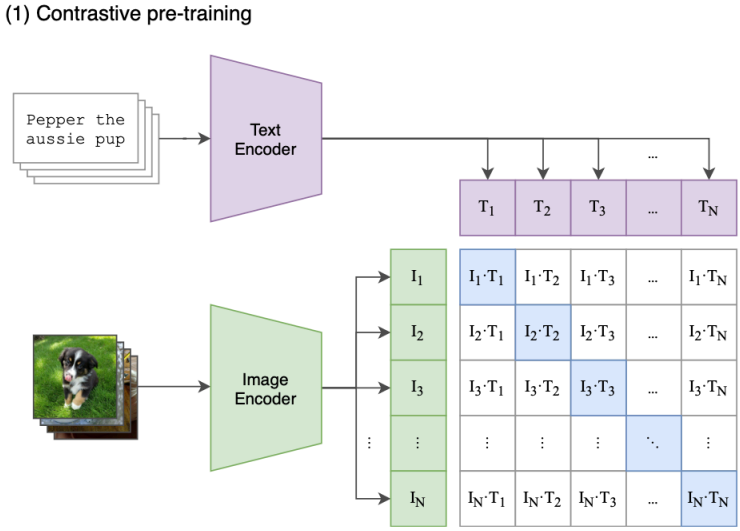
$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Figure 2.13 The token learner part of the BPE algorithm for taking a corpus broken up into individual characters or bytes, and learning a vocabulary by iteratively merging tokens. Figure adapted from Bostrom and Durrett (2020).

BPE - example


vocabulary
Corpus representation
l o w – 5
l o w e r – 2
n e w e r – 6
w i d e s t - 3
l
o
w
e
r
s
t
i
d
n
-
er
Updated Corpus representation
l o w – 5
l o w er – 2
n e w er – 6
w i d e s t - 3

Encoder


(1) Contrastive pre-training
Pepper the aussie pup
Text Encoder
Image Encoder
T1
T2
T3
...
TN
I1
I2
I3
IN
I1·T1
I1·T2
I1·T3
I1·TN
I2·T1
I2·T2
I2·T3
I2·TN
I3·T1
I3·T2
I3·T3
I3·TN
IN·T1
IN·T2
IN·T3
IN·TN

Training

Both models are optimized during pretraining to align similar text ad images in vector space. It does this by taking image-text pairs and pushing their output vectors nearer in vector space while pushing the output vectors of dissimilar pairs further apart.

Contrastive loss

takes pairs of example as input and trains a network to predict whether two inputs are from the same class or not.

$$\mathcal{L}_{\text{cont}}^{m}\ (x_i, x_j; f) = \mathbf{1}\{y_i = y_j\} \left\|f_i - f_j\right\|_2^2 + \mathbf{1}\{y_i \neq y_j\} \max\left(0, m - \left\|f_i - f_j\right\|_2\right)^2$$

$1\{y_i = y_j\}$ indicator function , $\left\|f_i - f_j\right\|_2 = \sqrt{\sum_k (f_i^{(k)} - f_j^{(k)})^2}$

$\max\left(0, m - \left\|f_i - f_j\right\|_2\right)^2$

- if $m - \left\|f_i - f_j\right\|_2^2 \leq 0 = 0$ no loss is added
- elif $m - \left\|f_i - f_j\right\|_2^2 > 0 = 0$ the loss is positive, penalty added, separating two embeddings in Euclidean space

Triplet loss

composed of triplets, consisting of a query, a positive example, and a negative example

$$\mathcal{L}_{\text{tri}}^{m}\ (x, x^{+}, x^{-}; f) = \max\left(0, \left\|f - f^{+}\right\|_2^2 - \left\|f - f^{-}\right\|_2^2 + m\right)$$

$\max\left(0, \|f - f^{+}\|_2^2 - \|f - f^{-}\|_2^2 + m\right)$

- if $\|f - f^{-}\|_2^2 - \|f - f^{+}\|_2^2 \geq m$ no loss is added
- elif $\|f - f^{-}\|_2^2 - \|f - f^{+}\|_2^2 < m$ the loss is positive, model penalised

Pretraining Method Selection-Distance Metric Learning

Limitation both loss functions are known to suffer from slow convergence and they often require expensive data sampling method to provide nontrivial pairs or triplets to accelerate the training
Trivial pairs $\rightarrow$ weak gradient updates $\rightarrow$ slow convergence

Nontrivial pairs

- Hard positive pairs: Pairs from the same class that are far apart in feature space force the model to learn better embeddings to bring them closer.
- Hard negative pairs: Pairs from different classes that are close together push the model to better separate different classes.

Pre-Training Method Selection-Distance Metric Learning with Multiple Negative Examples

Goal

- positives examples $\rightarrow$ shorten distances between embedding vectors
- negative examples $\rightarrow$ enlarging distances

Problem
During the update, the triplet loss only compares an example with one negative example while ignoring negative examples from the rest of the classes. The consequence is the embedding vector of an example is only guaranteed to be far from the selected negative class but not the others.

Solution A loss function that recruits multiple negatives for each update

Pretraining Method Selection - Distance Metric Learning with Multiple Negative Examples

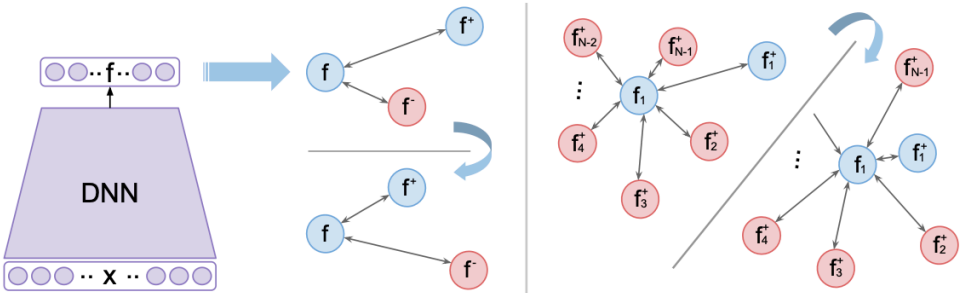


Figure: Triplet Loss vs. Multiple Negative Examples Loss

Pretraining Method Selection-Distance Metric Learning with Multiple Negative Examples

(N+1)-tuplet loss

$$\mathcal{L}\left(\left\{x, x^{+}, \{x_i\}_{i=1}^{N-1}\right\}; f\right) = \log\left(1 + \sum_{i=1}^{N-1} \exp\left(f^{\top} f_i - f^{\top} f^{+}\right)\right)$$

Cosine similarity $f^T f_i = ||f|| \, ||f_i|| \cos\theta$

- if dot product is large, $\cos\theta \to 1$, $\theta \to 0$, high similarity
- small, $\cos\theta \to 0$, $\theta \to \perp$, no similarity
- negative, $\cos\theta \to -1$, $\theta \to 180°$, strong dissimilarity

Pretraining Method Selection- N-pair Loss Batch Construction

Problem When the batch size of Stochastic Gradient Descent is M, there are $M \times (N+1)$ examples to be passed through f at one update. Too large and impractical to scale.

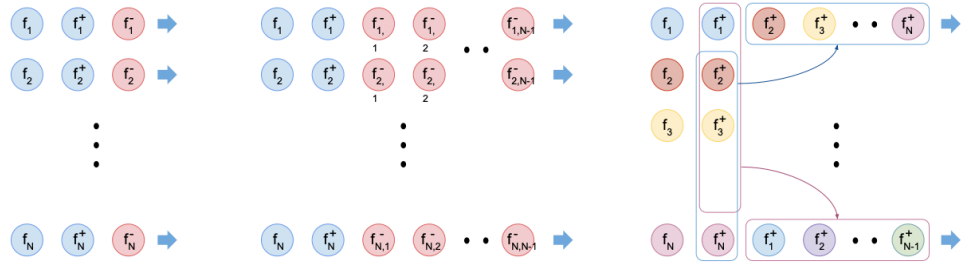
Effective Batch Construction

N pairs of examples from N different classes $\left\{\left(x_1, x_1^+\right), \cdots, \left(x_N, x_N^+\right)\right\}$

Build N tuples, denoted as $\{S_i\}_{i=1}^N$, from the N pairs.

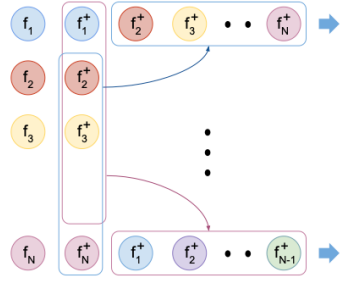
- $S_i = \left\{x_i, x_1^+, x_2^+, \cdots, x_N^+\right\}$
  - x_i is the query for S_i
  - x_i^+ a positive example, rest are negative examples

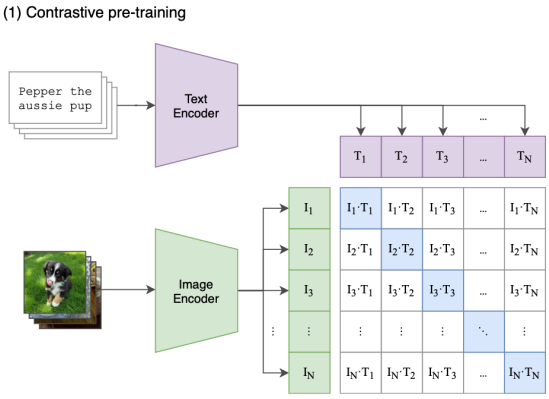
Pretraining Method Selection-Distance Metric Learning with Multiple Negative Examples



(a) Triplet loss (b) $(N+1)$-tuplet loss (c) N-pair-mc loss

Pretraining Method Selection-CLIP & Multi-class N-pair Loss





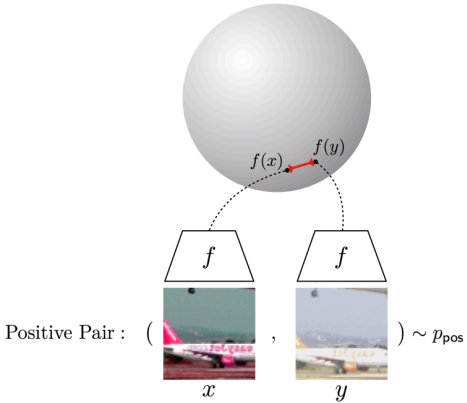
Pretraining Method Selection - Symmetric Entropy Loss(N-pair Loss/InfoNCE)

Samples a batch of N input pairs (x_v, x_u) for training data, and calculate their representation pairs (v, u), i-th pair denoted as (v_i, u_i). The training objective of CLIP involves two loss functions. The first loss function is an image-text contrastive loss for the i-th pair:

$$\ell_i^{(v \to u)} = -\log \frac{\exp\left(\langle \mathbf{v}_i, \mathbf{u}_i \rangle / \tau\right)}{\sum_{k=1}^{N} \exp\left(\langle \mathbf{v}_i, \mathbf{u}_k \rangle / \tau\right)}$$

where $\langle v_i, u_i \rangle$ represents the cosine similarity
$\tau \in \mathbb{R}^+$ represents a temperature parameter, affects the smoothness of the probability distribution over the negative examples.

Contrastive learning - temperature τ



Alignment: Similar samples have similar features.
(Figure inspired by Tian et al. (2019).)

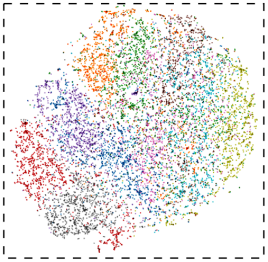
Alignment favors encoders that assign similar features to similar samples.

Contrastive learning - temperature τ

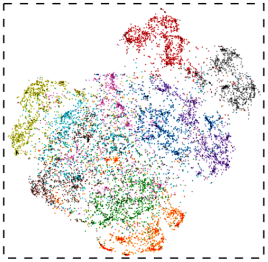
Uniformity: Preserve maximal information.

Uniformity prefers a feature distribution that preserves maximal information, i.e., the uniform distribution on the unit hypersphere.

Contrastive learning - temperature τ



$\tau = 0.07$



$\tau = 0.2$

Small temperature tends to generate more uniform distribution and be less tolerant to similar samples.

Pre-Training Method Selection - Symmetric Entropy Loss

Image-to-text contrastive loss is asymmetric for each input modality. Therefore, defined a similar text-to-image contrastive loss as:

$$\ell_i^{(u \to v)} = -\log \frac{\exp\left(\langle \mathbf{u}_i, \mathbf{v}_i \rangle / \tau\right)}{\sum_{k=1}^{N} \exp\left(\langle \mathbf{u}_i, \mathbf{v}_k \rangle / \tau\right)}$$

Final training loss is then computed as a weighted combination of the two losses averaged over all positive image-text pairs in each batch:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(\lambda \ell_i^{(v \to u)} + (1 - \lambda) \ell_i^{(u \to v)} \right)$$

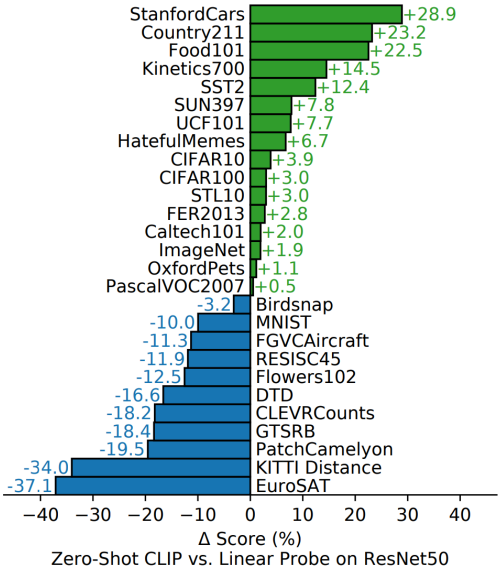
Analysis - Initial Comparison to Visual N-Grams

	aYahoo	ImageNet	SUN
Visual N-Grams	72.4	11.5	23.0
CLIP	**98.4**	**76.2**	**58.5**

Table 1. Comparing CLIP to prior zero-shot transfer image classification work. CLIP improves performance on all three datasets by a large amount. This improvement reflects many differences since the development of Visual N-Grams (Li et al., 2017).

- matches the performance of the original ResNet50 despite using one of the 1.28 million crowd-labeled training examples
- high 95% top-5 accuracy
  - any of the model 5 highest probability answers must match the expected answer
- Differences controlled comparison between CLIP ResNtt50 and N-Grams on the same YFCC100M dataset, matched performance

Analysis - Zero-Shot Performance



Zero-Shot CLIP vs. Linear Probe on ResNet50

Comparison

- **baseline** a fully supervised, regularized, logistic regression classifier on the features of the canonical ResNet50 across 27 datasets

Outperforms

- limited number of labeled examples
- general object classification datasets
- datasets measuring action recognition in videos

Underperforms

- visual concepts involving verbs
- specialized, complex, or abstract tasks

Analysis - Zero-Shot Performance

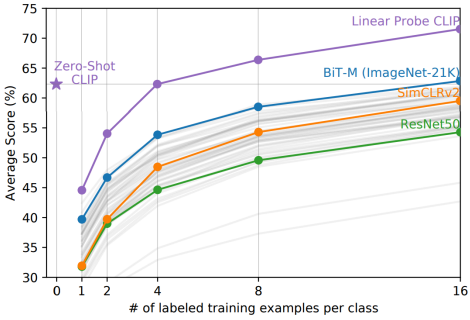


Figure 5. **Zero-shot CLIP outperforms few-shot linear probes.** Zero-shot CLIP matches the average performance of a 4-shot linear classifier trained on the same feature space and nearly matches the best results of a 16-shot linear classifier across publicly available models. For both BiT-M and SimCLRv2, the best performing model is highlighted. Light gray lines are other models in the eval suite. The 20 datasets with at least 16 examples per class were used in this analysis.

Comparison with few-shot methods

- few-shot logistic regression on the features of many image models

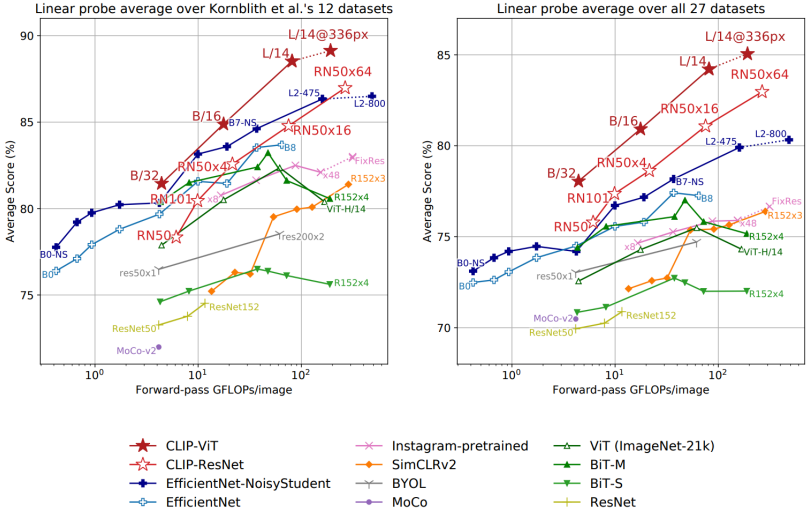
matches

- 4-shot logistic regression on the same feature space
  - CLIP's zero-shot classifier:is generated via natural languags which allows for visual concepts to be directly specified
  - normal supervised learning: must infer concepts indirectly from training examples

roughly matches

- 16-shot classifier, which uses the features of a BiT-M ResNet152x2 trained on ImageNet-21K.

Analysis - Representation Learning

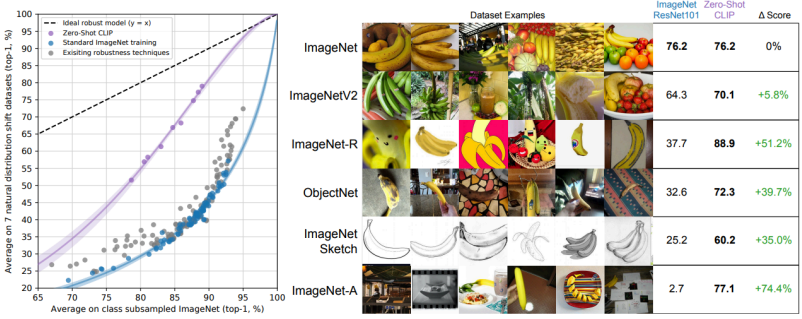


Analysis - Representation Learning

- the largest CLIP model slightly outperforms the best existing model on both overall score and compute efficiency
- CLIP transformers 3x more compute efficient than CLIP ResNets **Broader**

  evaluation suite tasks include geo-localization, optical character recognition, facial emotion recognition, and character recognition
  - All CLIP models, regardless of scale, outperform all evaluated systems in terms of compute efficiency

Analysis - Distribution shift



Zero-shot CLIP is much more robust to distribution shift than the ResNet101.

The End