It's a (Blind) Match!
Towards Vision-Language Correspondence without Parallel Data

Supplementary Material

A. Overview

- Appx. B shows that our QAP formulation is general enough to accommodate the commonly used distance and similarity measures: Mutual k-NN, CKA and the Gromov-Wasserstein distance.
- Appx. C follows up on the experiment in Sec. 3. It shows that the vision-language similarity measured by our distortion metrics decreases as the amount of shuffling increases. Furthermore, it elaborates on why randomly initialized networks exhibit a similar trend to pre-trained networks.
- Appx. D compares the original Hahn-Grant solver to our factorized Hahn-Grant solver including a correctness proof, implementation details and an ablation showing the improvements in primal and dual estimates by our solver.
- Appx. E gives more details on the used models, datasets and specific setups for the individual experiments.
- Appx. F provides results for other model and dataset combinations. It shows that pre-training appears more important than the model size for the matching accuracy in small-scale alignment. Also, it shows that the results for solver comparison consistently hold for all considered model and dataset combinations including unsupervised classification.
- Appx. G shows that the Gromov-Wasserstein optimal transport solvers can be seen as solvers for a relaxation of our QAP.

B. Mutual k-NN and CKA as distortion metrics

In Sec. 4, we introduce the notation of distortion metrics in terms of a unimodal kernel function k_x and k_y, and a distance function l, which is decomposable as

$$l(A, B) = f_1(A) + f_2(B) - h_1(A)h_2(B). \tag{15}$$

In this section, we show how Mutual k-NN [17], the centered kernel alignment (CKA) [22], and the Gromov-Wasserstein ($\mathcal{GW}$) distance [33] fit into this formulation.

Mutual k-NN [17]. Mutual k-NN is defined as the average overlap between the nearest neighbors in both modalities, *i.e.*, for one sample, it is

$$m_{\text{kNN}}(\mathbf{x}_i, \mathbf{y}_i) = \frac{1}{k} |\operatorname{top}_k^{\mathbf{x}}(i) \cap \operatorname{top}_k^{\mathbf{x}}(i)|. \tag{16}$$

Here, we define the top-k indices in terms of the highest inner product as $\operatorname{top}_k^{\mathbf{x}}(i)$. Following Huh et al. [17], we exclude i from this set. To include this into our framework, we define the kernel function as

$$k_x^{\text{kNN}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{\sqrt{Nk}} \mathbb{1}[j \in \operatorname{top}_k^{\mathbf{x}}(i)], \tag{17}$$

and accordingly for the language embeddings. Then, the similarity matrices are

$$\mathbf{X}_{ij}^{\text{kNN}} = k_x^{\text{kNN}}(\mathbf{x}_i, \mathbf{x}_j) \text{ and } \mathbf{Y}_{ij}^{\text{kNN}} = k_y^{\text{kNN}}(\mathbf{y}_i, \mathbf{y}_j). \tag{18}$$

Furthermore, we use the negative inner product as our distortion function:

$$l_{\text{inner}}(A, B) = -A \cdot B, \tag{19}$$

which trivially satisfies Eq. (15) with $f_1 = f_2 = 0$ and h_1, h_2 being identity functions. This choice of kernels and the distance metric leads to the Mutual k-NN metric in our framework:

$$\begin{aligned}
&\mathcal{D}_{\text{kNN}}(\mathbf{X}^{\text{kNN}}, \mathbf{Y}^{\text{kNN}}) \\
&= \sum_{i,j=1}^{N} l\left(\mathbf{X}_{ij}^{\text{kNN}}, \mathbf{Y}_{ij}^{\text{kNN}}\right) \\
&= -\sum_{i,j=1}^{N} k_x^{\text{kNN}}(\mathbf{x}_i, \mathbf{x}_j) k_y^{\text{kNN}}(\mathbf{y}_i, \mathbf{y}_j) \\
&= -\sum_{i,j=1}^{N} \frac{1}{\sqrt{Nk}} \mathbb{1}[j \in \operatorname{top}_k^{\mathbf{x}}(i)] \frac{1}{\sqrt{Nk}} \mathbb{1}[j \in \operatorname{top}_k^{\mathbf{y}}(i)] \\
&= -\frac{1}{N} \sum_{i,j=1}^{N} \frac{1}{k} \mathbb{1}[j \in \operatorname{top}_k^{\mathbf{x}}(i) \wedge j \in \operatorname{top}_k^{\mathbf{y}}(i)] \\
&= -\frac{1}{N} \sum_{i,j=1}^{N} \frac{1}{k} \mathbb{1}[j \in \operatorname{top}_k^{\mathbf{x}}(i) \cap \operatorname{top}_k^{\mathbf{y}}(i)] \\
&= -\frac{1}{N} \sum_{i=1}^{N} \frac{1}{k} |j \in \operatorname{top}_k^{\mathbf{x}}(i) \cap \operatorname{top}_k^{\mathbf{y}}(i)| \\
&= -\frac{1}{N} \sum_{i=1}^{N} m_{\text{kNN}}(\mathbf{x}_i, \mathbf{y}_i).
\end{aligned} \tag{20}$$

CKA. We derive CKA [22, 30] in a similar fashion. For a kernel function $\hat{k}$ and kernel matrices $\hat{\mathbf{X}}_{ij} = \hat{k}(\mathbf{x}_i, \mathbf{x}_j)$ and $\hat{\mathbf{Y}}_{ij} = \hat{k}(\mathbf{y}_i, \mathbf{y}_j)$, the CKA is defined as

$$\text{CKA}(\hat{\mathbf{X}}, \hat{\mathbf{Y}}) = \frac{\operatorname{tr}(\hat{\mathbf{X}}\mathbf{C}\hat{\mathbf{Y}}\mathbf{C})}{\sqrt{\operatorname{tr}(\hat{\mathbf{X}}\mathbf{C}\hat{\mathbf{X}}\mathbf{C}) \operatorname{tr}(\hat{\mathbf{Y}}\mathbf{C}\hat{\mathbf{Y}}\mathbf{C})}}, \tag{21}$$

Models	CIFAR-10 (%)	CINIC-10 (%)
Mutual k-NN	30.5 ± 16.7	16.5 ± 27.6
CKA	42.0 ± 4.1	40.0 ± 0.0
$\mathcal{GW}$ distance	$\mathbf{72.0 \pm 17.7}$	$\mathbf{77.0 \pm 7.3}$

Table 3. **Gromov-Wasserstein distance is the best measure for matching:** We show the accuracy for CIFAR-10 [23] and CINIC-10 [9] using DINOv2 [36] and all-mpnet-base-v2 [43] using Mutual k-nearest neighbor (Mutual k-NN) [17], centered kernel alignment (CKA) [22], and the Gromov-Wasserstein ($\mathcal{GW}$) distance [33] as a metric. The $\mathcal{GW}$ distance leads to the best matching accuracy for both datasets.

where $\mathbf{C} = \mathbf{I} - \frac{1}{N}\mathbb{1}\mathbb{1}^T$. Similar to previous work [30], we use the linear kernel in this work. Our kernel matrices are

$$\mathbf{X}^{\text{CKA}} = \frac{\hat{\mathbf{X}}\mathbf{C}}{\sqrt{\operatorname{tr}(\hat{\mathbf{X}}\mathbf{C}\hat{\mathbf{X}}\mathbf{C})}} \quad \text{and} \quad \mathbf{Y}^{\text{CKA}} = \frac{\mathbf{C}^T\hat{\mathbf{Y}}^T}{\sqrt{\operatorname{tr}(\hat{\mathbf{Y}}\mathbf{C}\hat{\mathbf{Y}}\mathbf{C})}}. \tag{22}$$

Using the negative inner product as the distance metric leads to the negative CKA:

$$\begin{aligned}
&\mathcal{D}_{\text{CKA}}(\mathbf{X}^{\text{CKA}}, \mathbf{Y}^{\text{CKA}}) \\
&= \sum_{i,j=1}^{N} l\left(\mathbf{X}^{\text{CKA}}_{ij}, \mathbf{Y}^{\text{CKA}}_{ij}\right) \\
&= -\sum_{i,j=1}^{N} \frac{(\hat{\mathbf{X}}\mathbf{C})_{ij}}{\sqrt{\operatorname{tr}(\hat{\mathbf{X}}\mathbf{C}\hat{\mathbf{X}}\mathbf{C})}} \frac{(\mathbf{C}^T\hat{\mathbf{Y}}^T)_{ij}}{\sqrt{\operatorname{tr}(\hat{\mathbf{Y}}\mathbf{C}\hat{\mathbf{Y}}}} \\
&= -\frac{1}{\sqrt{\operatorname{tr}(\hat{\mathbf{X}}\mathbf{C}\hat{\mathbf{X}}\mathbf{C})\operatorname{tr}(\hat{\mathbf{Y}}\mathbf{C}\hat{\mathbf{Y}}\mathbf{C})}} \sum_{i,j=1}^{N} (\hat{\mathbf{X}}\mathbf{C})_{ij}(\hat{\mathbf{Y}}\mathbf{C})_{ji} \\
&= -\frac{1}{\sqrt{\operatorname{tr}(\hat{\mathbf{X}}\mathbf{C}\hat{\mathbf{X}}\mathbf{C})\operatorname{tr}(\hat{\mathbf{Y}}\mathbf{C}\hat{\mathbf{Y}}\mathbf{C})}} \operatorname{tr}(\hat{\mathbf{X}}\mathbf{C}\hat{\mathbf{Y}}\mathbf{C}) \\
&= -\text{CKA}(\hat{\mathbf{X}}, \hat{\mathbf{Y}})
\end{aligned} \tag{23}$$

$\mathcal{GW}$ distance [33]. For the $\mathcal{GW}$ distance, we can choose the L_2-norm as the kernel and the squared distance as the distortion metric:

$$\begin{aligned}
k_x^{\mathcal{GW}}(\mathbf{x}_i, \mathbf{x}_j) &= \|\mathbf{x}_i - \mathbf{x}_j\|_2, \\
k_y^{\mathcal{GW}}(\mathbf{y}_i, \mathbf{y}_j) &= \|\mathbf{y}_i - \mathbf{y}_j\|_2, \\
l_{\mathcal{GW}}(A, B) &= (A - B)^2.
\end{aligned} \tag{24}$$

Then, the $\mathcal{GW}$ distance is given by

$$\mathcal{D}_{\mathcal{GW}}(\mathbf{X}^{\mathcal{GW}}, \mathbf{Y}^{\mathcal{GW}}) = \sum_{i,j=1}^{N} \left(\|\mathbf{x}_i - \mathbf{x}_j\|_2 - \|\mathbf{y}_i - \mathbf{y}_j\|_2\right)^2. \tag{25}$$

The objective function after finding the optimal permutation matrix similar to Eq. (6) is then equivalent to the original $\mathcal{GW}$ distance comparing two metric-measure spaces [38].

Tab. 3 empirically compares these formulations in a small-scale data regime, using CIFAR-10 [23] and CINIC-10 [9] datasets introduced in the main text. We find that despite the wider adoption of Mutual k-NN and CKA metrics in previous work, the Gromov-Wasserstein distance leads to a higher matching accuracy on both datasets.

C. Shuffled vision-language alignment

In this section, we elaborate on the setup for the shuffling experiment from Sec. 3. As we claimed in the main text (*cf*. Sec. 3), the observations are consistent accross all tested datasets and models, which we also report here.

Setup. Given aligned image and language representations, $(\mathbf{x}_i, \mathbf{y}_i)$, and a shuffling level $\alpha \in [0, 1]$, we randomly choose $\lfloor \alpha N \rfloor$ elements that are randomly permuted. Every other element is kept in place. Afterwards, the distortion of this permutation is computed with Eq. (3) or Eq. (4) after the permutation. Note that we use the image embeddings here instead of the averaged object embeddings. For classification datasets, we take the same language embedding for all elements of that class. However, we have seen in preliminary experiments that the curve looks similar when considering the averaged object representations instead of the image representations. We plot 21 equidistant shuffling levels at $\alpha \in \{0, 0.05, \ldots, 1\}$. Each level is based on 100 random seeds to sample the subset and permutation.

Shuffling with other kernels, datasets, and models. In addition to Fig. 2, which uses the CocoCaptions dataset [6], and Mutual k-NN as the distortion metric, we show more combinations in Fig. 7. In this setting, Mutual k-NN is only meaningful for paired datasets because the k-nearest neighbors are ambiguous when language features are replicated. For the $\mathcal{GW}$ distance, the pairwise distances are also dependent on the dimensionality of the embedding spaces. Therefore, we standardize the distance to be in the range between zero and one for this shuffling experiment. We observe that the similarity (/ distortion) decreases (/ increases) strongly monotonically with more shuffling. This behavior is consistent for all considered datasets and metrics. This observation suggests that the pairwise relations are more similar between the semantically corresponding vision and language representations than between the non-semantic ones.

A note on randomly initialized networks. In Fig. 8, we present a plot of the experiment in Fig. 2 with the addition of randomly initialized ViT-H/14 [11] models, which were initiated with 20 distinct random seeds. Additionally, we demonstrate the distortion of a representation based on the stacked pixel values, *i.e.*, without any neural network. Furthermore, we show a zoomed-in version on the left-hand side to illustrate the behavior on a finer scale.

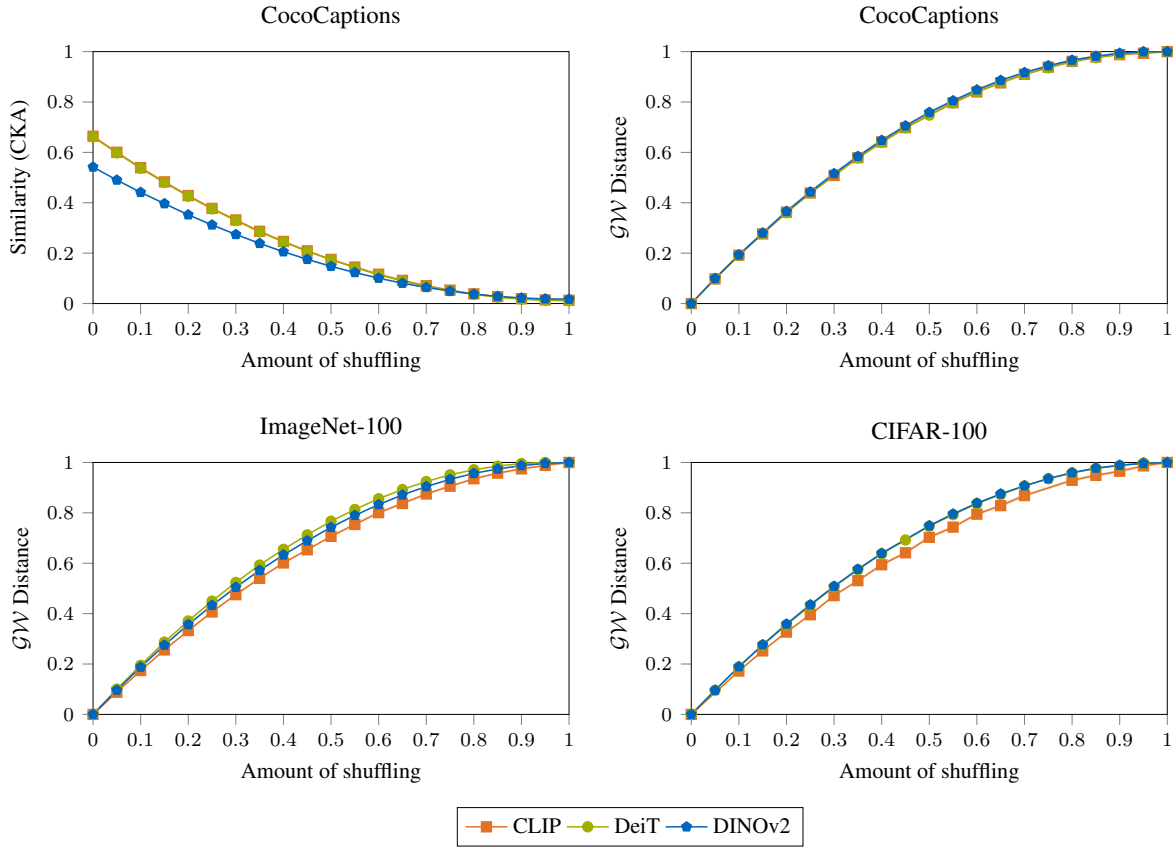

Figure 7. **Shuffling degrades vision-language alignment:** We report CKA similarity and the Gromov-Wasserstein distance on three datasets and based on five encoding methods, including pre-trained and randomly initialized networks, and raw pixel values. Similar to our observation with Mutual k-NN, the CKA and Gromov-Wasserstein distance correlate strongly w.r.t. the amount of shuffling, which suggests that these metrics are also suitable measures for blind vision-language matching.

We observe that the alignment of randomly initialized networks exhibits the same monotonically decreasing behavior with increased amount of shuffling, as for pre-trained networks (despite the absolute alignment value being smaller). For images, this can be explained by the similarity of the pixel distribution within each semantic category (*e.g.*, a green landscape for animal stock). These similarities appear more frequently for semantically affiliated classes and can dominate the pairwise distance encoding. To understand the behavior for randomly initialized networks, we plot the distribution of the *Empirical Lipschitz constant* in Fig. 9, defined by

$$\mathcal{K}_{\text{Lipschitz}}(\mathbf{x}, \mathbf{y}) = \frac{\|f(\mathbf{x}) - f(\mathbf{y})\|_2}{\|\mathbf{x} - \mathbf{y}\|_2}, \tag{26}$$

for samples $\mathbf{x}, \mathbf{y}$ and function f. Intuitively, it measures the degree of distance distortion in the output space w.r.t. the input space for each sample pair in the dataset. Here, we use CocoCaptions and the same language model as in Fig. 8. We observe that most values are close to one. This implies that the distance after encoding remains approximately preserved. Nevertheless, the distances are slightly distorted, which explains why the absolute similarity in Fig. 8 is lower for random ViTs than for the pixel values. In summary, shuffling reduces alignment even for the distance in terms of pixel values. This then transfers to random ViTs because these distances are approximately preserved.

D. On the *factorized* Hahn-Grant solver

In Sec. 4, we recap the Hahn-Grant solver [13] and introduce our *factorized* Hahn-Grant solver. Here, we provide more details on the implementation of the algorithms and show that both algorithms result in the same lower bounds. We also include an empirical study supporting the design choices behind our factorized Hahn-Grant solver: the factorization, faster LAP solvers, and finding primal solutions.

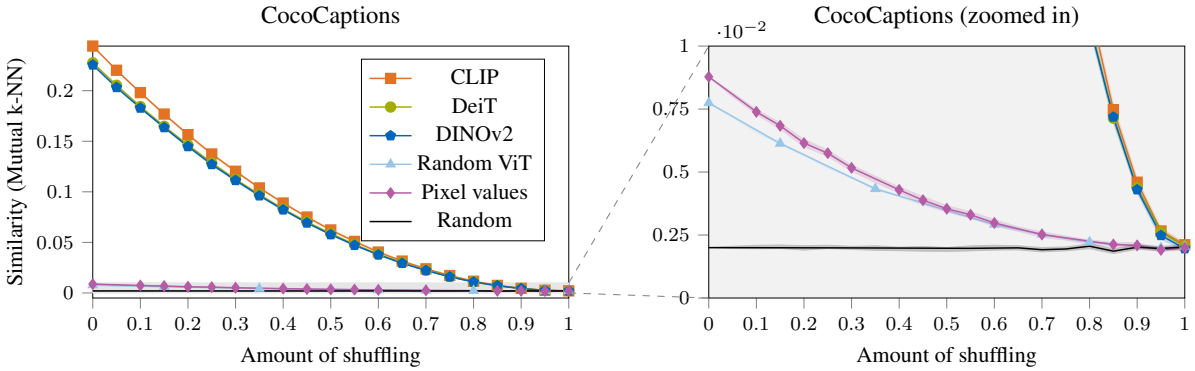

Figure 8. **Randomly initialized networks behave similarly to the pre-trained ones:** Zooming in (right) on the randomly initialized ViT, we observe a strikingly similar qualitative behavior – the similarity decreases monotonically with the increased degree of shuffling. This observation also holds for the curve resulting from raw pixel values. This surprising phenomenon presumably originates from the properties of the natural image manifold and the Lipschitz-continuity of neural networks.

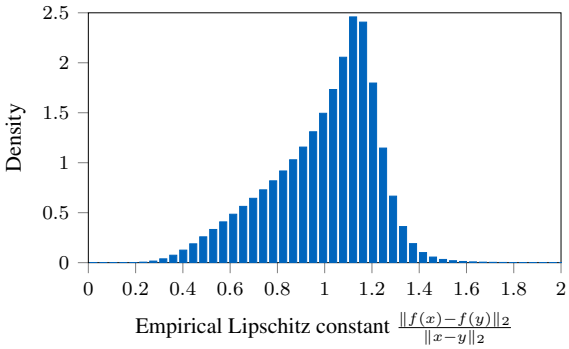

Figure 9. **Random networks roughly preserve distances:** Observe that the mode of the empirical Lipschitz constant is close to 1, which suggests that network encoding approximately preserves the distances in the input domain.

D.1. Proof of equivalence to Hahn-Grant solver

The main idea of the proof is that the lower bound, the `leader`, and all LAPs are the same for both algorithms. The main difference is that the dual vectors are stored in the tensors $\mathbf{U}$ and $\mathbf{V}$ instead of updating $\mathbf{C}$ in-place.

First, we assume that $\mathbf{C}_{ijkl} = \mathbf{C}^{(1)}_{ik}\mathbf{C}^{(2)}_{jl}$ with $\mathbf{C}^{(1)}$ and $\mathbf{C}^{(2)}$ being non-negative and symmetric. Given this assumption, we will show that Algorithm 1 and Algorithm 2 are equivalent without our adaptions (Lines 4, 7, and 14) and using the Hungarian matching as the LAP solver. We will show that both algorithms have the same lower bound l, the `leader`, and the sum of complementary costs $\mathbf{C}_{ijkl} + \mathbf{C}_{klij}$. In particular, denoting the variables from Algorithm 2 with a bar, we have

$$l = \bar{l}, \quad \texttt{leader} = \overline{\texttt{leader}}, \quad \text{and} \tag{27}$$
$$\mathbf{C}_{ijkl} + \mathbf{C}_{klij} = 2\overline{\mathbf{C}}^{(1)}_{ik}\overline{\mathbf{C}}^{(2)}_{jl} - \overline{\mathbf{U}}_{ijk} - \overline{\mathbf{V}}_{ijl} - \overline{\mathbf{U}}_{kli} - \overline{\mathbf{V}}_{klj}.$$

for all $i \neq k, j \neq l \in [N]$. Starting with the first iteration until after Algorithm 1, Line 5 and Algorithm 2, Line 5, we have that the lower bound $l = \bar{l} = 0$ and

$$\texttt{leader} = \mathbf{C}_{ijij} = \mathbf{C}^{(1)}_{ii}\mathbf{C}^{(2)}_{jj} = \overline{\texttt{leader}} \tag{28}$$

are initialized the same. Furthermore, because $\overline{\mathbf{U}} = \overline{\mathbf{V}} = 0$, we have that

$$\mathbf{C}_{ijkl} = \mathbf{C}^{(1)}_{ik}\mathbf{C}^{(2)}_{jl} - \overline{\mathbf{U}}_{ijk} - \overline{\mathbf{V}}_{ijl}. \tag{29}$$

Thus, both algorithms have an equivalent starting condition.

Next, given that Eq. (27) holds, we show that both values are changed in an equivalent way. Given the same `leader` matrix, Line 6-8 from Algorithm 1 change l and `leader` in the same way as Lines 6-9 from Algorithm 2. Moreover, after Algorithm 1, Line 9 and Algorithm 2, Line 10, $\mathbf{C}_{ijkl}$ and $\mathbf{U}_{ijk}$ are changed in the same way. Therefore, the sum is also preserved:

$$\begin{aligned}
&\mathbf{C}_{ijkl} + \mathbf{C}_{klij} \\
=&\mathbf{C}^{\text{prev}}_{ijkl} + \mathbf{C}^{\text{prev}}_{klij} + \frac{\texttt{leader}_{ij}}{N-1} + \frac{\texttt{leader}_{kl}}{N-1} \\
=&2\overline{\mathbf{C}}^{(1)}_{ik}\overline{\mathbf{C}}^{(2)}_{jl} - \overline{\mathbf{U}}^{\text{prev}}_{ijk} + \frac{\texttt{leader}_{ij}}{N-1} - \overline{\mathbf{V}}_{ijl} \\
&- \overline{\mathbf{U}}^{\text{prev}}_{kli} + \frac{\texttt{leader}_{kl}}{N-1} - \overline{\mathbf{V}}_{klj} \\
=&2\overline{\mathbf{C}}^{(1)}_{ik}\overline{\mathbf{C}}^{(2)}_{jl} - \overline{\mathbf{U}}_{ijk} - \overline{\mathbf{V}}_{ijl} - \overline{\mathbf{U}}_{kli} - \overline{\mathbf{V}}_{klj},
\end{aligned} \tag{30}$$

where the superscipt $(^{\text{prev}})$ denotes the value before executing the line.

In Line 11 and Line 12 of Algorithm 1, the values from the complementary positions are redistributed to the current submatrix. Line 12 from Algorithm 2 also aggregates the sum of the complementary elements, albeit not by changing the cost matrices. Therefore, the sum of both elements

Algorithm 1 Hahn-Grant solver [13]

Input: $\mathbf{C} \in \mathbb{R}^{N\times N\times N\times N}_{\geq 0}$ cost tensor
Output: $l \leq \arg\min_{\mathbf{P}\in\mathcal{P}_N} \sum_{i,j,k,l=1}^{N} \mathbf{C}_{ijkl}\mathbf{P}_{ij}\mathbf{P}_{kl}$
$l \leftarrow 0$
while not converged **do**
  $\texttt{leader}_{ij} \leftarrow \mathbf{C}_{ijij}$ for $i,j \in [N]$
  $\mathbf{u}, \mathbf{v}, \_ \leftarrow$ hungarian_matching($\texttt{leader}$)
  $l \leftarrow l + \sum_i \mathbf{u}_i + \sum_j \mathbf{v}_j$
  $\texttt{leader}_{ij} \leftarrow \texttt{leader}_{ij} - \mathbf{u}_i - \mathbf{v}_j$ for $i,j \in [N]$
  $\mathbf{C}_{ijkl} \leftarrow \mathbf{C}_{ijkl} + \frac{\texttt{leader}_{ij}}{N-1}$ for $i \neq k, j \neq l \in [N]$
  for $i,j \in [N]$ **do**
    $\mathbf{C}_{ijkl} \leftarrow \mathbf{C}_{ijkl} + \mathbf{C}_{klij}$ for $i \neq k, j \neq l \in [N]$
    $\mathbf{C}_{klij} \leftarrow 0$ for $i \neq k, j \neq l \in [N]$
    $\mathbf{u}, \mathbf{v}, \_ \leftarrow$ hungarian_matching($\mathbf{C}_{i,j,[N]\setminus\{i\},[N]\setminus\{j\}}$)
    $\mathbf{C}_{ijij} \leftarrow \sum_k \mathbf{u}_k + \sum_l \mathbf{v}_l$
    $\mathbf{C}_{ijkl} \leftarrow \mathbf{C}_{ijkl} - \mathbf{u}_k - \mathbf{v}_l$ for $i \neq k, j \neq l \in [N]$

Algorithm 2 Factorized Hahn-Grant solver (Ours)

Input: $\mathbf{C}^{(1)}, \mathbf{C}^{(2)} \in \mathbb{R}^{N\times N}_{\geq 0}$ symmetric cost tensors
Output: $l \leq \arg\min_{\mathbf{P}\in\mathcal{P}_N} \sum_{i,j,k,l=1}^{N} \mathbf{C}^{(1)}_{ik}\mathbf{C}^{(2)}_{jl}\mathbf{P}_{ij}\mathbf{P}_{kl}$, $\mathbf{P}^* \in \mathcal{P}_N$ permutation matrix
$l \leftarrow 0$; $\mathbf{U}, \mathbf{V} \leftarrow \mathbf{0}_{N\times N\times N-1}$; $\texttt{leader}_{ij} \leftarrow \mathbf{C}^{(1)}_{ii}\mathbf{C}^{(2)}_{jj}$
$\mathbf{P}^* \leftarrow$ primal_heuristic($\mathbf{C}^{(1)}, \mathbf{C}^{(2)}$)
while not converged **do**
  $\mathbf{u}, \mathbf{v}, \mathbf{P} \leftarrow$ lap_solver($\texttt{leader}$)
  $\mathbf{P}^* \leftarrow$ better($\mathbf{P}^*, \mathbf{P}$)
  $l \leftarrow l + \sum_i \mathbf{u}_i + \sum_j \mathbf{v}_j$
  $\texttt{leader}_{ij} \leftarrow \texttt{leader}_{ij} - \mathbf{u}_i - \mathbf{v}_j$ for $i,j \in [N]$
  $\mathbf{U}_{ijk} \leftarrow \mathbf{U}_{ijk} - \frac{\texttt{leader}_{ij}}{N-1}$ for $k \neq i, j \in [N]$
  for $i,j \in [N]$ **do**
    $\mathbf{C}^{\text{tmp}}_{kl} \leftarrow 2\mathbf{C}^{(1)}_{ik}\mathbf{C}^{(2)}_{jl} - \mathbf{U}_{ijk} - \mathbf{V}_{ijl} - \mathbf{U}_{kli} - \mathbf{V}_{klj}$ for $i \neq k, j \neq l \in [N]$
    $\mathbf{u}, \mathbf{v}, \mathbf{P} \leftarrow$ lap_solver($\mathbf{C}^{\text{tmp}}$)
    $\mathbf{P}^* \leftarrow$ better($\mathbf{P}^*, \mathbf{P}$)
    $\texttt{leader}_{ij} \leftarrow \sum_k \mathbf{u}_k + \sum_l \mathbf{v}_l$
    $\mathbf{U}_{ijk} \leftarrow \mathbf{U}_{ijk} + \mathbf{u}_k$ for $i \neq k \in [N]$
    $\mathbf{V}_{ijl} \leftarrow \mathbf{V}_{ijl} + \mathbf{v}_l$ for $j \neq l \in [N]$

Figure 10. **The Hahn-Grant solver (left) and the factorized Hahn-Grant solver (ours, right):** The Hahn-Grant solver [13] iteratively improves the dual bound of the QAP by solving linear assignment problems (LAPs). Our solver improves the memory requirements of the Hahn-Grant solver for factorized cost matrices, introduces a primal heuristic that reuses the assignment from the LAPs, and uses a faster solver for the LAPs.

remains the same. By definition of $\mathbf{C}^{\text{tmp}}$, it follows, that $\mathbf{C}_{ijkl} = \mathbf{C}^{\text{tmp}}_{kl}$ and the Hungarian matching produces the same values for both algorithms. As the next step, the objective value is added to the $\texttt{leader}$ in Algorithm 1 by first adding it to $\mathbf{C}_{ijij}$ in Line 14 and then to $\texttt{leader}_{ij}$ in Line 5 in the next iteration. In Algorithm 2, this value is directly added to $\texttt{leader}_{ij}$. Finally, the dual variables are subtracted from the cost in Line 15 of Algorithm 1 and Line 16 and Line 17 of Algorithm 2. This preserves the sum:

$$\begin{aligned}
&\mathbf{C}_{ijkl} + \mathbf{C}_{klij} = \\
&\quad = \mathbf{C}^{\text{prev}}_{ijkl} - \mathbf{u}_k - \mathbf{v}_l + \mathbf{C}^{\text{prev}}_{klij} - \mathbf{u}_i - \mathbf{v}_j \\
&\quad = 2\overline{\mathbf{C}}^{(1)}_{ik}\overline{\mathbf{C}}^{(2)}_{jl} - \overline{\mathbf{U}}^{\text{prev}}_{ijk} - \mathbf{u}_k - \overline{\mathbf{V}}^{\text{prev}}_{ijl} - \mathbf{v}_l \\
&\qquad - \overline{\mathbf{U}}^{\text{prev}}_{kli} - \mathbf{u}_i - \overline{\mathbf{V}}^{\text{prev}}_{klj} - \mathbf{v}_j \\
&\quad = 2\overline{\mathbf{C}}^{(1)}_{ik}\overline{\mathbf{C}}^{(2)}_{jl} - \overline{\mathbf{U}}_{ijk} - \overline{\mathbf{V}}_{ijl} - \overline{\mathbf{U}}_{kli} - \overline{\mathbf{V}}_{klj}.
\end{aligned} \tag{31}$$

Therefore, each iteration in both algorithms changes the costs in an equivalent way. As a result, the final bound l and each solution to the LAPs are the same. □

In practice, we can also remove the non-negativity constraint because adding a constant to $\mathbf{C}^{(1)}$ or $\mathbf{C}^{(2)}$ leads to an equivalent optimization problem with an additional constant term, *i.e.*

$$\begin{aligned}
&\sum_{i,j,k,l=1}^{N} (\mathbf{C}^{(1)}_{ik} + c)\mathbf{C}^{(2)}_{jl}\mathbf{P}_{ij}\mathbf{P}_{kl} = \\
&\qquad = \sum_{i,j,k,l=1}^{N} \mathbf{C}^{(1)}_{ik}\mathbf{C}^{(2)}_{jl}\mathbf{P}_{ij}\mathbf{P}_{kl} + c\sum_{j,l=1}^{N} \mathbf{C}^{(2)}_{jl}.
\end{aligned} \tag{32}$$

Therefore, we can subtract the minimal element from both matrices, apply the algorithm to the resulting non-negative matrices, and subtract the constant from Eq. (32) to the final objective value to retrieve the optimal objective value of the original problem.

D.2. Implementation details

We implement both Algorithm 2 and Algorithm 1 in Python using PyTorch [55]. The main computational bottleneck is the LAP solver. Therefore, we use a custom C++ algorithm for the forward-reverse auction [56] algorithm and for the Jonker-Volgenant algorithm [19, 31]. We stop the algorithm if the relative or absolute improvement of the dual bound l is smaller than $\varepsilon = 1e-6$ or the primal objective is within ε of the dual bound. Finally, for the auction algorithm, a larger relaxation ε^{auc} usually leads to a faster runtime with the drawback of worse objectives and dual vectors. The Hahn-Grant algorithm also works for suboptimal dual vectors, but we observe that towards the end of the algorithm,

Solver	Cost		Bound	
Hahn-Grant	-1.979877		–	
+ factorized	-1.979875 ↑	$2.0e^{-6}$	-2.089994 ↑	
+ auction	-1.979871 ↑	$4.3e^{-6}$	-1.981942 ↑	$1.1e^{-1}$
+ Jonker-Volgenant	-1.979878 ↓	$-7.3e^{-6}$	-1.980387 ↑	$1.6e^{-3}$
+ LAP solutions	-1.979914 ↓	$-3.5e^{-5}$	-1.980387 =	
+ primal heuristic	-1.980015 ↓	$-1.0e^{-4}$	-1.980387 =	

Table 4. **Cost and bounds for our Hahn-Grant adaptation ($N = 100$):** The factorization and the auction algorithm slightly increase the cost of the solution, while leading to a small bound. The Jonker-Volgenant and especially the LAP solutions and primal heuristics also lead to better primal solutions.

Solver	Cost		Bound	
Hahn-Grant	-1.950160		-2.245823	
+ factorized	-1.950166 ↓	$-6.0e^{-6}$	-1.951418 ↑	$2.9e^{-1}$
+ auction	-1.950197 ↓	$-3.2e^{-5}$	-1.950586 ↑	$8.3e^{-4}$
+ Jonker-Volgenant	-1.950197 ↑	$3.9e^{-7}$	-1.950623 ↓	$-3.8e^{-5}$
+ LAP solutions	-1.950268 ↓	$-7.0e^{-5}$	-1.950623 =	
+ primal heuristic	-1.950360 ↓	$-9.2e^{-5}$	-1.950623 =	

Table 5. **Cost and bounds for our Hahn-Grant adaptation ($N = 40$):** Our adaptations exhibit strong benefits either in terms of the solution cost or the tightness of the bound, frequently both. The auction algorithm leads to tighter bounds than the Jonker-Volgenant algorithm.

better solutions in the LAPs are required. Therefore, we initialize $\varepsilon^{\text{auc}} = 0.1$ relatively high in the beginning and multiply it with a factor of 0.9 in every iteration.

D.3. Ablations

Setup. We compare the original Hahn-Grant solver with our adaptation from Sec. 4: the factorization into matrices $\mathbf{C}^{(1)}$ and $\mathbf{C}^{(2)}$, using the auction algorithm or the Jonker-Volgenant as faster LAP solvers (Line 6 and Line 13), evaluating the individual LAP primal solutions (Line 7 and Line 14), and using primal heuristics (Line 4). For these experiments, we generate two sets with 100 vectors of dimensionality 1024 each, drawn element-wise from a standard normal distribution. These vectors are normalized, and the pairwise inner product is computed to produce two 100×100 similarity matrices. We apply all variations of the algorithms with a time limit of two hours to these cost matrices and evaluate the quality of the primal solutions and of the bound. We repeat the experiment with 5 random seeds and average the results. We also repeat the experiment with a set of 40 random vectors and a time limit of one hour.

Results. Tab. 4 shows the result for each adaptation for size 100 and Tab. 5 for size 40. We observe that the original Hahn-Grant solver reaches a non-trivial primal-dual bound for size 40 but does not finish the first iteration within the two hour time limit for size 100. Even though factorization was introduced for improved memory, it also slightly speeds up the computation, leading to a smaller bound for both sizes. The small increase in the objective for size 100 can be explained by the fact that the returned primal estimate is only the primal solution of the LAP for the `leader`. Therefore, the quality of this estimate can vary with every iteration. The auction and Jonker-Volgenant algorithms were introduced to improve the speed of the algorithm and, therefore, lead to smaller bounds. However, the Jonker-Volgenant algorithm leads to the tightest bounds empirically for large problems while the auction algorithm is slightly better on the size 40 problem. The LAP solutions do not change the dual estimate but improve the primal solution by a significant margin. Finally, the primal heuristics further improve the primal solution. Since we are mostly interested in good primal solutions and measuring the quality of the solution, we use the algorithm with all the adaptations.

E. Experimental details

In this section, we provide more details supplementing the experiments in Sec. 5, which were excluded from the main text due to space constraints.

Our method only assumes pre-computed embeddings (or already pre-computed similarity matrices). Therefore, we are not limited to specific architectures or pre-training strategies. Our goal is to choose a variety of different architectures, pre-trainings, and model sizes for both modalities.

Vision models. We consider self-supervised, fully-supervised and vision-language supervised models with convolutional and vision transformer architectures of different capacities. For self-supervised methods, we consider different models from DINO [3] trained on ImageNet-1k [44] and models from DINOv2 [36] trained on LVD-142M. For supervised models, DeiT [48] and ConvNeXt [27] pre-trained on ImageNet-1k, ImageNet-22k [44] and a a combination of both are used. Furthermore, we choose CLIP [42] as our vision-language model with both ResNet and ViT backbones. We use the code and models from the official repository except for ConvNeXt, where we use pre-trained models from the `timm` library. In total, we use 32 vision models.

Language models. Similar to the vision models, we consider different pre-trainings and network sizes. In particular, in addition the the CLIP [42] text models, contrastive learning, question-answer models, and average word embeddings are considered. We use the official CLIP repository for all CLIP text models and the SentenceTransformers [43] library for all other models. In total, we use 27 language models.

Datasets. We evaluate our experiments on CIFAR-10 [23], CINIC-10 [9], CIFAR-100 [23], and ImageNet-100 [44].

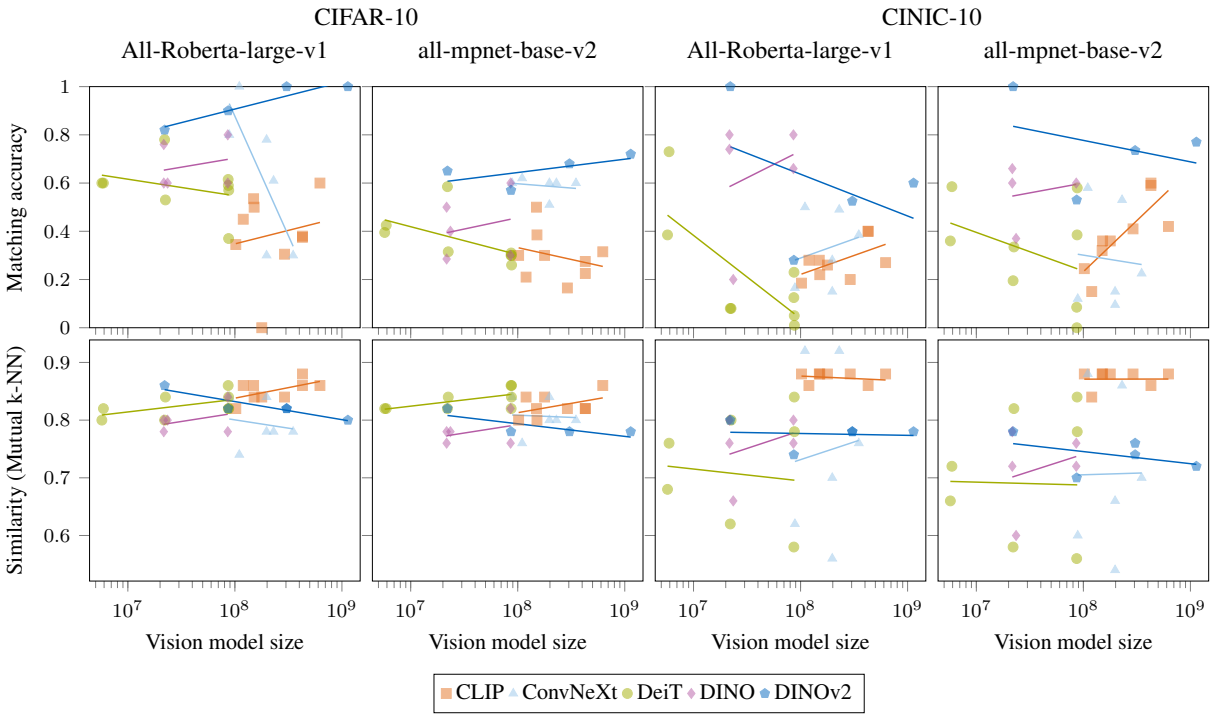

Figure 11. **Matching accuracy and Mutual k-NN on CIFAR-10 and CINIC-10:** Larger models do not necessarily lead to stronger alignment with language. Instead, it is the pre-training method that characterizes the vision-language alignment most. Here, DINOv2 yields the highest matching accuracy for both datasets and language models.

For ImageNet-100, we use the validation split and for all other datasets, we use the test split. We choose the same class names and prompts as ASIF [35] whenever available, and follow the same preprocessing as CLIP otherwise. For ImageNet-100, we observed an improved performance by encoding each class as "<name>: <definition>" using the WordNet definitions and more descriptive class names.

General setup. We implement all experiments in Python with PyTorch [55]. In general, we pre-compute the embeddings for each vision and language model and every dataset using PyTorch Lightning [57]. We do not use GPUs except for precomputing the embeddings. After computing the image-wise embeddings, we normalize them and average them for every class, see Eq. (1). The resulting object-wise embedding is again normalized. To evaluate the impact of small changes in the embeddings (and pairwise similarities), we only take a random subset of the image representations to compute the average. In particular, we use a half of the embeddings drawn uniformly for every random seed. For the language embeddings, we take the average over the embeddings for all different prompts.

Small-scale matching. We use 20 random seeds and all models on CIFAR-10 and CINIC-10 for small-scale matching. In each experiment, we enumerate all permutations and compute all costs explicitly, returning the permutation with the minimal cost. The comparison of all combinations of vision and language models is given in Appx. F.1.

Larger-scale matching. For larger-scale matching, we use different subsets of classes using the optimization problem introduced in Sec. 4.2 solved with Gurobi [12]. For each problem size $N \in \{10, 20, \dots, 100\}$, we use the 10 best subset of classes, each with 3 random seeds for computing the object-level vision embeddings. We use our factorized Hahn-Grant solver and a time limit of one hour for each matching problem. The models are DINOv2 ViT-G/14, CLIP ViT-L/14@336px, and the distilled DeiT-B/16@384px with all-mpnet-base-v2.

Solver comparison. We evaluate the solvers both on small-scale and larger-scale matching. For small-scale matching, we use 20 random seeds for DINOv2 ViT-G/14, CLIP ViT-L/14@336px, and the distilled DeiT-B/16@384px with all-mpnet-base-v2 and All-Roberta-large-v1 on CIFAR-10 and CINIC-10. We present the results for all combinations in F.3. The larger-scale benchmark follows the setting of larger-scale matching, but only considers one out of the ten subsets and one random seed

Solver	Accuracy (%)	Matching Accuracy (%)
Random	12.1±10.3	12.5±11.6
LocalCKA	9.1±7.0	8.0±7.7
OT	5.4±9.5	4.5±9.4
FAQ	6.5±8.7	6.0±9.9
MPOpt	16.0±21.4	17.0±23.9
Ours	**46.7±3.4**	**51.5±3.7**
GT	85.2±0.6	100.0±0.0

Table 6. **Solver comparison on unsupervised classification:** Using DeiT and All-Roberta-large-v1, we show the accuracy of the unsupervised classifier for different solvers. The right column further shows how many of the centroids are mapped to the best class for the given cluster in line with the matching accuracy from the unsupervised matching experiments. Our QAP solver achieves a considerably higher matching and classification accuracy than the other methods.

for CIFAR-100 using CLIP CLIP ViT-L/14@336px and all-mpnet-base-v2 with a time limit of 1.5 hours.

Unsupervised classifier. For unsupervised classification, we use K-Means [28] to cluster image embeddings into prototype (object) embeddings. We use K-Means++ [58] from Scikit-learn [54] with 100 initializations. The cluster centers are then matched to the language embeddings using our factorized Hahn-Grant solver. Similar to the previous experiments, we only use a random 50% subset of the vision embeddings and evaluate the method for 20 random seeds. We report the results for DINOv2 ViT-G/14, CLIP ViT-L/14@336px, and the distilled DeiT-B/16@384px with all-mpnet-base-v2 and All-Roberta-large-v1 on CIFAR-10.

F. Evaluation results

Following up on Sec. 5, we report the results for all vision and language models in the small-scale matching setting (*cf*. Appx. F.1) and benchmark of the different solvers using multiple datasets and models (*cf*. Appx. F.2 and Appx. F.3).

F.1. Comparison of vision and language models

In this section, we report the results on the small-scale matching, spanning all vision and language models in our study. The results show, in particular, that DINOv2 outperforms the other models on both datasets and that the model size is less important than the pre-training method.

In Fig. 11, we show the matching accuracy (top) and Mutual k-NN with $k = 5$ (bottom) of each vision model in combination with All-Roberta-large-v1 (left) and all-mpnet-base-v2 (right) for CIFAR-10 (left) and CINIC-10 (right). The lines show the trend for increasing model sizes that are fitted to the models of varying capacity for each model class (different colors). The model size corresponds to the number of parameters. First, we observe that for both datasets and language models, DINOv2 yields the highest matching accuracy. Furthermore, for every model class, there is no a clear propensity of larger models to perform better. As this seems to contradict the platonic representation hypothesis Huh et al. [17], we report the Mutual k-NN in the bottom plots. We observe that in line with our conclusions, there is still no clear improvement of the Mutual k-NN metric w.r.t. an increasing model capacity. This implies that larger models do not necessarily lead to stronger alignment with language – at least on CIFAR-10 and CINIC-10. Considering the observations by Huh et al. [17], this suggests that scaling the models could lead to a better alignment on the Wit dataset [59], even though it may not be sufficient to improve the alignment on every other dataset. We report the individual matching accuracy of each combination of vision and language model for CIFAR-10 in Fig. 12 and for CINIC-10 in Fig. 13.

F.2. Unsupervised classification: solver comparison

We also compare the solvers on the unsupervised classification setting in Tab. 6. In addition to the classification accuracy, we also report the matching accuracy. Given a clustering and the ground truth labels, we compute the optimal ground truth matching. The matching accuracy evaluates how well our unsupervised matching coincides with the ground truth matching. Finally, we report the performance when the ground truth matching is used instead of our unsupervised matching to evaluate the quality of the clustering.

Similar to the unsupervised matching experiments, we observe that our solver outperforms the other solvers in terms of both classification accuracy and matching accuracy. This means that our solver finds matches that agree well with the ground truth matches. We also observe that the k-means clustering is not perfect, and approximately 15% of the images are clustered in the wrong clusters. This hurts our clustering in two ways. First, since only whole clusters are assigned to classes, misclassified images will remain in the respective (wrong) cluster. Second, incorrect samples distort the centroids, which in turn can further affect the pairwise distances, leading to a worse matching. However, even with imperfect centroids, our matching finds non-trivial matches and leads to the first non-trivial unsupervised classification, which was not possible using previous local solvers.

F.3. Comparison of solvers on small-scale problems

Similar to Tab. 1, we benchmark different solvers for all combinations in Tab. 7. Our factorized Hahn-Grant solver and Gurobi [12] find the global optimum for all problems. For most combinations, these solvers are also the only solvers finding the global optimum. MPOpt [18] also finds the global optimum for some problems (*e.g.*, CIFAR-10 with DINOv2) but fails for other problems (*e.g.*, CIFAR-10 with CLIP). For all but two experiments (CIFAR-10 with DeiT and all-mpnet-base-v2 and CINIC-10 with DeiT and

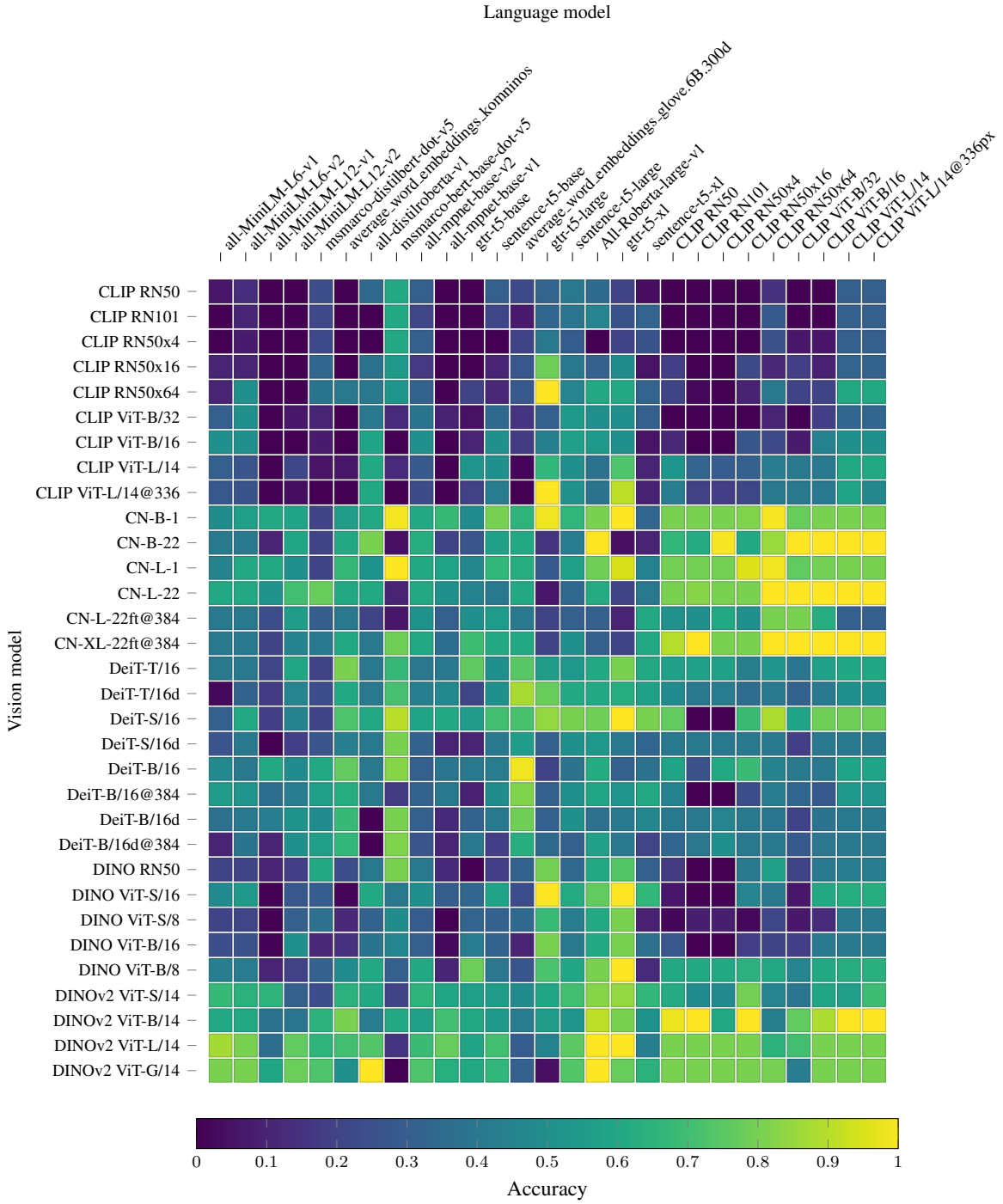

Figure 12. **Vision-language alignment accuracy on CIFAR-10:** We observe that the pre-training strategy tends to have a more impactful role on vision-language alignment than the model capacity. Here, DINOv2 and ConvNeXt (*e.g.* CN-B-1) families exhibits a prominent degree of image-text alignment.

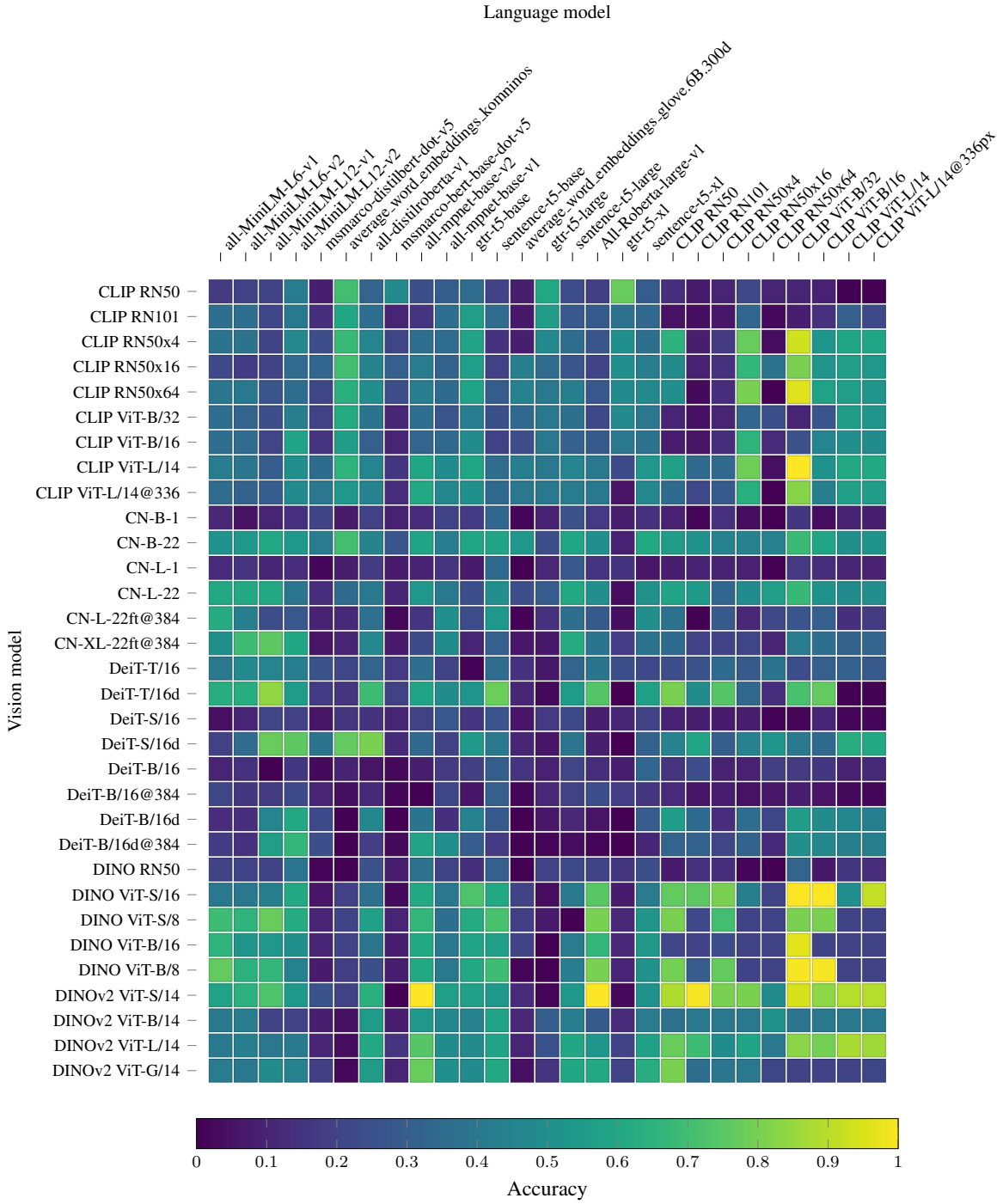

Figure 13. **Vision-language alignment accuracy on CINIC-10:** Here, DINO-based pre-training in conjunction with the CLIP-based text embedding exhibit a higher degree of conformity.

All-Roberta-large-v1), the global optimum also leads to the best matching accuracy, which underlines the importance of the Gromov-Wasserstein distance as a proxy measure for blind matching. Finally, the matching accuracy is close to random for most solvers (LocalCKA [30], OT [38], and FAQ [49]). This shows that a global solver is crucial to obtain non-trivial results.

G. Optimal transport as QAP solver

We show that optimal transport with uniform source and target probability distributions is equivalent to relaxing the QAP with doubly stochastic matrices. In our experiments, we use this equivalence to compare to the solutions from optimal transport solvers.

For uniform probability distributions $\mathbf{p} = \mathbf{q} = \frac{1}{N}\mathbb{1}$, let

$$\mathbf{T}^* \in \underset{\substack{\mathbf{T}\in[0,1]^{N\times N}\\ \mathbf{T}\mathbb{1}=\mathbf{p}\\ \mathbf{T}^T\mathbb{1}=\mathbf{q}}}{\arg\min} \mathbf{L}_{ijkl}\mathbf{T}_{ij}\mathbf{T}_{kl} \tag{33}$$

be an optimal transport matrix for the four axes tensor $\mathbf{L} \in I\!R^{N\times N\times N\times N}$. Then, for any other transport matrix $\mathbf{T} \in [0,1]^{N\times N}$ with $\mathbf{T}\mathbb{1} = \mathbf{p}$ and $\mathbf{T}^T\mathbb{1} = \mathbf{q}$, it holds that

$$\mathbf{L}_{ijkl}\mathbf{T}_{ij}\mathbf{T}_{kl} \geq \mathbf{L}_{ijkl}\mathbf{T^*}_{ij}\mathbf{T^*}_{kl}. \tag{34}$$

Now, we can define our optimal stochastic matrix $\mathbf{S}^* \in [0,1]^{N\times N}$ by $\mathbf{S}^* = N\mathbf{T}^*$. This is indeed a stochastic matrix as $N\mathbf{T}^* \geq 0$, $\mathbf{S}^*\mathbb{1} = N\mathbf{T}^*\mathbb{1} = N\mathbf{p} = \mathbb{1}$ and $(\mathbf{S}^*)^T\mathbb{1} = N\mathbf{T}^*\mathbb{1} = N\mathbf{q} = \mathbb{1}$. Moreover, this stochastic matrix is optimal, because for every other stochastic matrix $\mathbf{S} \in [0,1]^{N\times N}$,

$$\begin{aligned}\mathbf{L}_{ijkl}\mathbf{S}_{ij}\mathbf{S}_{kl} =& N^2\mathbf{L}_{ijkl}\frac{\mathbf{S}_{ij}}{N}\frac{\mathbf{S}_{kl}}{N} \geq \\ \geq& N^2\mathbf{L}_{ijkl}\mathbf{T^*}_{ij}\mathbf{T^*}_{kl} = \mathbf{L}_{ijkl}\mathbf{S^*}_{ij}\mathbf{S^*}_{kl}.\end{aligned} \tag{35}$$

Here, we use the fact that $\frac{\mathbf{S}}{N}$ is a valid transport matrix as $\frac{\mathbf{S}}{N}\mathbb{1} = \frac{1}{N}\mathbb{1} = \mathbf{p}$ and $\frac{\mathbf{S}^T}{N}\mathbb{1} = \frac{1}{N}\mathbb{1} = \mathbf{q}$. The other direction can be derived in a similar way by dividing the doubly stochastic matrix by N.

References

[54] Fabian Pedregosa, Gaël Varoquaux, and Alexandre Gramfort, et al. Scikit-learn: Machine learning in Python. In *JMLR*, 2011. viii

[55] Adam Paszke, Sam Gross, and Francisco Massa, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. v, vii

[56] Dimitri P. Bertsekas and David A. Castanon. In *Comput. Optim. Appl. 1*, pages 277-297, 1992. v

[57] William Falcon et al. PyTorch Lightning. https://github.com/Lightning-AI/lightning, 2019. vii

[58] Arthur David and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proc. 18th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1027–1035, 2007. viii

[59] Krishna Srinivasan, Karthik Raman, Jiecao Chen, et al. Wit: Wikipedia-based image text dataset for multimodal multilingual machine learning. In *SIGIR*, pages 2443-2449. 2021. viii

			All-Roberta-large-v1			all-mpnet-base-v2		
			Accuracy (%)	Cost	Global? (%)	Accuracy (%)	Cost	Global? (%)
CIFAR-10	CLIP	Random	8.5±10.4	1.77±0.32	0.0±0.0	8.5±10.4	1.8 ±0.29	0.0±0.0
		LocalCKA [30]	9.5±8.87	1.82±0.33	0.0±0.0	12.5±11.64	1.72±0.31	0.0±0.0
		OT [38]	11.0±3.08	0.67±0.03	0.0±0.0	0.0±0.0	0.36±0.01	0.0±0.0
		FAQ [49]	1.5±3.66	0.78±0.06	0.0±0.0	5.0±8.89	0.36±0.02	20.0±41.04
		MPOpt [18]	0.0±0.0	0.51±0.01	0.0±0.0	0.0±0.0	0.6 ±0.01	0.0±0.0
		Gurobi [12]	**38.0±10.05**	**0.4 ±0.01**	**100.0±0.0**	**22.5±4.44**	**0.33±0.01**	**100.0±0.0**
		Ours	**38.0±10.05**	**0.4 ±0.01**	**100.0±0.0**	**22.5±4.44**	**0.33±0.01**	**100.0±0.0**
	DeiT	Random	8.5±10.4	1.77±0.29	0.0±0.0	8.5±10.4	1.81±0.27	0.0±0.0
		LocalCKA [30]	10.0±9.18	1.85±0.2	0.0±0.0	6.0±6.81	1.77±0.22	0.0±0.0
		OT [38]	0.0±0.0	0.84±0.01	0.0±0.0	19.0±3.08	0.28±0.01	30.0±47.02
		FAQ [49]	22.5±10.7	0.36±0.13	0.0±0.0	**37.0±14.9**	0.3 ±0.01	0.0±0.0
		MPOpt [18]	0.0±0.0	0.74±0.02	0.0±0.0	0.0±0.0	0.72±0.01	0.0±0.0
		Gurobi [12]	**57.0±4.7**	**0.27±0.01**	**100.0±0.0**	26.0±6.81	**0.27±0.0**	**100.0±0.0**
		Ours	**57.0±4.7**	**0.27±0.01**	**100.0±0.0**	26.0±6.81	**0.27±0.0**	**100.0±0.0**
	DINOv2	Random	8.5±10.4	1.8 ±0.27	0.0±0.0	8.5±10.4	1.82±0.28	0.0±0.0
		LocalCKA [30]	14.0±11.42	1.75±0.31	0.0±0.0	7.5±9.1	1.67±0.24	0.0±0.0
		OT [38]	24.5±26.25	0.63±0.44	0.0±0.0	24.0±24.15	1.13±0.54	5.0±22.36
		FAQ [49]	34.0±35.3	0.49±0.18	5.0±22.36	39.0±23.82	0.65±0.22	0.0±0.0
		MPOpt [18]	**100.0±0.0**	**0.33±0.01**	**100.0±0.0**	**72.0±17.65**	**0.45±0.01**	**100.0±0.0**
		Gurobi [12]	**100.0±0.0**	**0.33±0.01**	**100.0±0.0**	**72.0±17.65**	**0.45±0.01**	**100.0±0.0**
		Ours	**100.0±0.0**	**0.33±0.01**	**100.0±0.0**	**72.0±17.65**	**0.45±0.01**	**100.0±0.0**
CINIC-10	CLIP	Random	8.5±10.4	1.76±0.31	0.0±0.0	8.5±10.4	1.73±0.3	0.0±0.0
		LocalCKA [30]	14.5±8.87	1.81±0.25	0.0±0.0	10.0±8.58	1.77±0.27	0.0±0.0
		OT [38]	1.5±3.66	1.06±0.13	0.0±0.0	1.5±3.66	0.64±0.11	0.0±0.0
		FAQ [49]	27.5±5.5	0.37±0.02	0.0±0.0	1.0±3.08	0.62±0.08	0.0±0.0
		MPOpt [18]	39.5±2.24	0.36±0.04	95.0±22.36	49.0±12.1	0.36±0.02	50.0±51.3
		Gurobi [12]	**40.0±0.0**	**0.35±0.01**	**100.0±0.0**	**60.0±0.0**	**0.34±0.02**	**100.0±0.0**
		Ours	**40.0±0.0**	**0.35±0.01**	**100.0±0.0**	**60.0±0.0**	**0.34±0.02**	**100.0±0.0**
	DeiT	Random	8.5±10.4	1.73±0.25	0.0±0.0	8.5±10.4	1.74±0.25	0.0±0.0
		LocalCKA [30]	9.0±7.88	1.9 ±0.17	0.0±0.0	6.5±8.75	1.66±0.22	0.0±0.0
		OT [38]	**37.0±6.57**	0.38±0.02	0.0±0.0	3.0±4.7	0.51±0.05	0.0±0.0
		FAQ [49]	29.5±17.01	0.38±0.07	0.0±0.0	9.0±3.08	0.58±0.04	0.0±0.0
		MPOpt [18]	3.5±4.89	**0.31±0.01**	55.0±51.04	**58.0±6.16**	**0.49±0.01**	**100.0±0.0**
		Gurobi [12]	1.0±3.08	**0.31±0.01**	**100.0±0.0**	**58.0±6.16**	**0.49±0.01**	**100.0±0.0**
		Ours	1.0±3.08	**0.31±0.01**	**100.0±0.0**	**58.0±6.16**	**0.49±0.01**	**100.0±0.0**
	DINOv2	Random	8.5±10.4	1.79±0.23	0.0±0.0	8.5±10.4	1.76±0.23	0.0±0.0
		LocalCKA [30]	10.5±9.45	1.96±0.14	0.0±0.0	13.0±7.33	1.68±0.21	0.0±0.0
		OT [38]	50.5±15.72	0.65±0.06	35.0±48.94	14.0±5.03	0.96±0.22	0.0±0.0
		FAQ [49]	52.5±13.33	0.65±0.06	0.0±0.0	9.0±10.21	0.91±0.12	0.0±0.0
		MPOpt [18]	**60.0±0.0**	**0.6 ±0.02**	95.0±22.36	73.0±13.8	0.66±0.03	90.0±30.78
		Gurobi [12]	**60.0±0.0**	**0.6 ±0.02**	**100.0±0.0**	**77.0±7.33**	**0.65±0.02**	**100.0±0.0**
		Ours	**60.0±0.0**	**0.6 ±0.02**	**100.0±0.0**	**77.0±7.33**	**0.65±0.02**	**100.0±0.0**

Table 7. **Vision-language alignment on CIFAR-10 and CINIC-10.** Our QAP solver achieves predominantly favourable matching accuracy, cost and optimality guarantees – even in comparison to proprietary solvers (Gurobi). This holds across two datasets and three pre-trained models: CLIP, DeiT and DINOv2.