Assignment 2. Transfer Learning for CNNs, Visual Prompting, Graph Neural Networks

University of Amsterdam – Deep Learning Course 1

November 16, 2023

The deadline for this assignment is December 2nd, 2023.

1 Transfer Learning for CNNs **(Total: 20 points)**

Although standard Multilayer Perceptrons (MLP) achieve good results in some tasks as seen in the previous assignment, they are not as scalable as eventually needed in practice. The disadvantage is that the number of total parameters can grow very quickly, especially when having high dimensional inputs like images. This is inefficient because there is redundancy in such high dimensions. To address the issue, Convolutional Neural Networks (CNNs) have been proposed. Using a shared set of weights at each layer, CNNs are more efficient than MLPs. The intrinsic feature of images, which is translation invariance, can be employed to boost the performance of downstream tasks.

After the proposal of CNNs, different CNN-based architectures have been introduced, which usually differ in depth, number of filters, ways of applying filters to the input information flow, etc. Some of the most famous architectures are VGG-11 [1], ResNet-32 [2], and DenseNet-121 [3]. The latter two we have already implemented and trained in the 4th week's tutorial. Although state-of-the-art (SOTA) CNN-based methods have even achieved "better than human" performance during the last few years, the key questions of transferability, generalization, and robustness of the learned features are still a matter of debate and are considered to be lagging behind.

In this part, you will first compare different architectures in terms of their accuracy, inference speed and number of parameters to get a feel of different architectures and their properties. Next, you will use these ImageNet [4] pretrained models to adapt a network to a different dataset, CIFAR-100. This resembles how you would tackle a new problem that you would encounter: especially if you do not have enough data, you would start with fairly generic, pre-trained features and adapt the model.

Question 1.1 (8 points)

Create and submit a notebook file. Specify what GPU you have used. You can also use Google Colab. We prepared a starter code for you, feel free to use it. Make sure you upload your answers to ANS.

(a) (4 points) Plot for VGG11, VGG11 with batch normalization, ResNet18, DenseNet121, MobileNet-v3-Small, ViT-B/32 and ViT-S/8 the Top-1 accuracy on ImageNet vs the inference speed at *batch_ size=8*. The value for the Top-1 accuracy of each model can be found on the PyTorch website. Also plot the inference speed vs the number of parameters. Does it scale proportionately? Make sure to set the model on evaluation mode, use `torch.no_grad()` and a GPU. Report the inference speed in ms for one image. Average the inference speed across multiple forward passes. Describe the trends you observe in max. 80 words.

(b) (2 points) Do you expect the inference speed to increase or decrease without `torch.no_grad()`? Why? What does `torch.no_grad()` do? Provide an answer in max. 50 words. For the same models as in (a), plot the inference speed with and without `torch.no_grad()`, similarly with *batch_ size=8*.

(c) (2 points) For the same models as in (a), plot the amount of GPU vRAM (you can check this with code by executing `torch.cuda.memory_allocated()` or with the terminal using `nvidia-smi`) while conducting a forward pass with `torch.no_grad()` and without. Does `torch.no_grad()` influence the memory usage? Why? Provide an answer in max. 50 words. Make sure to save the output after the forward pass. Use `batch_size=64` and report the memory in MB.

Hint: You can create a fake image with `torch.rand()`.

Question 1.2 (12 points)

Fill the missing parts in `train.py` and `cifar100_utils.py`. Submit both files, and upload your answers to ANS.

(a) (5 points) Adapt a ResNet-18 model that was pre-trained on ImageNet to CIFAR-100, via randomly resetting its last layer's parameters and **retraining only that layer**. When resetting, set the bias term to zero and the linear layer's weight term to be normally distributed around zero with $\sigma = 0.01$. Report the obtained accuracy. You should get around 58% on test set.

(b) (4 points) Try to increase the model's performance by using any augmentation method and report the accuracy. Why might augmentations help despite the majority of the model being fixed? (limit: 50 words)

(c) (3 points) What kind of information do the first and last convolutional layers capture? If you had to fine-tune one of them (along with the classifier layer), which would you pick? Why? (limit: 80 words)

2 Visual Prompting **(Total: 50 points)**

2.1 A brief introduction to CLIP

CLIP [5] (**C**ontrastive **L**anguage-**I**mage **P**re-training) is a multi-modal deep neural network based on the Transformer [6] architecture. It comprises a text and an image encoder, which project their corresponding input to a dense high-dimensional vector. These vectors are commonly referred to as **text/image embeddings**, and they are a numerical representation that can be used by other components in the pipeline to perform a downstream task (i.e. classification).

Given a pair ($\mathbf{img}_i$, $\mathbf{txt}_i$) of an image and its caption, CLIP processes each modality with the corresponding encoder, producing an embedding for each. We can formulate this as $\mathcal{I}(\mathbf{img}_i) = \mathbf{I}_i \in \mathbb{R}^d$ and $\mathcal{T}(\mathbf{txt}_i) = \mathbf{T}_i \in \mathbb{R}^d$, where $\mathcal{I}$ and $\mathcal{T}$ are the image and text encoders respectively, and d is the dimensionality of the embeddings.

What makes CLIP special is its pre-training regime. If we batch multiple image-caption pairs together, we can train the model in such a way that its two encoders produce:

- similar[1] embeddings for images and their corresponding captions, and
- dissimilar embeddings for images and the captions from other samples.

Formally put, CLIP's *contrastive* loss objective ensures that $\mathbf{I}_i \cdot \mathbf{T}_i \gg \mathbf{I}_i \cdot \mathbf{T}_j, \forall j \neq i$, i.e. the dot-product of the corresponding captions is higher than those of non-corresponding pairs.

2.2 Prompting CLIP for image classification

Prompting has been recently popularized in the field of Natural Language Processing as a method of adapting (large) language models to new downstream tasks. The core idea is to perform a transformation in the *data space* rather than the *model space* in order to achieve the sought objective, which is different than the one that the model was originally trained on. This differs to common transfer learning approaches as prompting modifies the existing model's input rather than some neural weights.

It is important to note that in this entire process, the parameters of the model remain **frozen**, i.e. they do not get updated with back-propagation during training. This is desirable as it is quite efficient to narrow down the number of trainable parameters when adapting a model to a new task. Hence, prompting is an appealing approach that allows anyone with access to these models – but not necessarily the resources to train them (recall the difference between using gradients vs not in the first part of this assignment) – to tackle problems better than a smaller, more specialized model would.

Contemporary work [7] has transferred this paradigm to the domain of Computer Vision, where they attempt to prompt the Vision Transformer [8] in CLIP (among other vision models) to perform a task that it has never been trained on (this is called "zero-shot"). The results show that visual prompting is particularly effective for CLIP, as it manages to perform on par with other standard adaptation techniques such as linear probes. Additionally, once a visual prompt has been learnt, it can be applied to images that come from a different distribution (such as a dataset with different properties) without as big of an impact on performance, compared to other common approaches. You can read the original paper at `https://arxiv.org/pdf/2203.17274.pdf` for more details.

Figure 1 provides a simple visualization of the proposed method. In this example, we want to perform image classification on a picture of a dog. To do this, we apply a *learnable* visual prompt in the form of a pixel perturbation on top of the image, and enumerate all

[1] Recall that similarity in the vector space can be defined, for instance, as either the dot product or the cosine similarity of two elements. CLIP uses the former during its training.

possible captions (corresponding to all possible classes for our task). If we imagine that this is part of the CIFAR-10 dataset, which has ten discrete classes, then we would end up with 10 image-caption pairs, one of which would be the correct “This is a photo of a dog”. Notice that in these captions, anything other the actual class name is considered a fixed **text template**, which can also be (manually) tuned for the task at hand.

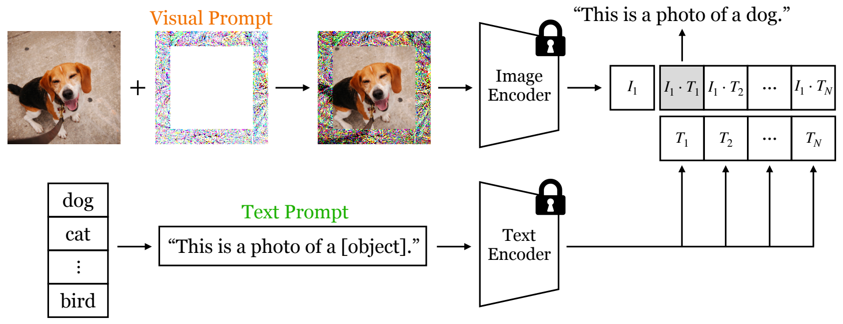


Figure 1. Prompting CLIP by transforming the input of the downstream dataset (images to classify) to match the format of the pre-training task (image-caption pairs). Figure from original paper [7].

Question 2.1 (15 points)

In order to establish a baseline, you should evaluate the **zero-shot** abilities of CLIP in image classification. This means that you won’t be learning any visual prompts yet, but rather provide the original picture as input to the image encoder, along with a fixed text template for all classes in the dataset to the text encoder.

(a) (10 points) Calculate and report the **top-1** classification accuracy of CLIP-B/32 (which is CLIP using a ViT-B/32 backbone) for the train and test sets of CIFAR-10 by filling the missing code snippets in the file `clipzs.py`. Repeat the same for CIFAR-100, and compare its **top-1** accuracy on both sets with CIFAR-10.

(b) (5 points) Try prompting CLIP to tell you whether an image:

1. is mostly *red*, *green*, or *blue* (primary color identification),
2. displays an object that is *human-made*, or one that comes from *nature*.

To achieve this, think of a suitable **text template** (`--prompt`) and the **set of labels** (`--class_names`) to use in either case, as you are no longer interested in predicting the *true class* of the object in the image. In each scenario, report the model’s predictions for eight random images from CIFAR-100’s test set by using the argument `--visualize_predictions`. How does CLIP perform on these tasks? When could this approach be useful?

*Hint: As the captions are the same for all images, and correspond to the number of classes in your task, you can calculate and store the respective text embeddings to a list **once** before iterating through the images.*

2.3 Learning a visual prompt

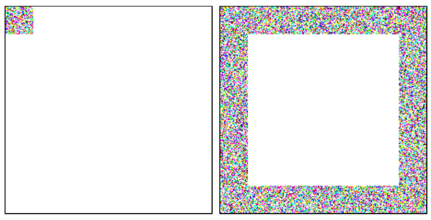
Once we evaluate how CLIP performs out-of-the-box, we can attempt to improve its performance by *learning* a visual prompt to apply on top of the dataset’s images. This allows us to essentially steer the model to perform better at the task we are interested in. One advantage of prompting in the vision domain is that the input space (pixel values) is inherently continuous, while in language it is discrete (word tokens). Thus, we can back-propagate the loss to the input and update our prompt according to the gradients, in order to increase the effectiveness of the model.

The area of the image where we apply the visual prompt, as well as its size, naturally determine its impact. However, both parameters depend entirely on the inner workings of the model – there have been popular examples in the field of adversarial attacks[2] where as little as a *single pixel* [9] can lead to a misclassification by the model. Hence, it is not trivial to determine the ideal properties for a prompt, as something that has an influence on the human perception may not be as effective for a neural model, and vice versa.

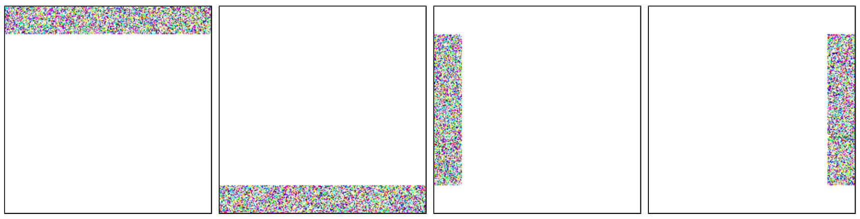
On the other hand, if we do not reduce the search space to a specific configuration of locations or prompt sizes, then it becomes computationally prohibitive to optimize for it. As a middle ground, the original paper proposed three types of visual prompts:

1. a **single** pixel patch (1x1 px) at a *fixed* location (Fig. 2(a)),
2. a **padding** (or frame) of size p=30 pixels at the *inner edges* of the image (*i.e.*, Fig. 2(c), which is built using parts (d-g)).

See Figure 2 for a visualization of these prompts, along with other design ideas.



(a) Fixed **(b)** Padding



(c) Pad up **(d)** Pad down **(e)** Pad left **(f)** Pad right

Figure 2. Example visualization of prompts of size 30.

Question 2.2 (10 points)

Learn a visual prompt for the two types mentioned above by filling the missing code snippets in the files `learner.py`, `vpt_model.py`, and `vp.py`. Compare the effectiveness of each prompt type on both CIFAR-10 and CIFAR-100, while keeping in mind the baselines from your zero-shot experiments. Which type of visual prompts leads to the biggest increase in top-1 classification accuracy for each dataset?

2.4 Learning deep prompts

So far, the visual prompts were applied to the input of the model in the image space. An alternative approach is to depart from the image space and learn so-called *deep* prompts that are added as tokens to an intermediate representation within some layer of the model. In other words, instead of adding the prompts to an image, they will be appended to

[2]You can read more about adversarial attacks in the 10th tutorial of the Deep Learning 1 notebooks.

the list of original tokens determined based on the input. In the case of CLIP, these deep tokens can be injected before any one of the Transformer blocks. See Figure 3 for a visualization of the concept of deep prompts for Transformer-based vision models.

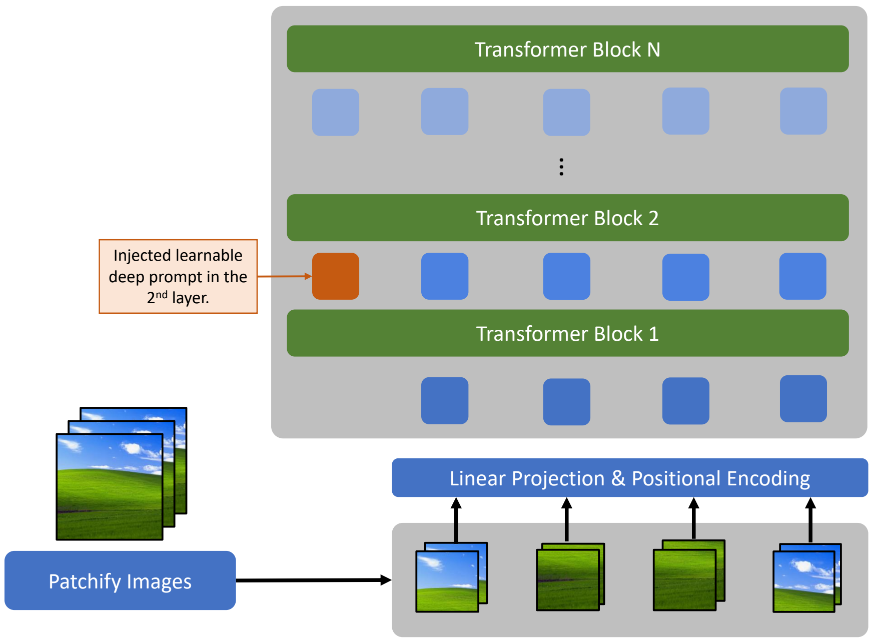

Transformer Block N
Transformer Block 2
Injected learnable deep prompt in the 2nd layer.
Transformer Block 1
Linear Projection & Positional Encoding
Patchify Images

Figure 3. Deep Prompt Learning with Vision Transformers

Question 2.3 (10 points)

Similar to the previous exercise on visual prompts, implement the missing code snippets in `dpt_model.py` and report the top-1 accuracy. Compare your results from using deep prompts with the visual prompts from the previous question. What are the effects of injecting the deep prompts at different blocks of the Transformer and how can these performance differences be explained?

Question 2.4 (5 points)

In order to evaluate the robustness of your learnt prompts against distributional shifts, you can try adding noise to the test set of each dataset and observe whether there is a significant drop in performance.

Fill the missing code snippets in the file `dataset.py` so that specifying the launch argument `--test_noise` adds noise to the test set's images from the following distribution: $\mathcal{N}(0, 0.1)$. Compare how CLIP handles the noisy test set versus the fine-tuned ResNet you trained in Section 1. What do you observe?

Hint: You can use the `train.py` *to run experiments for ResNet and for the CLIP model you can use the* `robustness.py`*.*

Question 2.5 (6 points)

Finally, you can evaluate the effectiveness of your learnt visual prompts on the *combination* of CIFAR-10 and CIFAR-100. More specifically, you should compare CLIP's performance when predicting a class out of both datasets' labels (i.e. perform a 110-way classification, as the two sets of classes are mutually exclusive) with its performance on each dataset individually from the previous questions.

Fill the missing code snippets in the file `cross_dataset.py`. Use the prompts that you learnt from CIFAR-10 and report the top-1 classification accuracy on the concatenation of both test sets. Repeat the same using the prompts that you learnt from CIFAR-100. Which set of prompts seems to work best when combining the two datasets? Is this what you expected, and what could be the reason for it?

Hint: When combining the target labels of the two datasets, you should offset the labels for one of the two, as CIFAR-10's range from 0 to 9 and CIFAR-100's from 0 to 99. Ultimately, your new target labels should range from 0 to 109.

2.5 Comparing transfer learning with prompting

After experimenting with both the standard methods in transfer learning (Section 1) and prompting, you should be able to draw some conclusions regarding the advantages, disadvantages, and possible applications for each approach.

Question 2.6 (4 points)

Compare prompting to linear probing and full fine-tuning in terms of:

1. robustness against distributional shifts,
2. computational resources required,
3. memory requirements, and
4. overall performance on the downstream task.

Think of the advantages and/or disadvantages of prompting over linear probing and full fine-tuning for each item, and answer the multiple choice questions on ANS.

3 Graph Neural Networks **(Total: 30 points)**

A graph $G = (V, E)$ can represent a set of objects and their inter-object relations. The objects are abstracted to nodes V and the connections between them to edges E between the nodes. Describing objects and their relations makes graphs very general structures. Hence, they are universally relevant in science: In the humanities and social sciences, they may appear, for example in social, or citation networks, in the natural sciences, for example in physical interactions, biological networks, or molecular structures in chemistry. Graph Neural Networks (GNNs) are a particular class of artificial neural networks designed to process such graph-structured data.

3.1 Adjacency matrices

An adjacency matrix of an *undirected* graph represents the graph structure by indicating the edges between nodes:

$$A_{uv} = \begin{cases} 1 & \text{if is there is an edge between node } u \text{ and node } v, \\ 0 & \text{else.} \end{cases} \tag{1}$$

Note that a connection from a node to itself is not considered an edge here, $A_{uu} = 0$. To account for self-connections, you would need to add the identity matrix: $A + I$.

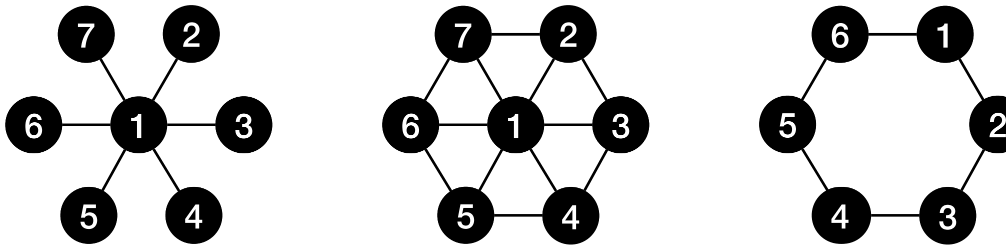


Figure 4. Three simple graph structures.

Question 3.1 (4 points)

- For the sub-questions **a** and **b** provide your answers as a matrix in the notation of a 2D python list. For example: [[1,2,3], [4,5,6], [7,8,9]]

(a) **(2 points)** Write down adjacency matrices for the graphs shown in Fig. 4. Please respect the indicated node ordering.

(b) **(1 point)** For the third graph (Fig. 4, right), also write down the square of the adjacency matrix A^2.

(c) **(1 point)** How can you interpret the entries of the squared adjacency matrix A^2? How can you interpret the entries of the adjacency matrix to the power of n, $(A^n)_{uv}$?

Hint: Being concise and to the point will help you get all the points.

3.2 Graph convolution

For an *undirected* graph with N nodes, each node v is associated with a d-dimensional embedding h_v. Let us consider the following graph convolution that propagates the embeddings for all nodes from layer l to the next layer $l + 1$:

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(l)}}{|\mathcal{N}(v)|} + B^{(l)} h_v^{(l)}\right). \tag{2}$$

The nonlinearity, e.g., ReLU, is denoted by σ. The matrices $B^{(l)}, W^{(l)} \in \mathbb{R}^{d^{(l+1)} \times d^{(l)}}$ are parameterizing the self-connection and the combination of neighboring node activations respectively. The neighborhood of a node v is denoted by $\mathcal{N}(v)$ and consists of all nodes connected to v with an edge. Hence, $|\mathcal{N}(v)|$ stands for the *number* of neighboring nodes.

Question 3.2 (4 points)

(a) (1 point) Where in the definition of graph convolution (Eq. 2) does the structural information on the graph appear?

(b) (3 points) In message-passing neural networks (Gilmer et al. 2017 [10]), after every layer, the information contained in the embeddings of a node is propagated to the neighbor nodes in so-called *messages*. These are summed up to update the node embedding:

$$\begin{aligned} m_v^{l+1} &= \sum_{u \in \mathcal{N}(v)} \text{message}(h_v^l, h_u^l, e_{uv}), \\ h_v^{l+1} &= \text{update}(h_v^l, m_v^{l+1}). \end{aligned} \tag{3}$$

Here, 'message' and 'update' are nonlinear functions that can be parametrized by artificial neural networks.

How is the graph convolution defined above (Eq. 2) related to message-passing neural networks, Eqs. 3?

Hint: Being concise and to the point will help you get all the points.

The degree matrix D of a graph with N nodes is an $N \times N$ diagonal matrix. For each node v, it counts the nodes in its neighborhood $\mathcal{N}(v)$ (nodes connected to node v with an edge):

$$D_{vu} := \begin{cases} |\mathcal{N}(v)| & \text{if } v = u, \\ 0 & \text{else.} \end{cases} \tag{4}$$

Question 3.3 (12 points)

(a) (8 points) Rewrite the graph convolution (Eq. 2) to matrix form $H^{(l+1)} = f(H^{(l)})$ with embedding matrix $H^{(l)} = [h_1^{(l)}, \dots, h_{|V|^{(l)}}]^T$. Use the adjacency matrix A (Eq. 1) and the degree matrix D (Eq. 4).

(b) (2 points) Which function does this graph convolutional network (Eq. 2) use to aggregate over neighboring nodes?

(c) (2 points) Is this (as in **(a)**) rewriting of the update formula from index to matrix form possible for all aggregation functions? If not, give an example of an aggregation function that does not work and explain why.

Hint: Being concise and to the point will help you get all the points.

3.3 Connections to standard neural network architectures

Question 3.4 (6 points)

(6 points) In principle, we can concatenate the adjacency matrix and the node embeddings to classify each data point with a standard feed-forward neural network. What problems do you expect with this approach? Where does it fail completely? Name three different aspects.

Hint: Keep in mind different applications: for example, classifying molecular structures in chemistry (i.e., graph classification) or categorizing people in a social media network graph (i.e., node classification).

Hint: Being concise and to the point will help you get all the points.

Question 3.5 (4 points)

(a) **(2 points)** Consider a standard transformer model applied to a sentence in natural language. How can a transformer model be seen as a graph neural network? What would be the nodes and edges, and what would be the graph structure?

(b) **(2 points)** Transformers perform state-of-the-art in the natural language domain, which is inherently not permutation invariant: it matters in which order we say something. How can the transformer model process inputs for which the order matters while staying permutation invariant in its architecture and computation?

Hint: Being concise and to the point will help you get all the points.

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1

[4] Li Fei-Fei, Jia Deng, and Kai Li. Imagenet: Constructing a large-scale image database. *Journal of vision*, 9(8):1037–1037, 2009. 1

[5] 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. PMLR, 2021. 3

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 3

[7] Hyojin Bahng, Ali Jahanian, Swami Sankaranarayanan, and Phillip Isola. Exploring visual prompts for adapting large-scale models. *arXiv preprint arXiv:2203.17274*, 2022. 3, 4

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021. 3

[9] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019. 5

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. 2017. 9