Advanced topics

M1 ARIA
Image and Video Processing

Gabriele Facciolo

Lecture 4 - 27-09-2024

Contents

- Transfer learning
  - Multi-task learning
  - Representation learning
- Attention and transformers
  - Text translation
  - Attention!
  - Vision transformers
- Text-image models
  - CLIP
- Image denoising
  - Generative models ?

"He'll have to call you back. He's in the middle of a shallow dive."

Transfer learning

Review: supervised training

We have a set of data points with corresponding **labels**

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m), \quad \text{with } x_i \in \mathcal{X}, y_i \in \mathcal{Y}, \quad (\text{e.g. } \mathcal{X} = \mathbb{R}^d, \mathcal{Y} = \mathbb{R}^t)$$

We assume that the data pairs are IID samples of a joint PDF: $(x_i, y_i) \sim p(x, y)$.

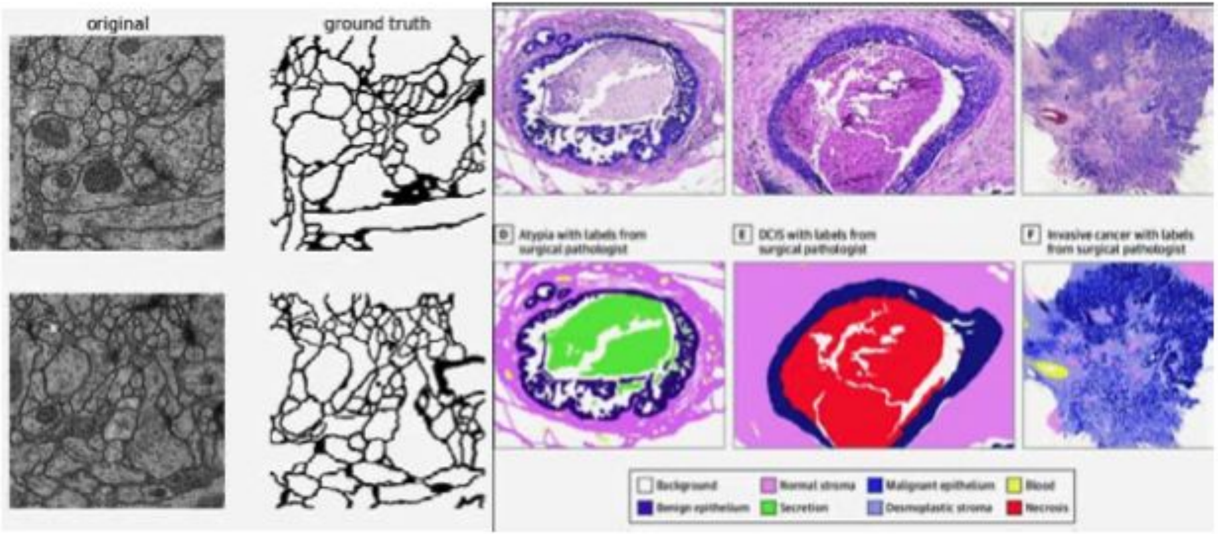
We want a function $\mathcal{F}_\theta : \mathcal{X} \to \mathcal{Y}$ such that $\mathcal{F}_\theta(x)$ "$=$" y, for $(x, y) \sim p(x, y)$.

To that end we define a **loss function** ℓ which measures the error between $\mathcal{F}_\theta(x)$ and y, and set the **parameters** θ to minimize the expected loss:

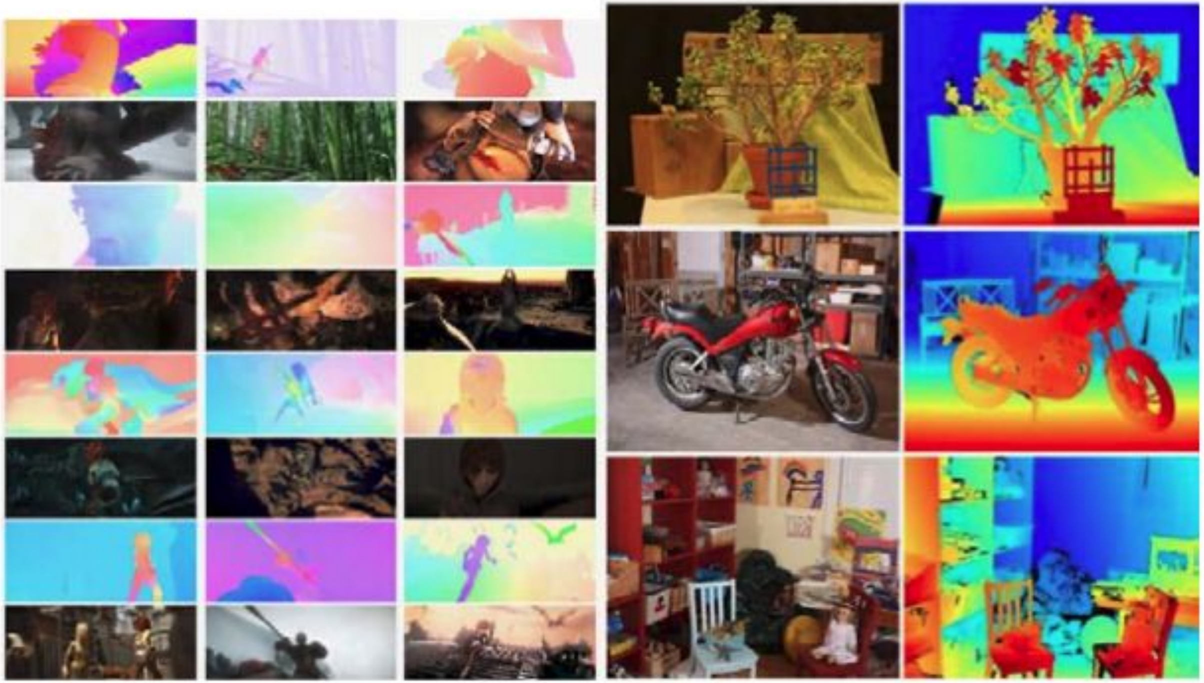
$$\underbrace{\mathcal{R}^{\text{emp}}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell(\mathcal{F}_\theta(x_i), y_i)}_{\text{empirical risk}} \to_{m \to \infty} \underbrace{\mathbb{E}\{\ell(\mathcal{F}_\theta(x), y)\} = \mathcal{R}(\theta)}_{\text{risk}}.$$

Supervised training needs labels

Supervised training needs labels



Supervised training needs labels



Labelling is expensive (or impossible)

In the best case label is expensive: requires several hours of effort.

- classification
- segmentation

Sometimes, it requires expert knowledge:

- medical imaging
- satellite imagery

For some problems the ground truth is not know, or very difficult to measure:

- motion estimation
- depth estimation
- image restoration

In practice: small labeled dataset

We need strategies to cope with small datasets. We have see already:

- data augmentation: synthetically generate new data by applying transforms to existing data
- regularization: prevent overfitting to the dataset

These help, but are insufficient if dataset is very small.

Transfer learning

A function (e.g. a network) $\mathcal{F}_S : \mathcal{X}_S \to \mathcal{Y}_S$ has been trained to trained to solve a **source problem**:

$$S = (\mathcal{X}_S, \mathcal{Y}_S, p_S(x, y) = p_S(x) p_S(y|x), \ell_S).$$

Can it be used to help training a **second target problem**?

$$T = (\mathcal{X}_T, \mathcal{Y}_T, p_T(x, y) = p_T(x) p_T(y|x), \ell_T).$$

Transfer learning

Most usual cases:

(Inductive) transfer learning: Input spaces are the same, but task changes:

$$\begin{cases} (\mathcal{X}_S, p_S(x)) = (\mathcal{X}_T, p_T(x)), \\ \mathcal{Y}_T \neq \mathcal{Y}_S \text{ or } p_S(y|x) \neq p_T(y|x) \text{ or } \ell_T \neq \ell_S. \end{cases}$$

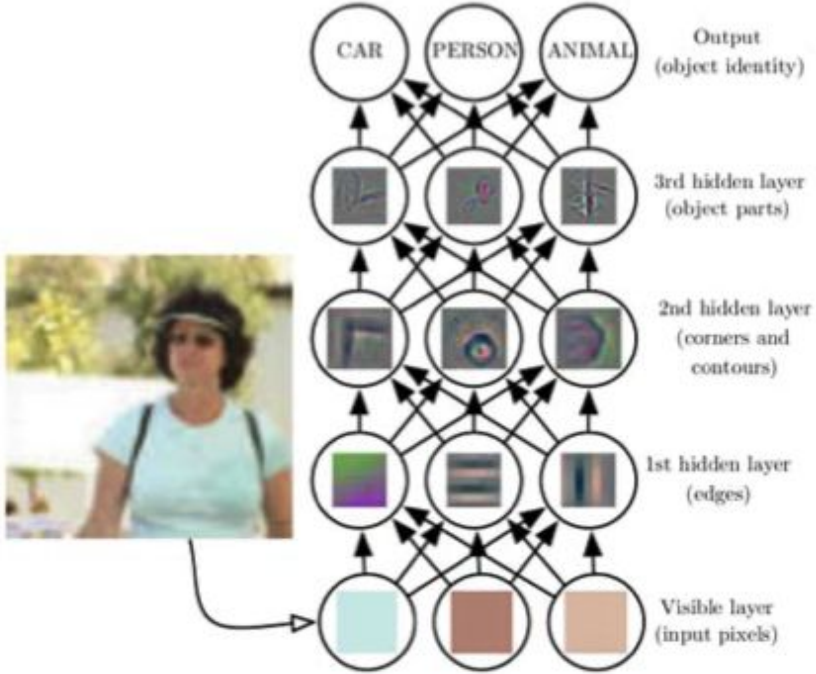
Example: detect objects in natural images → segmentation of natural images

Domain adaptation: Input spaces are different, but task is the same:

$$\begin{cases} \mathcal{X}_S \neq \mathcal{X}_T \text{ or } p_S(x)) \neq p_T(x), \\ \mathcal{Y}_T = \mathcal{Y}_S \text{ and } \ell_T = \ell_S. \end{cases}$$

Example: sentiment classification in hotel reviews → sentiment classification in reviews of technological items

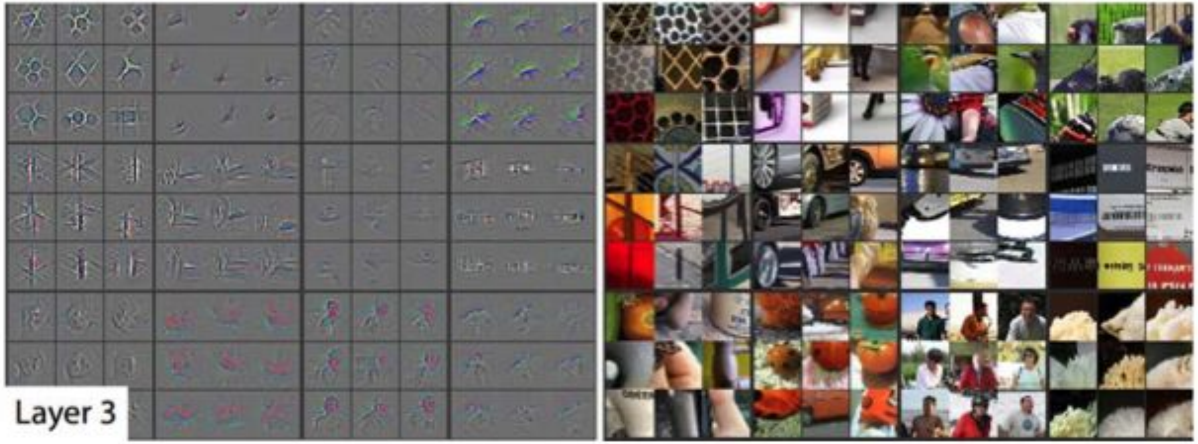
Intuition: why is transfer learning possible



Weights in first layers are low-level features (related to input domain). In deeper layers more complex higher level concepts begin to appear (related to task).

Figure from [Deep Learning, Goodfellow, Bengio and Courville, 2016]

Intuition: why is transfer learning possible



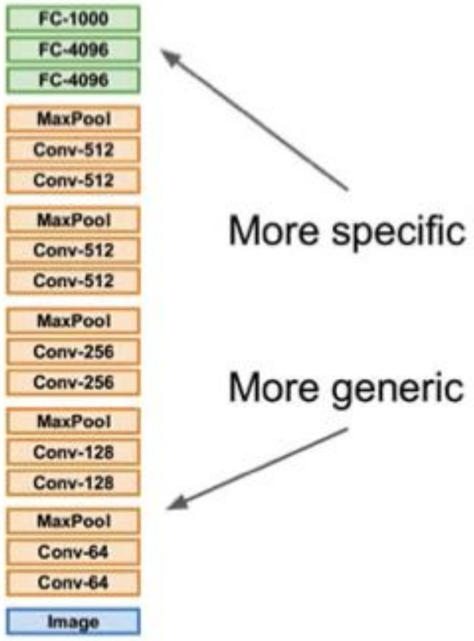
Figures from [Visualizing and Understanding Convolutional Networks. Zeiler, Fergus, 2013]

Intuition: why is transfer learning possible

https://www.youtube.com/watch?v=AgkfIQ4IGaM

Demo video from one of the authors of [Understanding Neural Networks Through Deep Visualization, Yosinski et al. 2015.]

Transfer learning in practice



Early layers

- low-level features (edges, textures, corners, blobs)
- seem to be independent from the task
- networks for different tasks have similar filters

Deeper layers

- higher level concepts (faces, text, clothing)
- could be transfered between tasks requiring similar semantic concepts

Final layers (head of the network)

- computes the required output: classifier, localization, etc.

Transfer learning in practice: fine-tune from ImageNet

Final layers have to be replaced with new ones and trained from scratch.

The weights of the "freezed" layers can be kept constant, or used as initialization for a **fine-tuning** with a small learning rate. The larger the training set, the more layers we can fine-tune. This is a trial-and-error process.

Transfer learning in practice

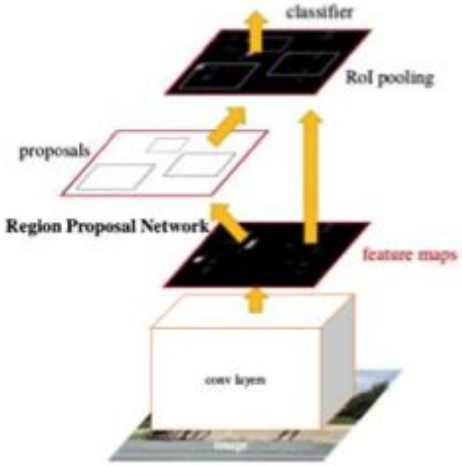
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

This and the previous 2 slides were taken from [CS231n course of Stanford University]

Transfer learning in detection and localization

. . . when labeled training data is scarce, supervised pre-training for an auxiliary task, followed by domain-specific fine-tuning, yields a significant performance boost.

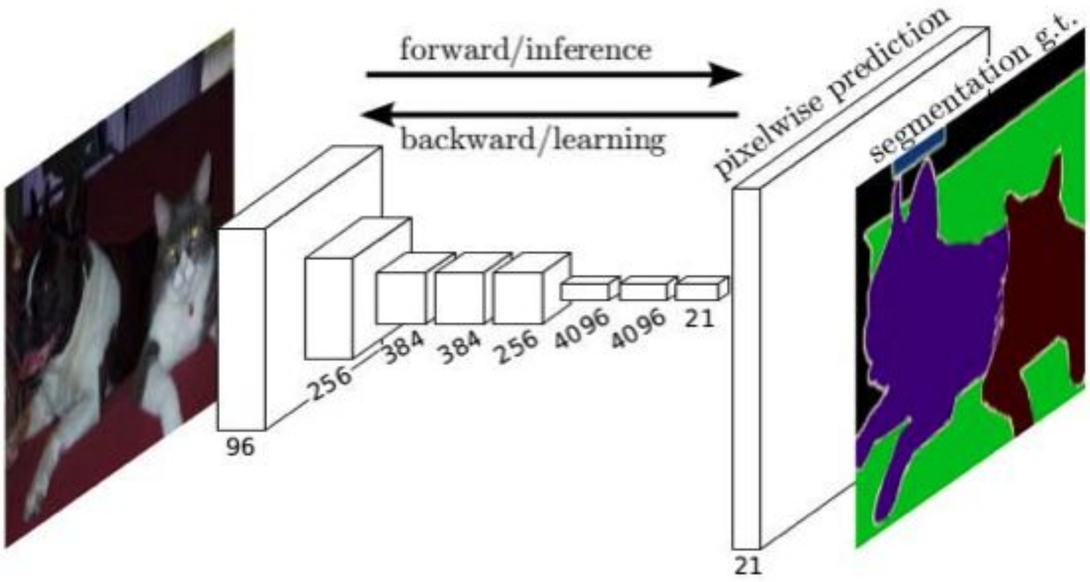
[R-CNN, Girshick et al. 2014]



All other layers (. . .) are initialized by pre-training a model for ImageNet classification [36], as is standard practice.

[Faster R-CNN, Ren et al. 2016]

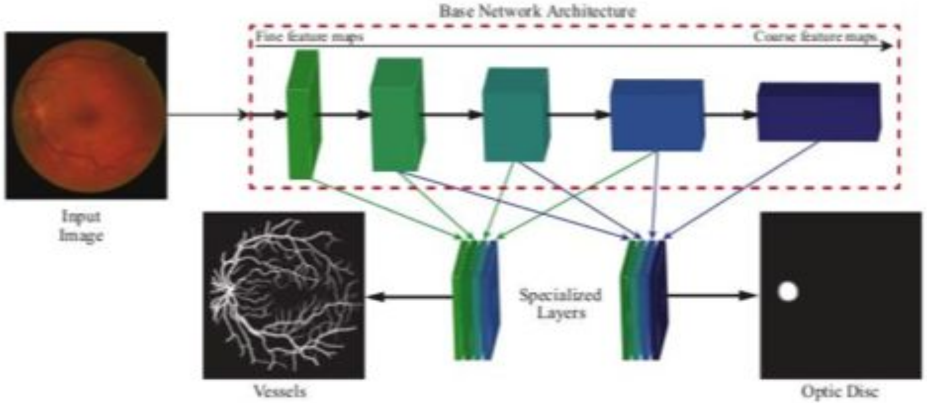
Transfer learning in semantic segmentation



We adapt contemporary classification networks (AlexNet [22], the VGG net [34], and GoogLeNet [35]) into fully convolu- tional networks and transfer their learned representations by fine-tuning [5] to the segmentation task.

[Fully Convolutional Networks for Semantic Segmentation, Long et al. 2014]

Transfer learning in medical image segmentation



We start from the VGG [18] network (. . .) the fully connected layers at the end of the network are removed.

(. . .) we fine-tune the entire architecture for 20000 iterations. Due to the lack of data, the learning rate is set to a very small number.

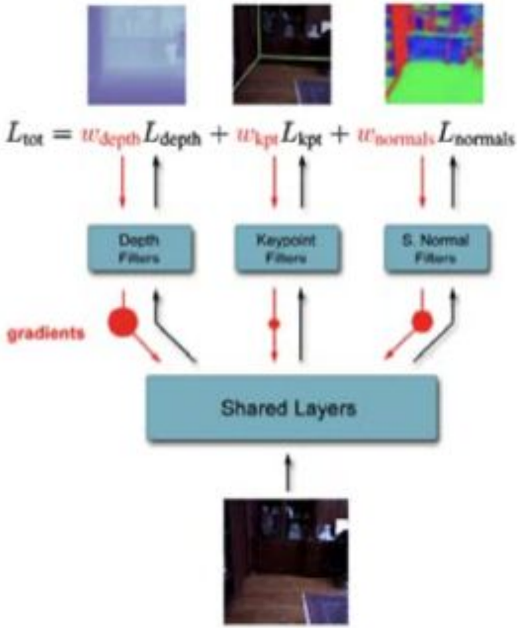
[Deep retinal image understanding, Maninis et al. 2016]

Multi-task learning

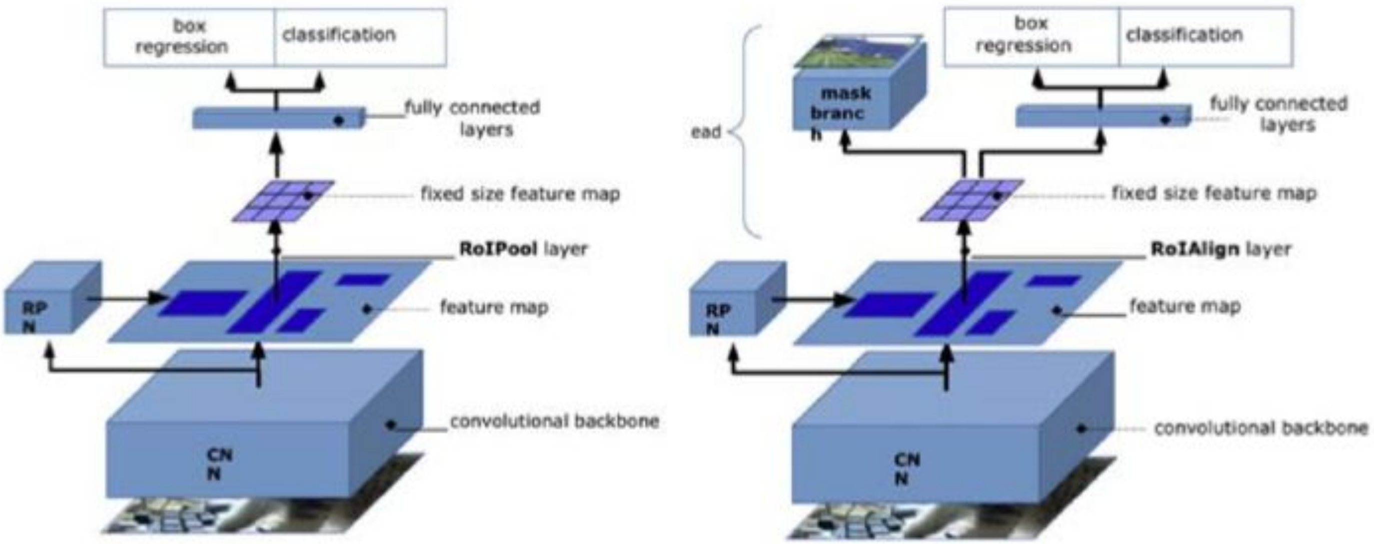
Multi-task learning

Multi-task network: A single network with shared layers and tasks specific outputs

- Multi-task loss combining individual losses
- Don't need to have all labels for all training samples
- If tasks are related, the shared weights benefit from the training samples for all tasks
- Related to transfer learning, but different: tasks are learned simultaneously, and works better if number of training samples is similar for all tasks



Multi-task learning: famous examples



Faster R-CNN [Ren et al. 2015] and Mask R-CNN [He et al. 2017] are multi-task networks. The convolutional backbone is shared for different tasks (object classification, boundix box prediction and object mask).

Figure from [Ildoo Kim]

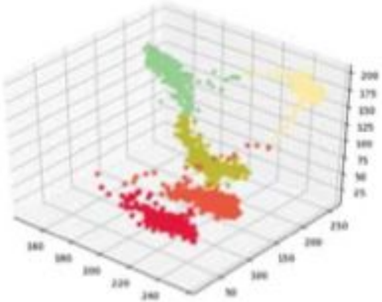
Unsupervised representation learning

Unsupervised representation learning

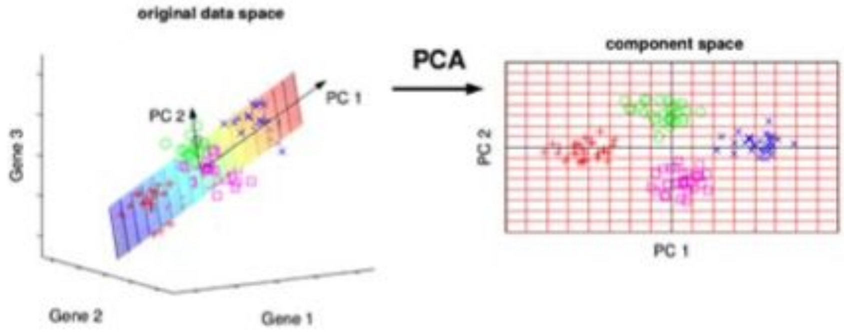
Representation learning has been typically addressed using **unsupervised training**:

- No external labels are required
- The goal of the model is not to predict an output (with notable exceptions)
- Learn structure, patterns, modes of variation from unlabeled dataset
- Since no labels are needed, a lot of data is available

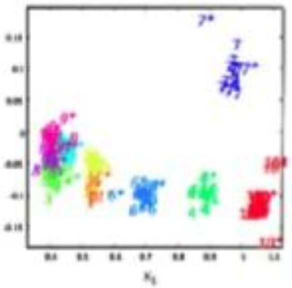
Examples of unsupervised learning

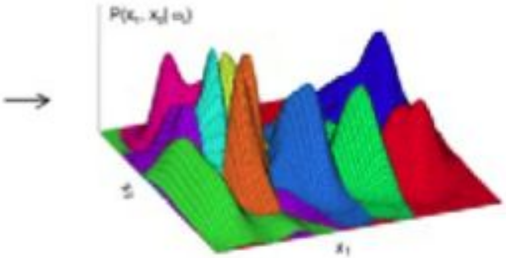


Find clusters in dataset



Dimensionality reduction (e.g. PCA)

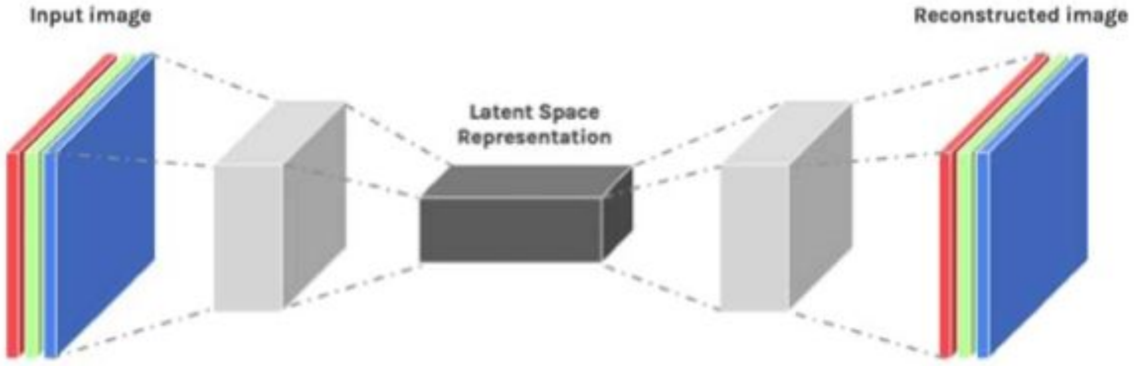




Density estimation

Figures from [Swagatam Das, 2019], [Eulertech] and [Matthias Scholz, 2006]

Example of unsupervised representation learning: autoencoders



Autoencoders have been proposed for learning a feature representation:

1. Given an image x, the encoder networks coputes the code $\varphi(x)$ (also embedding, latent representation, etc).
2. The decoder network reconstruct the image $\hat{x} = \psi(\varphi(x))$ from the code $\varphi(x)$
3. Both networks are trained end-to-end by minimizing $\|\hat{x} - x\|^2$
4. Even if the training requires a loss, this is considered often unsupervised training
5. The decoder is mainly used for training, and then it is discarded. The encoder $\varphi(x)$ can then be used for different tasks by appending a small network.

Self-supervised representation learning

Self-supervised representation learning aims at solving these issues:

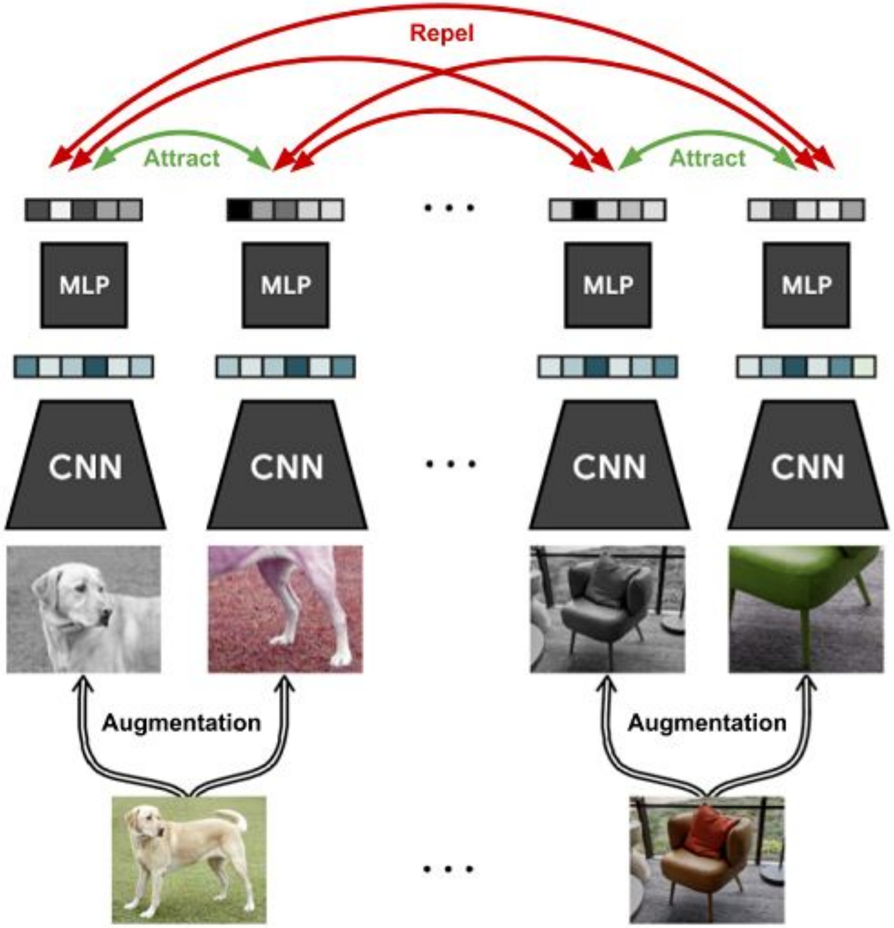
- Train a network to solve an auxiliary task for which we know the labels (the **pretext** task)
- General principle: hide some information from the input, and train the network to recover it.
- It is supervised learning, but does not require an external label, since the label is part of the data (the hidden information)!!
- The pretext task has to be related to the real task we need to solve, so that we can **transfer**.

SimCLR

Pretext task: augmentation

Same object (different image) must have a similar representation

Different objects must have different representations



“A Simple Framework for Contrastive Learning of Visual Representations” Chen et al. 2020

Many other pretext task are possible

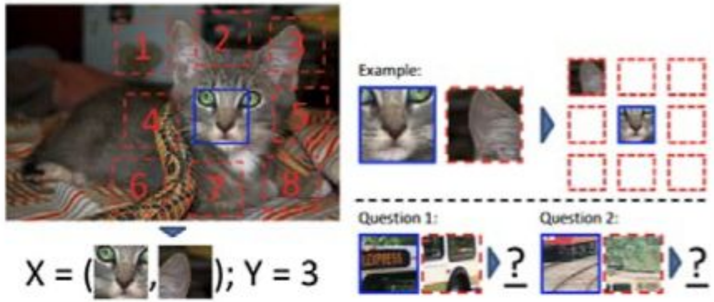


Fig. 4. Illustration of self-supervised learning by predicting the relative position of two random patches. (Image source: Doersch et al., 2015)

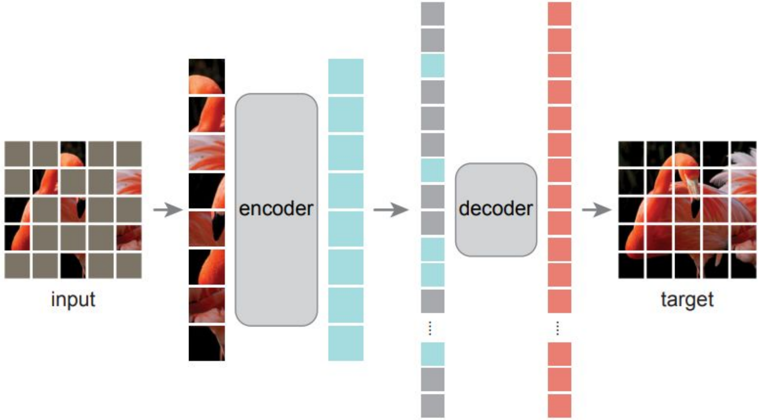


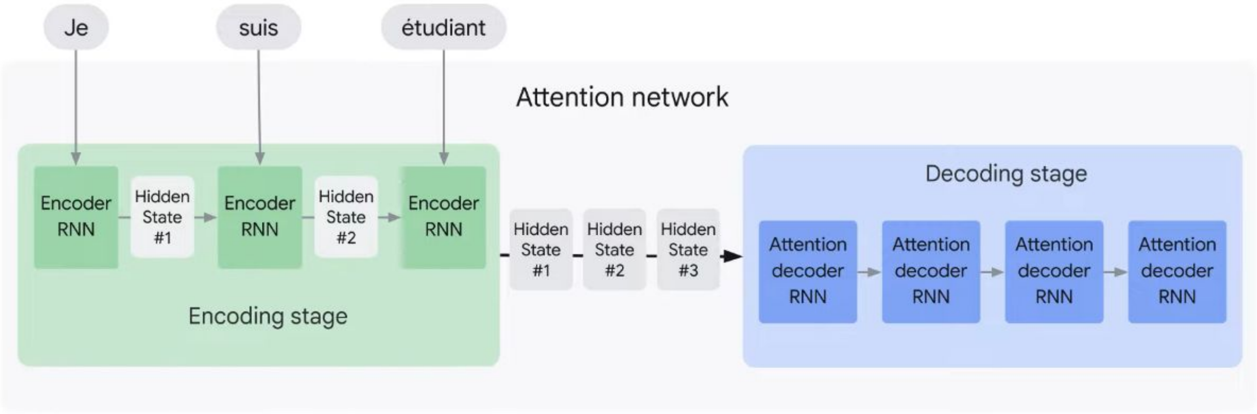
Figure 1. **Our MAE architecture**. During pre-training, a large random subset of image patches (*e.g.*, 75%) is masked out. The encoder is applied to the small subset of *visible patches*. Mask tokens are introduced *after* the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images (full sets of patches) for recognition tasks.

“Masked Autoencoders Are Scalable Vision Learners” He et al. 2021

Attention and transformers

Transformers step by step

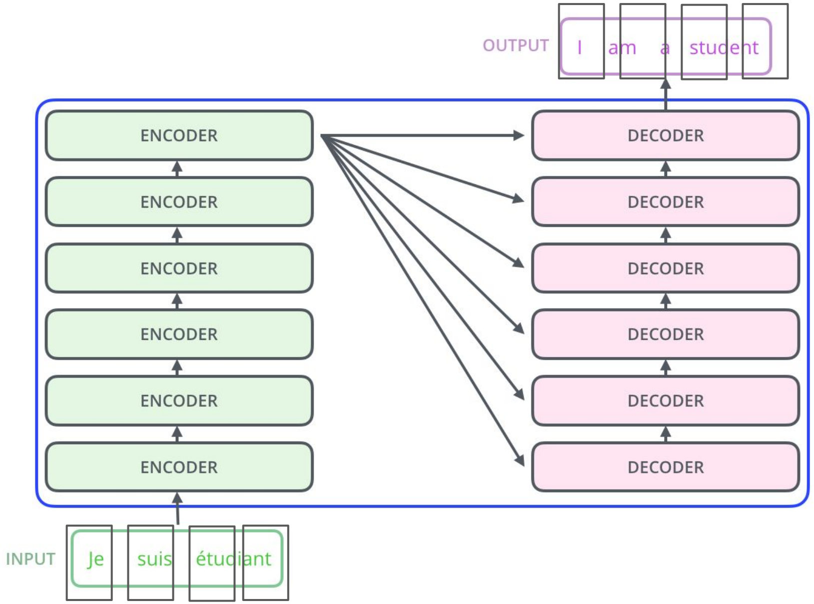
A transformer processes sequential data, like text or time series, by capturing complex dependencies and relationships between elements in the sequence using self-attention mechanisms.

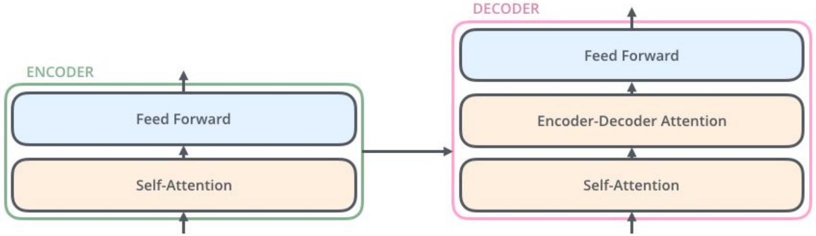


“Attention is all you need” Vaswan et al 2017.

Transformers step by step

The encoder and decoders are a stack of identical layers or blocks (each with with different weights). The encoder and decoder block share a core feature: the self-attention mechanism





Transformer block

A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \qquad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

$$\alpha_{i,j}^{(h)} = \text{softmax}_j \left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right), \tag{2}$$

$$\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j}^{(h)} V^{(h)}(\mathbf{x}_j), \qquad W_{c,h} \in \mathbb{R}^{k\times d}, \tag{3}$$

$$\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d, \tag{4}$$

$$\mathbf{z}_i' = W_2^T \text{ReLU}(W_1^T \mathbf{u}_i), \qquad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}, \tag{5}$$

$$\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2), \qquad \gamma_2, \beta_2 \in \mathbb{R}^d. \tag{6}$$

A transformer block “transforms” a collection of n objects in R^d to another collection of objects in R^d.

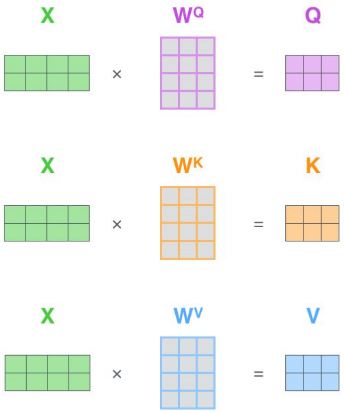
“The Transformer Model in Equations” John Thickstun, 2021

Transformer block

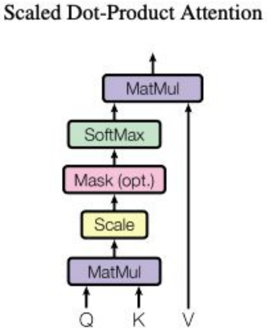
A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \quad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

$$\alpha_{i,j}^{(h)} = \text{softmax}_j \left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right), \tag{2}$$



n=2



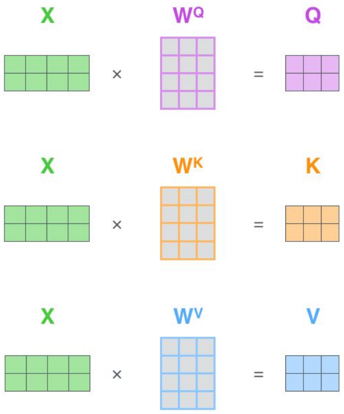
Transformer block

A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

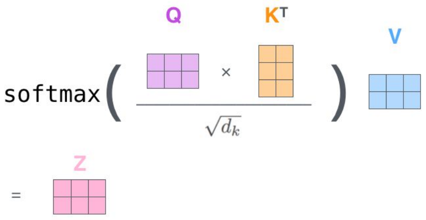
$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \qquad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

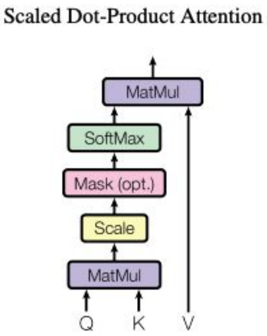
$$\alpha_{i,j}^{(h)} = \text{softmax}_j \left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j)\rangle}{\sqrt{k}} \right), \tag{2}$$

$$\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j}^{(h)} V^{(h)}(\mathbf{x}_j), \qquad W_{c,h} \in \mathbb{R}^{k\times d}, \tag{3}$$



n=2





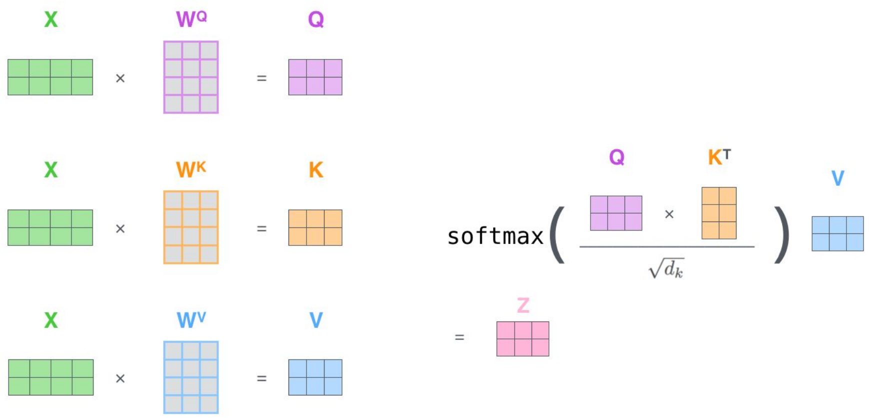
Transformer block

A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

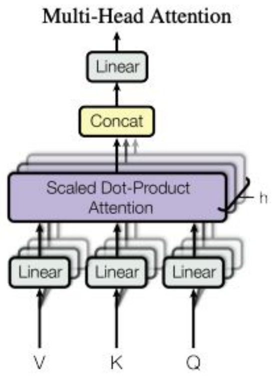
$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \qquad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

$$\alpha_{i,j}^{(h)} = \mathrm{softmax}_j \left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right), \tag{2}$$

$$\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j}^{(h)} V^{(h)}(\mathbf{x}_j), \qquad W_{c,h} \in \mathbb{R}^{k\times d}, \tag{3}$$



n=2



Transformer block

A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \qquad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

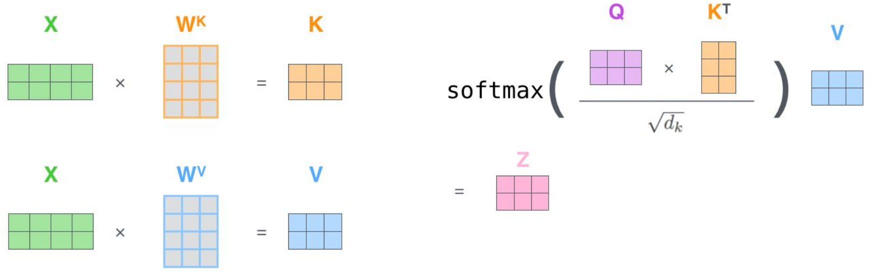
$$\alpha_{i,j}^{(h)} = \mathrm{softmax}_j \left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j) \rangle}{\sqrt{k}} \right), \tag{2}$$

$$\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j}^{(h)} V^{(h)}(\mathbf{x}_j), \qquad W_{c,h} \in \mathbb{R}^{k\times d}, \tag{3}$$

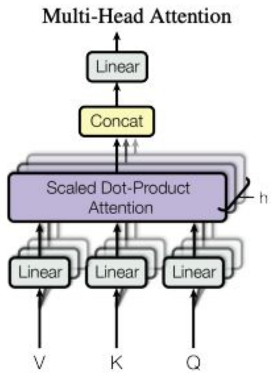
$$\mathbf{u}_i = \mathrm{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d, \tag{4}$$

$$\mathbf{z}_i' = W_2^T \mathrm{ReLU}(W_1^T \mathbf{u}_i), \qquad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}, \tag{5}$$

$$\mathbf{z}_i = \mathrm{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2), \qquad \gamma_2, \beta_2 \in \mathbb{R}^d. \tag{6}$$



n=2



Transformer block

A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \qquad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

$$\alpha_{i,j}^{(h)} = \text{softmax}_j \left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j)\rangle}{\sqrt{k}} \right), \tag{2}$$

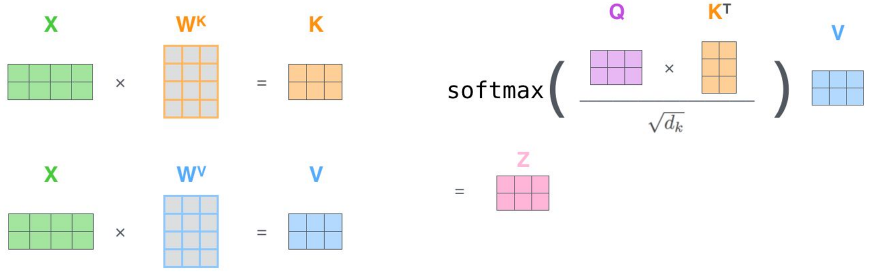
$$\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j}^{(h)} V^{(h)}(\mathbf{x}_j), \qquad W_{c,h} \in \mathbb{R}^{k\times d}, \tag{3}$$

$$\mathbf{u}_i = \text{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d, \tag{4}$$

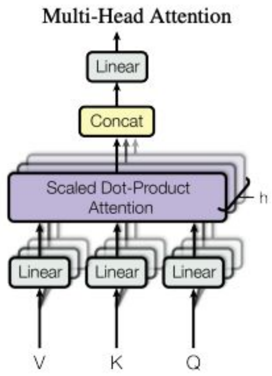
$$\mathbf{z}_i' = W_2^T \text{ReLU}(W_1^T \mathbf{u}_i), \qquad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}, \tag{5}$$

$$\mathbf{z}_i = \text{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2), \qquad \gamma_2, \beta_2 \in \mathbb{R}^d. \tag{6}$$

These linear layers expand (from d to m) then reduce (back to d) the dimension of the vectors



n=2



Transformer block

A **transformer block** is a parameterized function class $f_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. If $\mathbf{x} \in \mathbb{R}^{n\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$Q^{(h)}(\mathbf{x}_i) = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \qquad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times k}, \tag{1}$$

$$\alpha_{i,j}^{(h)} = \mathrm{softmax}_j \left(\frac{\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}(\mathbf{x}_j)\rangle}{\sqrt{k}} \right), \tag{2}$$

$$\boxed{\mathbf{u}_i' = \sum_{h=1}^{H} W_{c,h}^T \sum_{j=1}^{n} \alpha_{i,j}^{(h)} V^{(h)}(\mathbf{x}_j),} \qquad W_{c,h} \in \mathbb{R}^{k\times d}, \tag{3}$$

$$\mathbf{u}_i = \mathrm{LayerNorm}(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1), \qquad \gamma_1, \beta_1 \in \mathbb{R}^d, \tag{4}$$

$$\mathbf{z}_i' = W_2^T \mathrm{ReLU}(W_1^T \mathbf{u}_i), \qquad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}, \tag{5}$$

$$\mathbf{z}_i = \mathrm{LayerNorm}(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2), \qquad \gamma_2, \beta_2 \in \mathbb{R}^d. \tag{6}$$

The matrices W and Layernorm parameters are all learnable parameters.

If we suppose the weights α fixed, then the output of the block boils down to a stack of two linear layers.

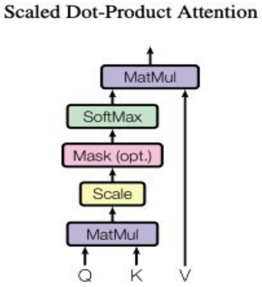
However, since the weights α change with the input, a different linear is applied to each the n inputs!

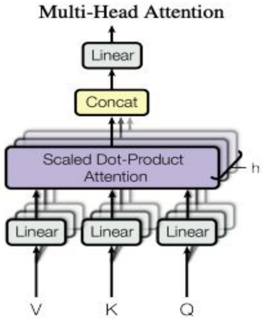
The transformer ignores any sequence structure. If this structure exists, it must be explicitly encoded in the input vectors

Classic transformer parameters

- Dimension of the input x vectors d = 512
- Dimension of the “projected” vectors k = 64
- The intermediate dimension of the linear layers is set to m = 2048
- Attention heads H = 8
- Transformer layers L=6 (in the encoder)

Training consisted in predicting the next “word” in sentences.





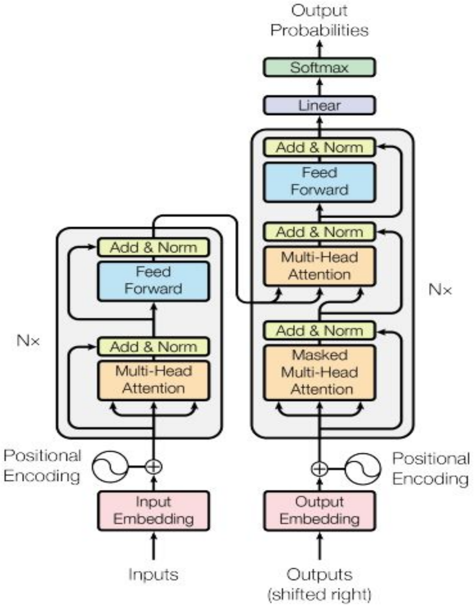


Figure 1: The Transformer - model architecture.

Illustrations from “Attention is all you need” Vaswan et al. 2017.

Positional encoding!

A transformer is fundamentally a bag of features model, operating on a collection of n unordered, d-dimensional features.

To model positions in a transformer, we need to express these positional relationships as data.

For that each vector x in the sequence is attached with a “positional code” in the form some extra dimensions.

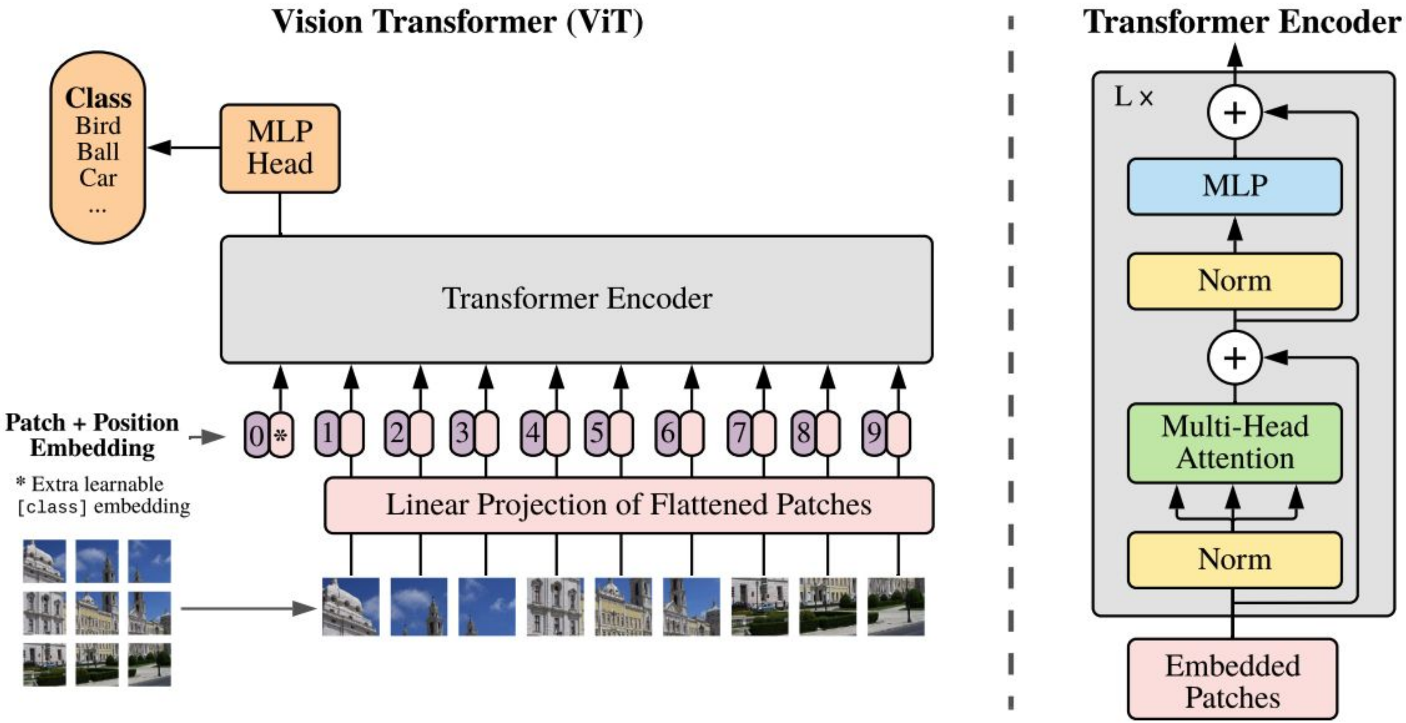
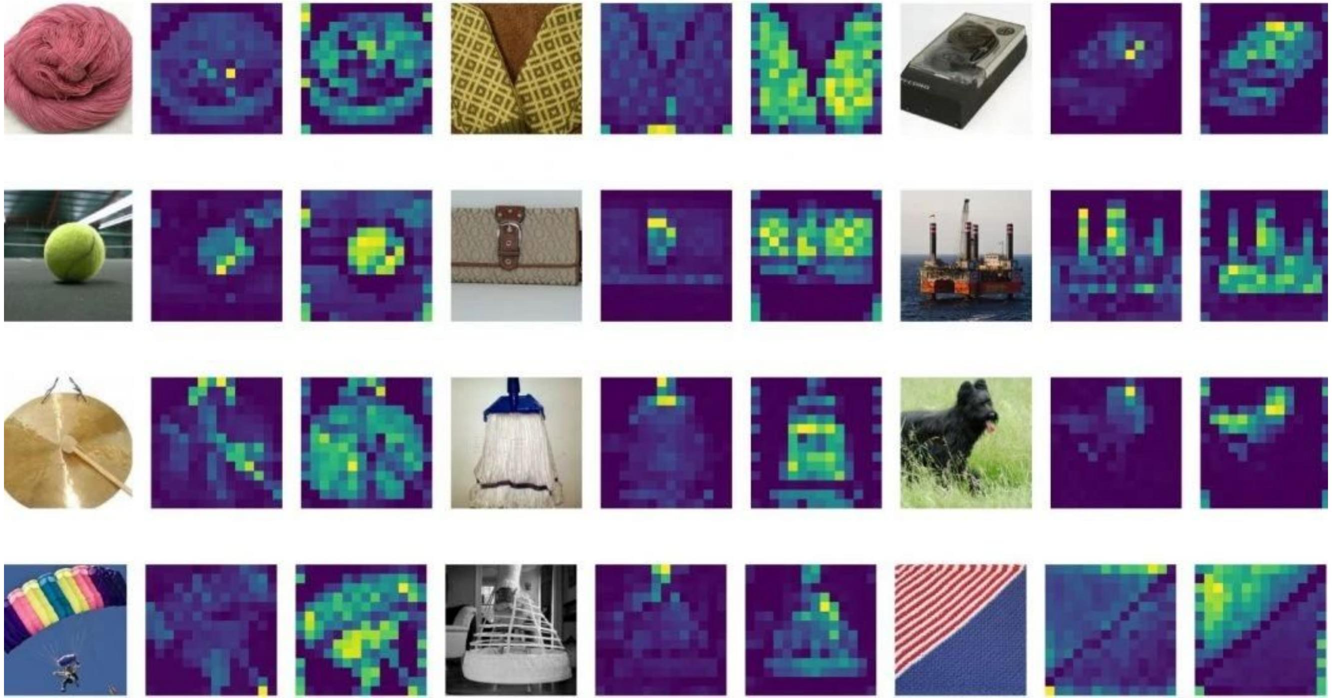
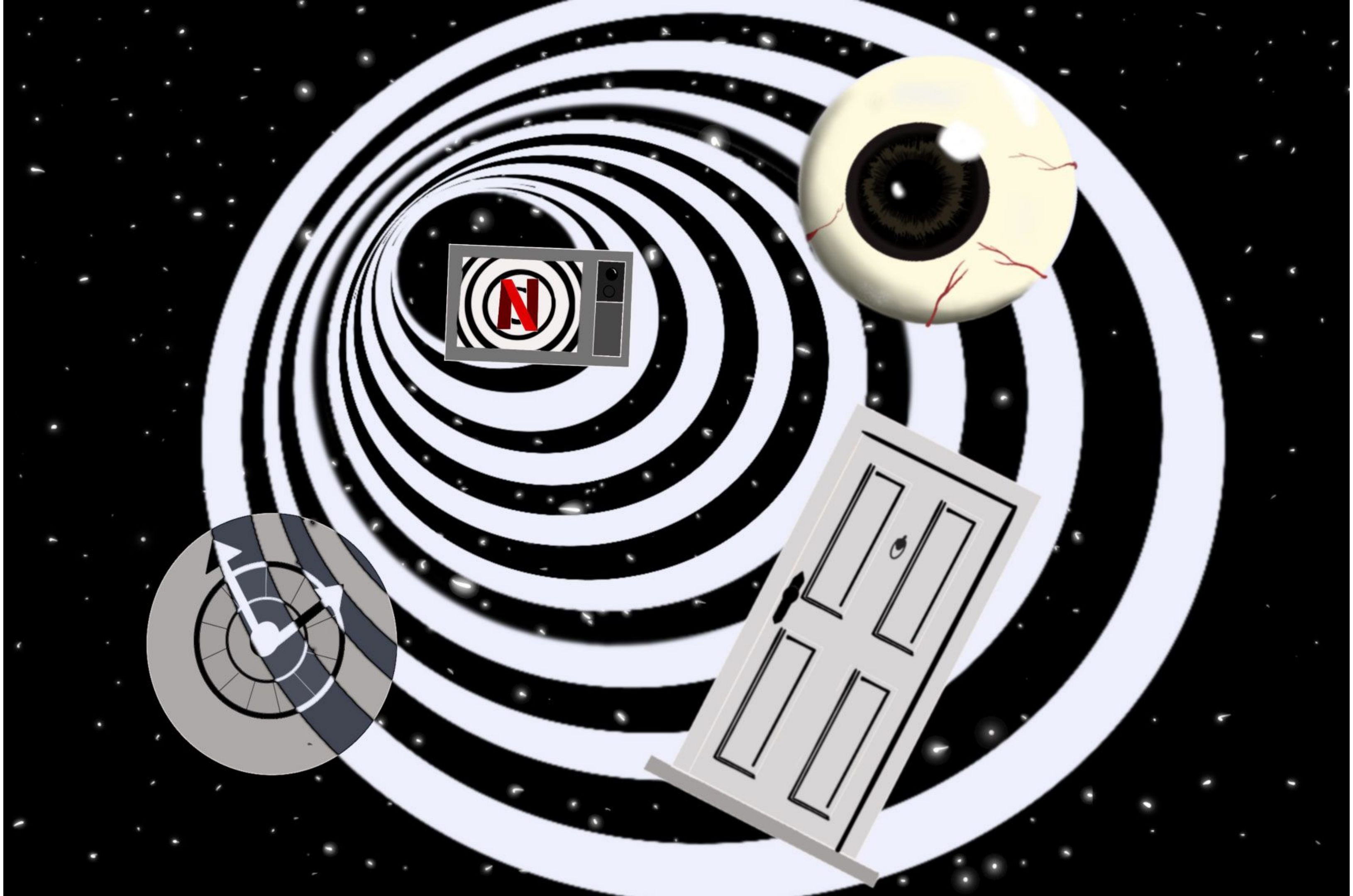


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Vision Transformer (ViT)

Some attention maps from a ViT





Text-Image models

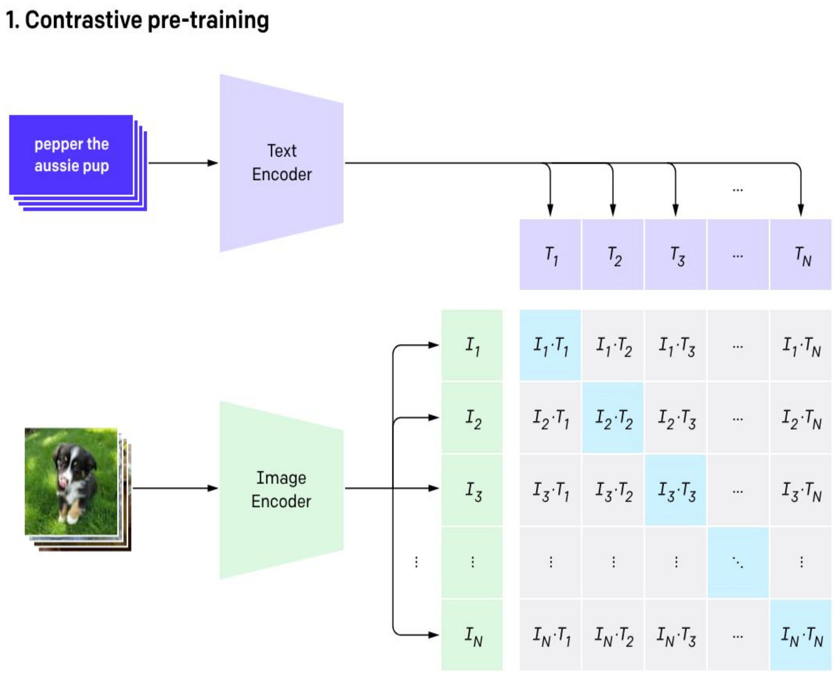
Image + text

Why using text and images?

- Appreciating natural language as a training signal → multi-modality
- No burdensome label crafting
- More scalable data (lots of it)
- Flexible zero-shot transfer

CLIP (Contrastive Language–Image Pre-training) 2021

- Encodes image, and text to similar embeddings
- Proprietary dataset WebImageText 400M of various images with a caption text from the internet → openCLIP alternative 5B images
- Trained with **contrastive learning,** maximizing cosine similarity of corresponding image and text
- CLIP's output image embeddings contain both style and semantics
- Multi-modal understanding
    - leverage natural language as a flexible prediction space to enable generalization and transfer



CLIP contrastive losses

- Image/text embedding vectors: $$\mathbf{v} = g_v(f_v(\tilde{\mathbf{x}}_v)) \qquad \mathbf{u} = g_u(f_u(\tilde{\mathbf{x}}_u))$$
- Image → text contrastive loss: $$\ell_i^{(v\ !\ u)} = -\log \frac{\exp(\langle \mathbf{v}_i, \mathbf{u}_i \rangle / \tau)}{\sum_{k=1}^{N} \exp(\langle \mathbf{v}_i, \mathbf{u}_k \rangle / \tau)}$$
- Text → image contrastive loss: $$\ell_i^{(u\ !\ v)} = -\log \frac{\exp(\langle \mathbf{u}_i, \mathbf{v}_i \rangle / \tau)}{\sum_{k=1}^{N} \exp(\langle \mathbf{u}_i, \mathbf{v}_k \rangle / \tau)}$$
- Loss function: $$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(\lambda \ell_i^{(v\ !\ u)} + (1 - \lambda) \ell_i^{(u\ !\ v)} \right)$$

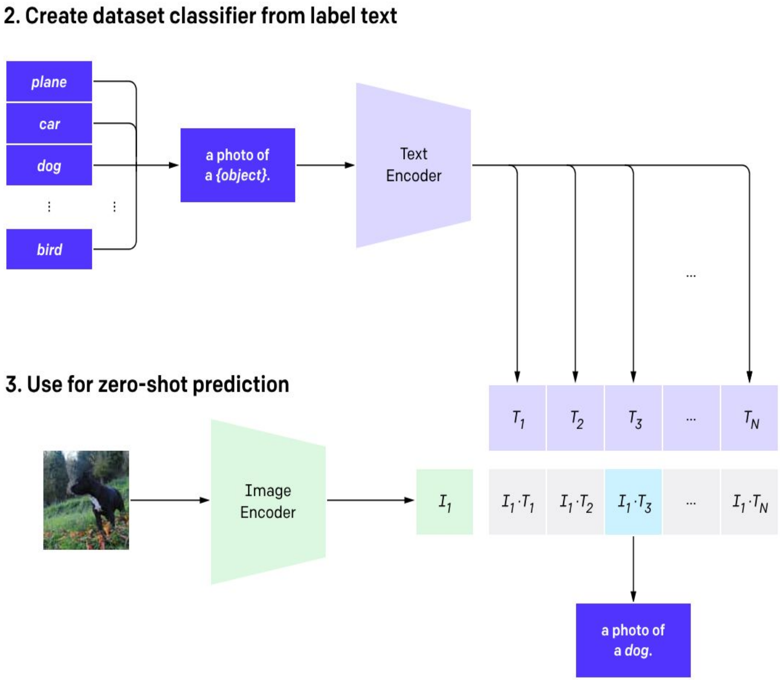
CLIP architecture

- text and image have separate transformer encoders
- visual encoder is ViT (vision transformer)
- text encoder is GPT-2 transformer
- the fixed-length text embedding is extracted from [EOS] token position,
- trained on 256 GPUs for 2 weeks

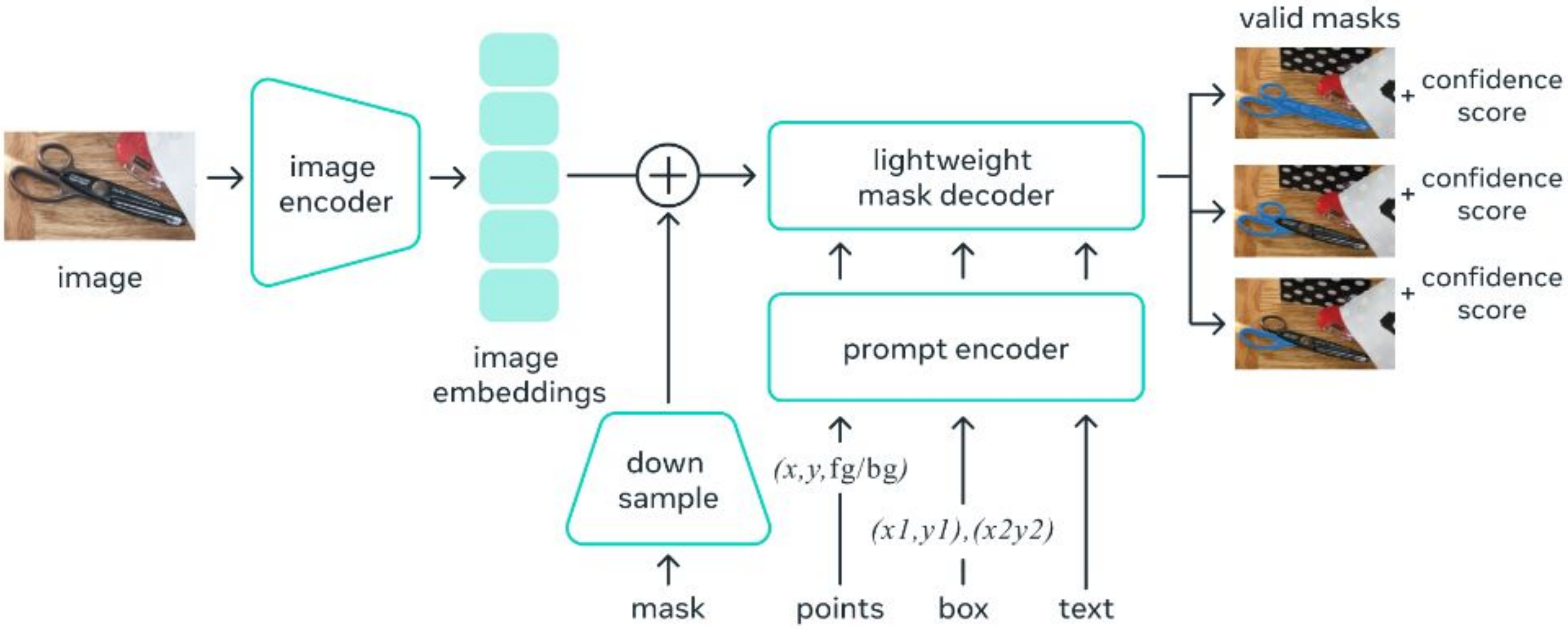
CLIP zero shot classification

- create for each class a text -> embedding
- cosine similarity between image and text embeddings

- Zero-shot classification, but fails on abstract or systematic tasks like counting



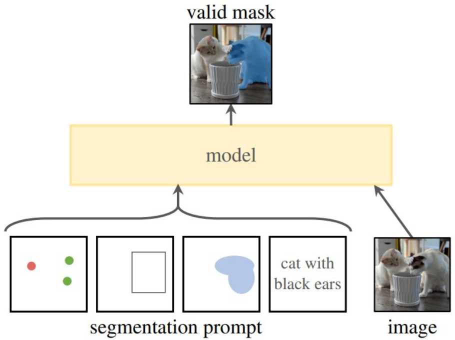
Segment Anything Model [Kirillov et al. 2023]



https://segment-anything.com/demo

Segment Anything Model (SAM)

- It is a **prompt-based zero-shot image segmentation model**: It can segment any object without prior training of the specific object class.
- It is based on a **Vision Transformer (ViT)** pretrained with a self-supervised Masked Autoencoder (MAE) strategy.
- SAM was trained using billions of images and high-quality segmentation masks from diverse image sources (SA-1B Dataset).
- SAM is trained to respond to various prompts (points, boxes, text)



Segment Anything Model

Resolving ambiguity. With one output, the model will average multiple valid masks if given an ambiguous prompt. To address this, we modify the model to predict multiple output masks for a single prompt (see Fig. 3). We found 3 mask outputs is sufficient to address most common cases (nested masks are often at most three deep: whole, part, and subpart). During training, we backprop only the minimum loss [15, 45, 64] over masks. To rank masks, the model predicts a confidence score (*i.e.*, estimated IoU) for each mask.

Figure 3: Each column shows 3 valid masks generated by SAM from a single ambiguous point prompt (green circle).