Vision Generative AI

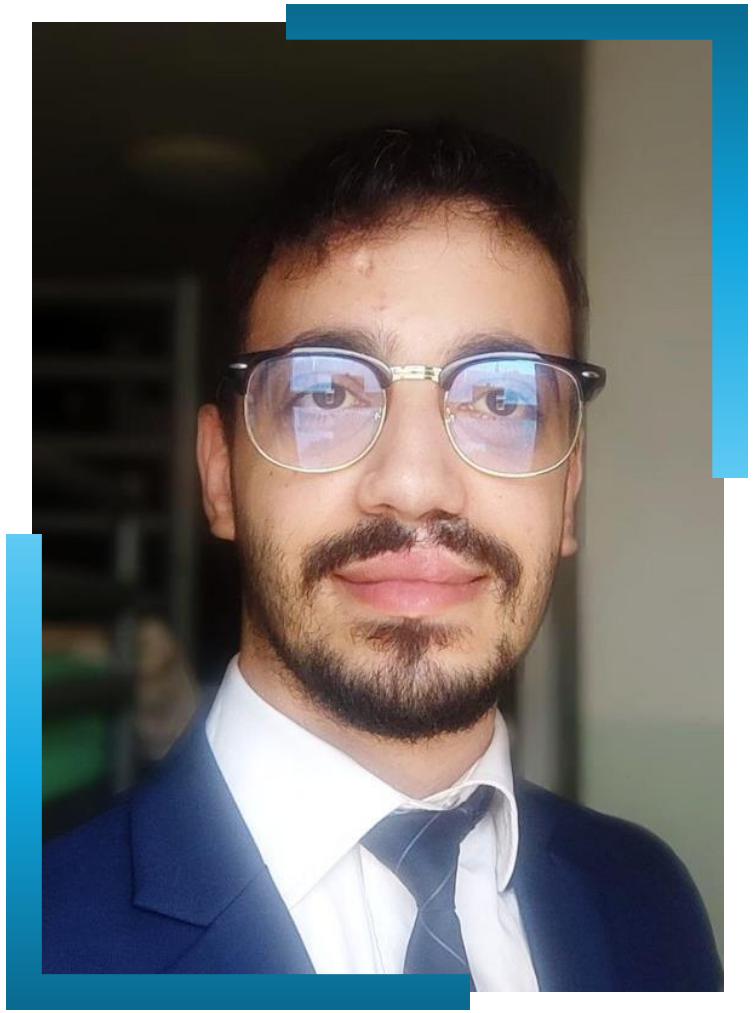
AN INTRODUCTION

By:

Dr. Safouane El Ghazouali

safouane.elghazouali@toelt.ai / safouane.elghazouali@hslu.ch

BIO



Safouane El Ghazouali, Ph. D.

- Senior researcher in AI at TOELT
- Senior data scientist in Computer Vision
- Experienced in VLM / LLM inference and fine-tuning
- Buit a web app UI for fast annotation (VisioFirm)
- Experienced in training / fine-tuning of ML model for domain-specific application

Background

- Ph.D. in AI applied to 3D metrology, LNE, Paris
- Master in computer vision and embedded systems, Polytechnique Haut-de-France, Valenciennes

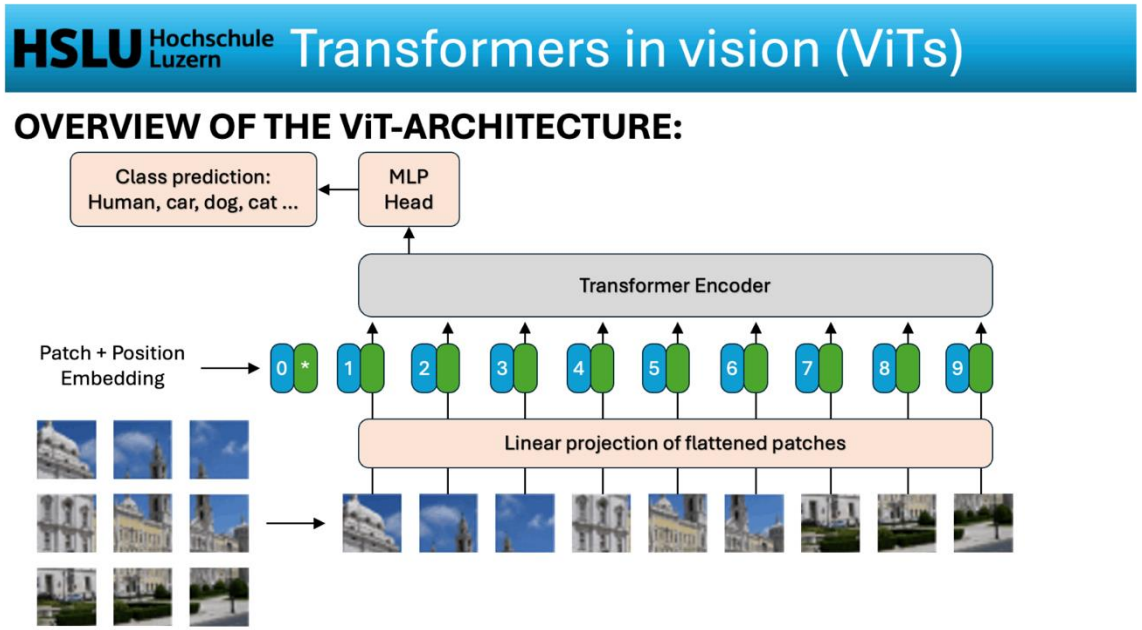
GitHub repo for the course :

https://github.com/safouaneelg/HSLU-GENAI

Transformers in Vision Course - HSLU

Welcome to the GitHub repository for the "Transformers in Vision" course at HSLU.

This repo contains materials and hands-on exercises for Day 3 and Day 4, from the bootcamp starting August 25, 2026.



AGENDA. DAY 1

- Wild introduction to computer vision
  - Basic image operations
  - Feature Detection and Description
  - Motion Analysis
- Open-source frameworks
  - Pillow / OpenCV
  - Pytorch / Tensorflow / Keras
  - HuggingFace / Unsloth / Ollama / Llama.cpp
  - Hand-on: Python env setup
- Neural Network architectures
  - Basic architecture explained
  - More efficient architectures
  - Hand-on and use of frameworks and pretrained models

AGENDA. DAY 2

- Pre-requisits
- Introduction to Generative AI
- Autoencoder
  - Introduction to autoencoders
  - Path to Variational Autoencoders
- Part 2.
  - From Adversarial Training to GANs
  - GAN's Architecture
  - GAN's objective
  - DCGANs
- Part 3.
  - From scratch hand-on implementation of DCGAN
  - Implementation of Conditional GAN

AGENDA. DAY 3

Generative AI

- Background & Motivations
- Transformers in vision (ViTs)
- Foundation models used in new generation of Generative AI models : CLIP and Zero-Shot Learning
- Hands-on:
  - Pre-trained classification model
  - Visualization of features map
  - Finetuning ViT on classification task
  - Inference CLIP for zero-shot classification
- Diffusion Model – introduction to Markov Chain
- Hand on how to implement Diffusion

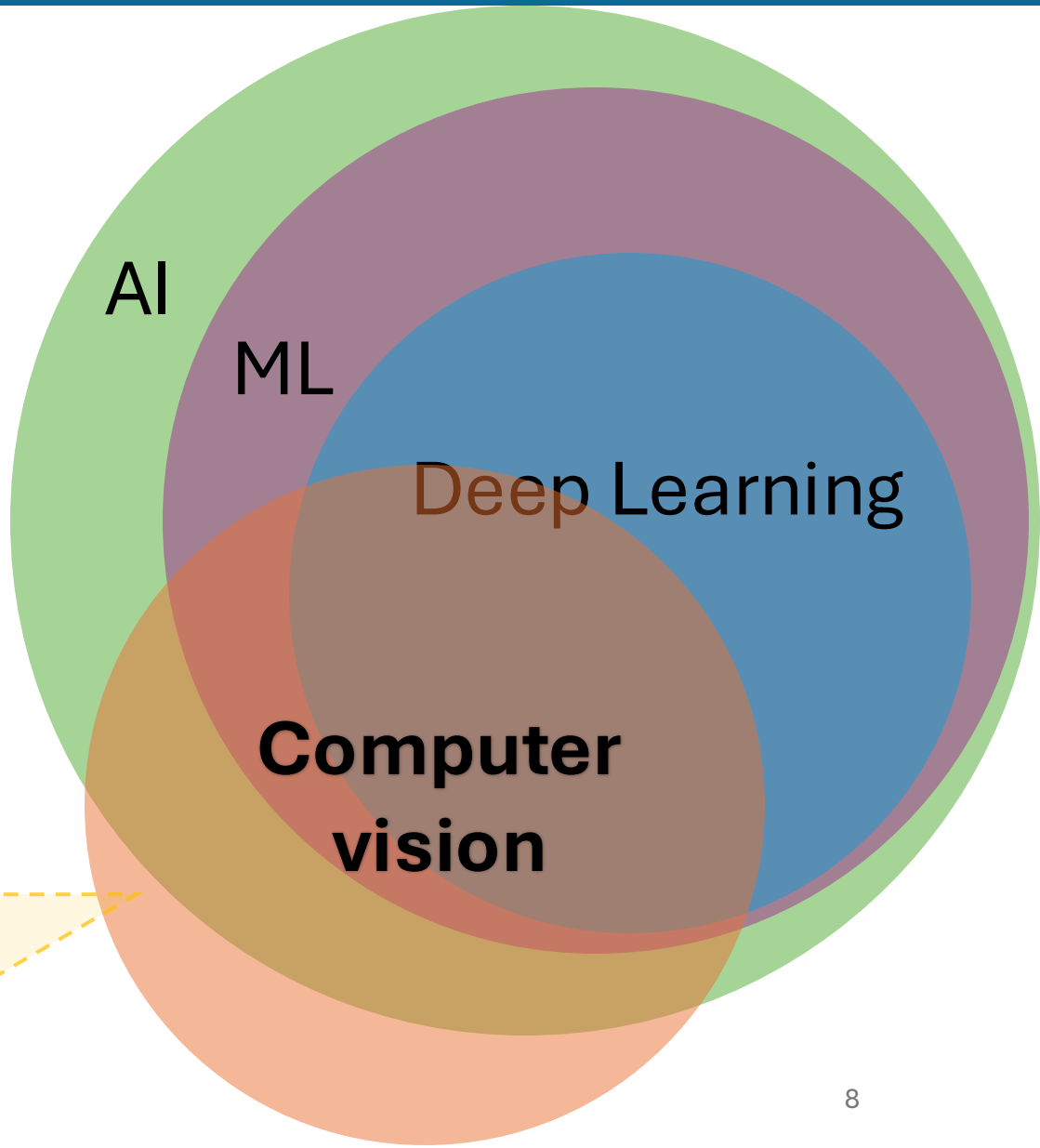
HSLU Hochschule Luzern

General introduction

General introduction

Computer vision includes:

1- Image/signal processing
2- 3D reconstruction
3- Feature extraction
4- Image restoration
5- Image classification
6- Object detection
7- Semantic/instance segmentation
8- Scene understanding
9- Vision Generative AI
10- Optical flow analysis
11- Pose estimation
12- Video tracking



Overview of neural architectures evolution

Computer vision evolution timeline

1950

Foundations in optics and early mathematical-based processing. Focus on understanding light, vision, pinhole and basic digital imaging.

1970

Shift to more structured NNs with hierarchical feature learning. Introduction of convolutional ideas for shift-invariant pattern recognition.

2014

Appearance of generative AI model through Generative Adversarial Networks

2017

Emergence of advanced modular architectures and powerful backbones able to extract deep features.
The most known backbones

- VGGNet 16–19 layers with uniform 3x3 convolutions for depth exploration
- AlexNet: 8-layer CNN winning ImageNet challenge
- DenseNet: Layer-wise connections for feature reuse

2020

Rise of pure vision transformers, outperforming CNNs on large datasets.
Multimodal integration begins.
Vision Transformer: Patched images fed into transformers for classification.
DeiT (2021): Data-efficient ViT variant.
Swin Transformer (2021): Hierarchical shifted windows for efficiency.

HSLU Hochschule Luzern

Computer vision

HSLU Hochschule Luzern Basic image operations

Technique	Architecture/Example	Description
Filtering & Convolution	Hybrid 2D Gaussian Filter + CNN (e.g., ResNet/VGG)	Applies Gaussian blur as preprocessing to smooth noisy images before CNN classification/detection.
Filtering & Convolution	CNN with Gaussian Blur Augmentation (e.g., AlexNet)	Uses Gaussian blurring in data augmentation to simulate degradations during training.
Sobel Operator	Sobel-Initialized CNN (EdgeNet Variants)	Initializes first conv layer with Sobel kernels for edge detection, then fine-tunes for segmentation.
Sobel Operator	CNN-SVM with Sobel Filter (e.g., VGG-16)	Applies Sobel for edge enhancement before CNN feature extraction and SVM classification.
Erosion	Morphological-Convolutional Neural Network (MCNN)	Replaces early conv layers with erosion/dilation ops, followed by CNN for texture analysis.
Erosion	Hybrid Morphological Neural Networks	Integrates learnable erosion in residual blocks for noise handling in digit recognition.
Histogram Equalization	CNN with Histogram Equalization Preprocessing (e.g., U-Net/SegNet)	Uses adaptive histogram equalization (AHE) to boost contrast before medical segmentation.
Histogram Equalization	Fully Convolutional Network (FCN) for Histogram Learning	Learns adaptive histogram mappings as a contrast module in end-to-end low-light restoration.
Feature Extraction & Matching	SIFT/ORB + Neural Network Classifier (MLP/Shallow CNN)	Extracts keypoints/descriptors, then classifies with NN for traffic/image matching.
Feature Extraction & Matching	SURF-Enhanced Siamese Networks	Augments Siamese CNN with SURF keypoints for triplet loss in image registration.

Basic image operations

1. Image manipulation: **Filtering and Convolution**

Gaussian Blur:
Purpose: Reduces noise and detail by smoothing the image.
Mathematics: Applies a convolution with a Gaussian kernel, which is a 2D matrix derived from the Gaussian function:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation controlling the blur strength, and x, y are pixel coordinates relative to the kernel center. The kernel is normalized to ensure the sum of its elements is 1.

Basic image operations

1. Image manipulation: **Sobel Operator**

Uses two 3x3 kernels to approximate the gradient in the horizontal (G_x) and vertical (G_y) directions:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The gradient magnitude is computed as:

$$G = \sqrt{G_x^2 + G_y^2}$$

or approximated as $G = |G_x| + |G_y|$.

Basic image operations

1. Image manipulation: **Erosion**

Purpose: Shrinks bright regions and enlarges dark regions, removing small objects or noise.

Concept : For a binary image I and structuring element S, erosion is defined as:

$$I \ominus S = \{z \mid S_z \subseteq I\}$$

where S_z is the structuring element centered at pixel z. A pixel remains 1 only if the entire structuring element fits within the foreground.

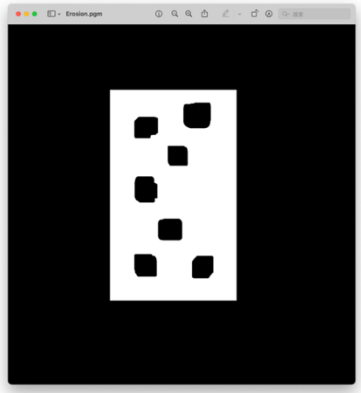
Process result of "noisy_fingerprint.pgm" and "noisy_rectangle.pgm" :

Source Image:

Result of erosion (element size: 3*3):

Source Image:

Result of erosion (element size: 50*50):



Basic image operations

1. Image manipulation: **Histogram equalization**

Purpose: Enhances contrast by redistributing pixel intensities.

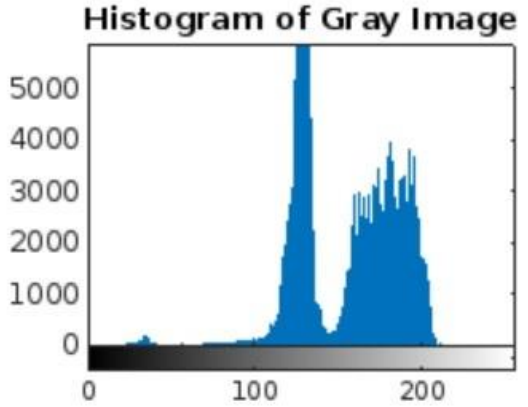
Concept: For an image with intensity levels in [0, L-1], compute the histogram $h(r)$,the number of pixels with intensity r. The cumulative distribution function (CDF) is:

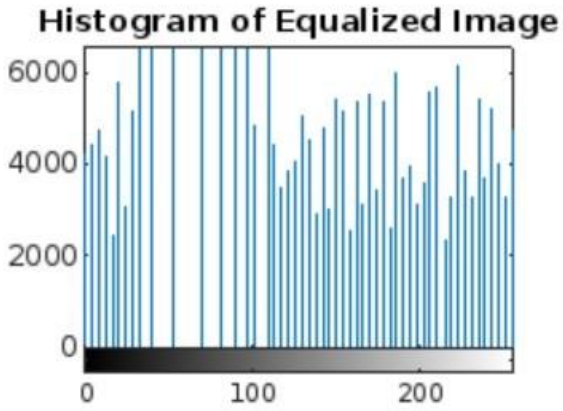
$$CDF(r) = \sum_{k=0}^{r} h(k)$$

The equalized intensity for a pixel with intensity $r$is:

$$r' = floor\left(\frac{CDF(r) - CDF_{min}}{n - CDF_{min}} \cdot (L-1)\right)$$

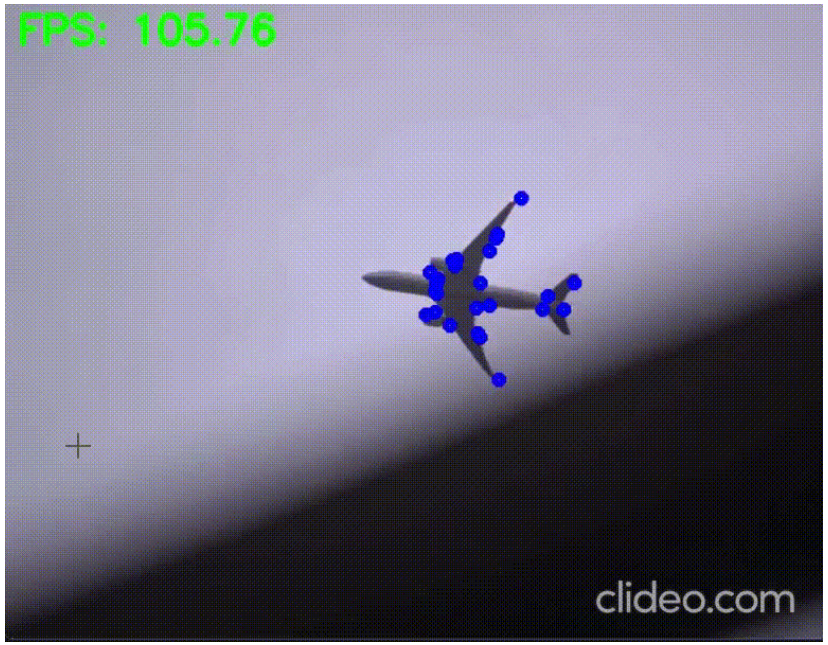
where $n$is the total number of pixels, and $CDF_{min}$is the minimum non-zero CDF value.





Feature extraction

Feature detection and description are fundamental in computer vision for identifying and characterizing key points (e.g., corners, edges) in images to enable tasks like image matching, object recognition, or 3D reconstruction. Three popular methods are **SIFT**, **SURF**, and **ORB**.



Feature extraction

SIFT (Scale-Invariant Feature Transform)

detects and describes local features in an image that are invariant to scale, rotation, and partially to illumination changes. It identifies keypoints (distinctive image points) and generates descriptors (numerical representations) for matching across images.

1.Scale-Space Extrema Detection: SIFT builds a scale-space pyramid by convolving the image with Gaussian filters at different scales. It detects keypoints as local maxima/minima in the Difference of Gaussians (DoG), which approximates the Laplacian of Gaussian.

2.Keypoint Localization: Refine keypoint locations by filtering out low-contrast or edge-based points.

3.Orientation Assignment: Assign an orientation to each keypoint based on local gradient directions

- **Scale-Space**: $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$
  - $I(x, y)$:Input image intensity at pixel (x, y).
  - $G(x, y, \sigma)$:Gaussian kernel with scale σ.
  - $*$: Convolution operation.
  - **DoG**: $D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$,where $k$is a constant scale factor.
- Keypoints are extrema in $D(x, y, \sigma)$across spatial and scale dimensions.

Parameters:

- σ: Gaussian scale, controlling blur level.
- k: Scale factor between consecutive levels in the pyramid.
- x, y: Pixel coordinates.

Feature extraction

SURF (Speeded-Up Robust Features)

faster alternative to SIFT, also detecting scale- and rotation-invariant keypoints and descriptors. It uses integral images and box filters to approximate computations, improving speed.

Scale-Space via Integral Images: Use integral images to compute approximations of the Hessian matrix for keypoint detection.

Keypoint Detection: Identify keypoints as maxima of the determinant of the Hessian matrix.

Orientation Assignment: Use Haar wavelet responses to assign dominant orientations.

Hessian Matrix: For a pixel at (x, y) with scale σ, the Hessian is:

$$H(x, y, \sigma) = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix}$$

L_{xx}, L_{xy}, L_{yy} :Second-order Gaussian derivatives approximated using box filters.

Keypoints are where $\det(H) = L_{xx}L_{yy} - (L_{xy})^2$ is maximized.

Parameters:

σ: Scale of the filter, controlling blob detection size.

x, y: Pixel coordinates

Feature extraction

ORB (Oriented FAST and Rotated BRIEF)

combines **FAST** (keypoint detector) and **BRIEF** (descriptor) to create a fast and efficient alternative to SIFT and SURF. It's designed for real-time applications.

Keypoint Detection (FAST): Detect corners by comparing a pixel's intensity to a surrounding circle of pixels. If enough pixels differ significantly, it's a corner.

Orientation Assignment: Use intensity centroid to assign orientation, making keypoints rotation-invariant.

Descriptor Generation (BRIEF): Create a binary descrip by comparing intensities of pixel pairs in a patch around keypoint.

FAST Score: For a pixel p with intensity I_p ,compare to a circle of 16 pixels. A keypoint is detected if Ncontiguous pixels satisfy:

$$| I_p - I_i |> t, i \in circle$$

t: Intensity threshold.

N: Number of contiguous pixels (typically 9–12).

BRIEF Descriptor: For a keypoint, compute a binary vector:

$$d_i = \begin{cases} 1 & if\ I(p_i) > I(q_i), \\ 0 & otherwise \end{cases}$$

p_i, q_i :Predefined pixel pairs in the patch.

Motion analysis

Optical Flow

refers to the pattern of apparent motion of objects, surfaces, or edges in a visual scene caused by relative motion between the observer (camera) and the scene. It estimates the motion vector (u, v) for each pixel between two consecutive image frames.

Assumptions:

Brightness Constancy: Pixel intensity $I(x, y, t)$remains constant over time:

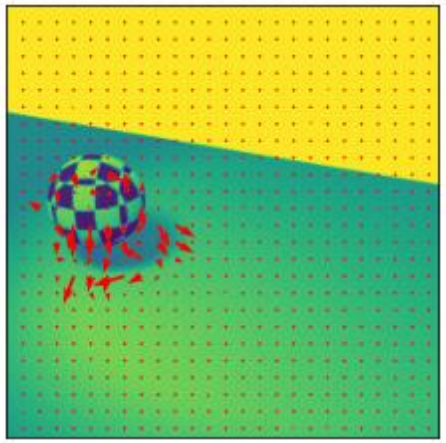
$$I(x, y, t) = I(x, +, uy + vt + 1)$$

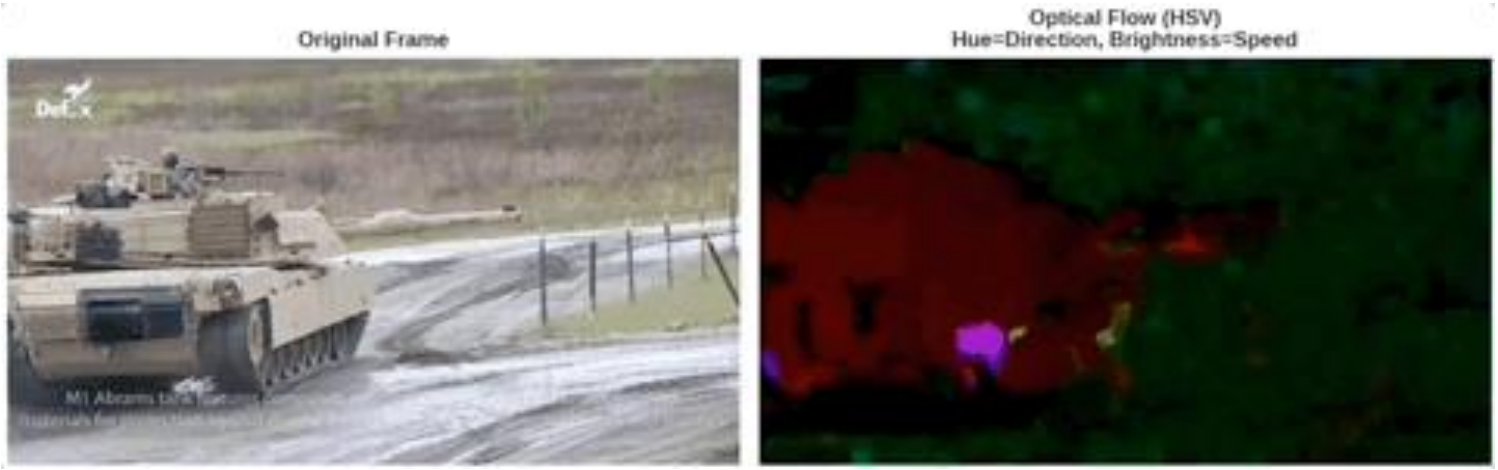
u, v :Horizontal and vertical motion components.

t: Time (frame index).

Small Motion: Motion between frames is small, allowing linear approximations.

Spatial Coherence: Neighboring pixels have similar motion.





HSLU Hochschule Luzern

Open-source frameworks

All those operations can be mathematically heavy on CPU

Framework	Category	Primary Use Cases	License	Language
OpenCV	Computer Vision	Filtering, feature detection, object recognition	Apache 2.0	C++ (Python)
Matplotlib	Visualization	Image display, plotting	BSD-compatible	Python
torchvision	ML/DL	Image classification, augmentation	BSD 3-Clause	Python
Pillow	General Image Processing	Image manipulation, format conversion	HPND	Python
scikit-image	General Image Processing	Segmentation, feature extraction	BSD 3-Clause	Python
ImageIO	General Image Processing	Image I/O, metadata handling	BSD 2-Clause	Python
Mahotas	General Image Processing	Feature detection, texture analysis	MIT	Python
SimpleITK	General Image Processing	Medical image processing, segmentation	Apache 2.0	C++ (Python)
Albumentations	ML/DL	Image augmentation for ML	MIT	Python
Kornia	ML/DL	Differentiable image processing, augmentation	Apache 2.0	Python

Setup Python Environment

The Problem with Global Python

- Multiple projects with conflicting dependencies
- System Python pollution
- Version conflicts between packages
- Deployment issues
- Security concerns

Example Scenario

Project A needs `numpy==1.19.0`, `torch==2.4.1`
Project B needs `numpy==1.23.0`, `torch==2.9`

Setup Python Environment

Solution: setup virtual environment

1. Using virtual environment *venv*

A lightweight, built-in Python tool for creating isolated Python-only environments. Environment can be created using this command:

```
python -m venv myenvname
```

You can replace ” myenvname” with whatever name you want but a common practice is to name it with a dot before name.

```
# example
python -m venv .venv
```

```
# package installation
pip install [PKG_NAME] # or
pip install -r requirements.txt
```

```
# Activate env: windows
myenv\Scripts\activate

# Linux/macOS 5 source
myenv/bin/activate
```

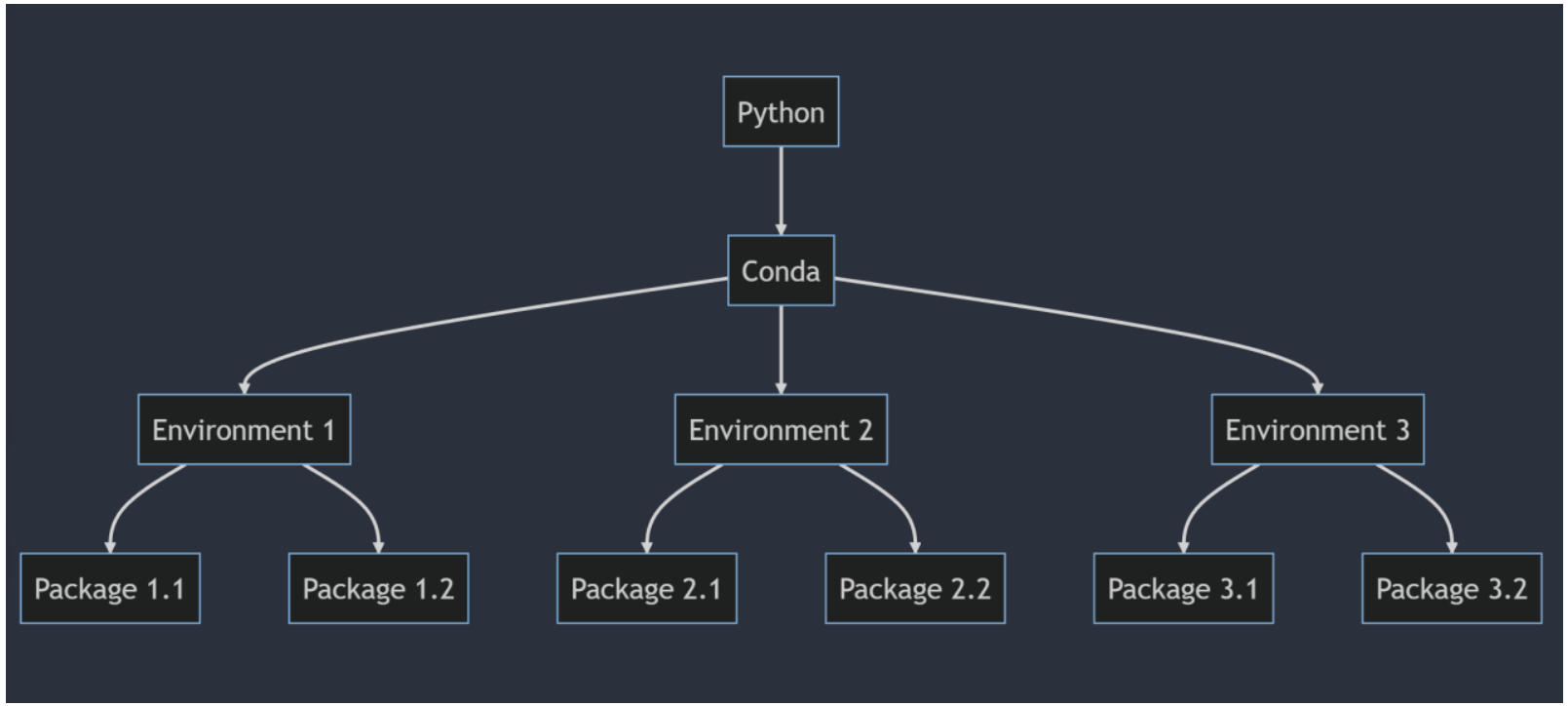
```
# Deactivate
deactivate
# Remove
rmdir /s myenv # Windows
rm -rf myenv/ # Linux/macOS
```

Setup Python Environment

Solution: setup virtual environment

2. Using Conda (Anaconda/Miniconda)

Better alternative, more powerful, language-agnostic package and environment manager that excels in data science and scientific computing by handling complex dependencies beyond Python



Setup Python Environment

Solution: setup virtual environment

2. Using Conda (Anaconda/Miniconda)

Download and installation:
https://www.anaconda.com/docs/getting-started/miniconda/install

Basic install instructions

- Windows installation
- macOS/Linux installation
- Verify your install

Setup Python Environment

After installation, a conda environment can be create using this command:

```
conda create --name [ENV_NAME] python=[PYTHON_VERSION]
```

For example

```
conda create --name genai python=3.10

# Activate
conda activate genai

# Install packages
pip install matplotlib
pip install -c conda -forge package_name
```

For repeatability, environment can be saved

```
# List environments
conda env list

# Export environment
conda env export > environment.yml
conda env create -f environment.yml
```

HSLU Hochschule Luzern

OpenCV

Hands-on:

Python Environment Setup

Matplotlib

HSLU Hochschule Luzern

OpenCV

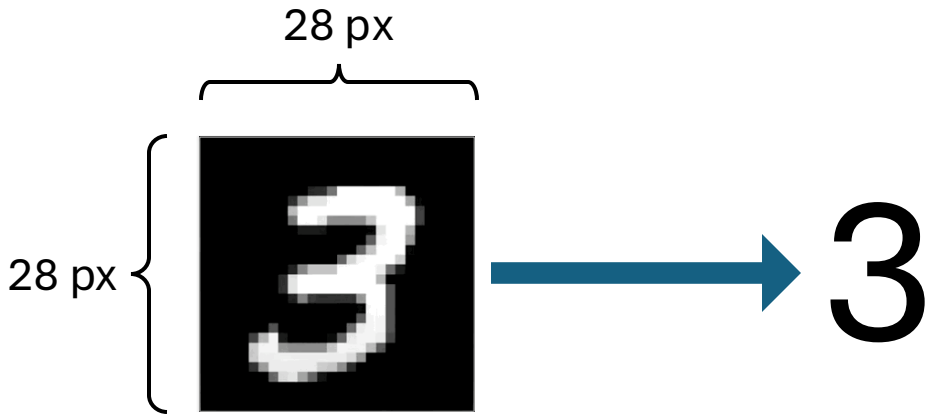
Hands-on:

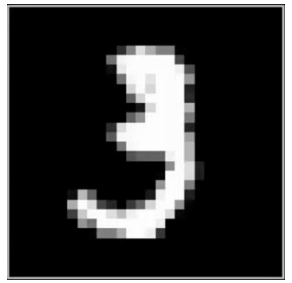
Image manipulation

Matplotlib

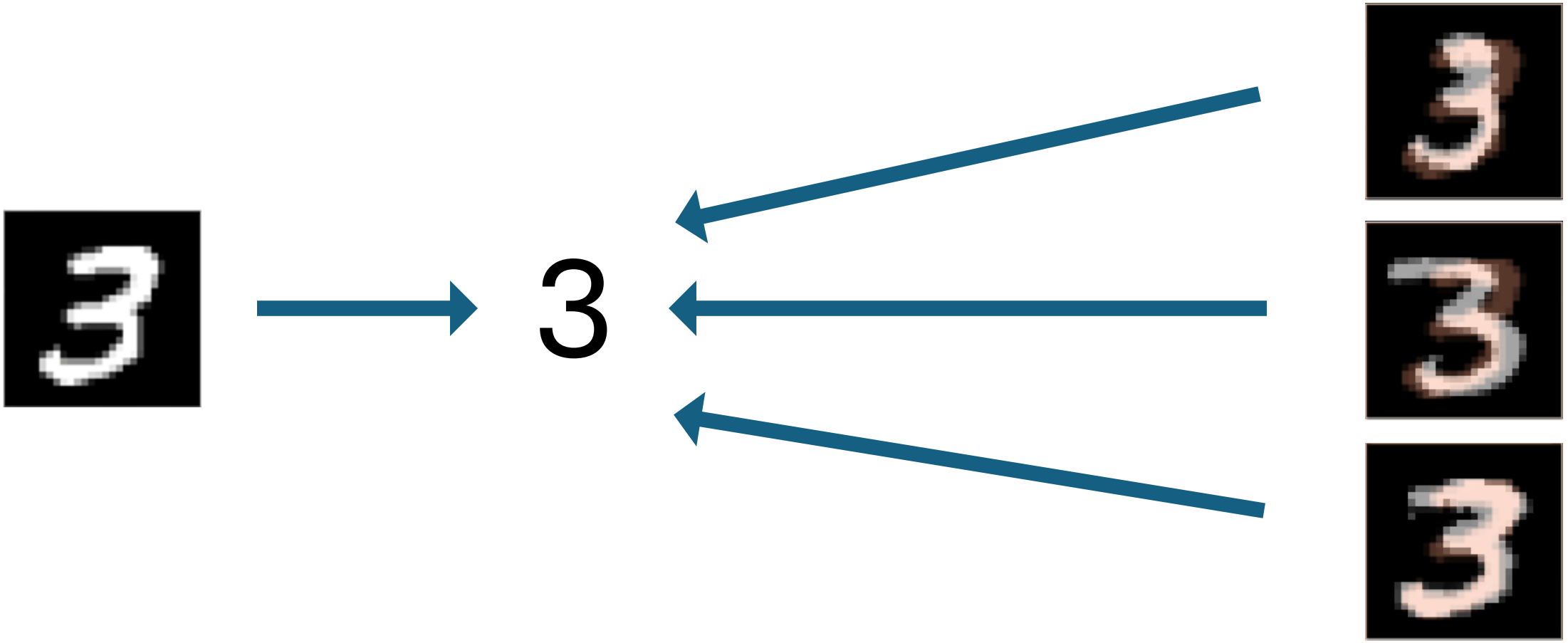
Neural Network architectures

Friendly reminder

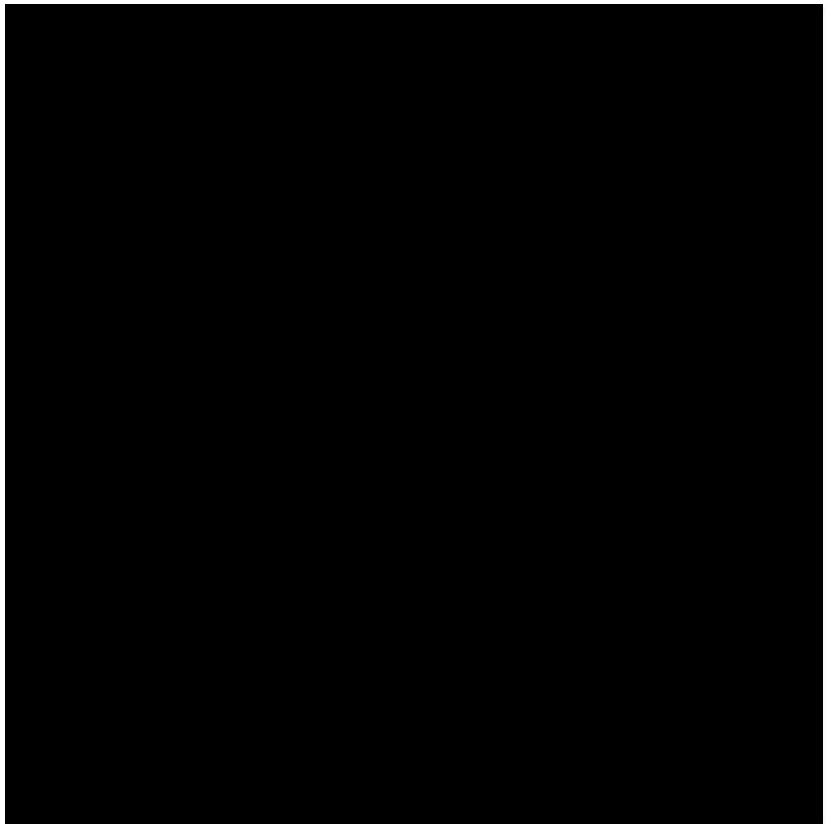




Neural Network architectures



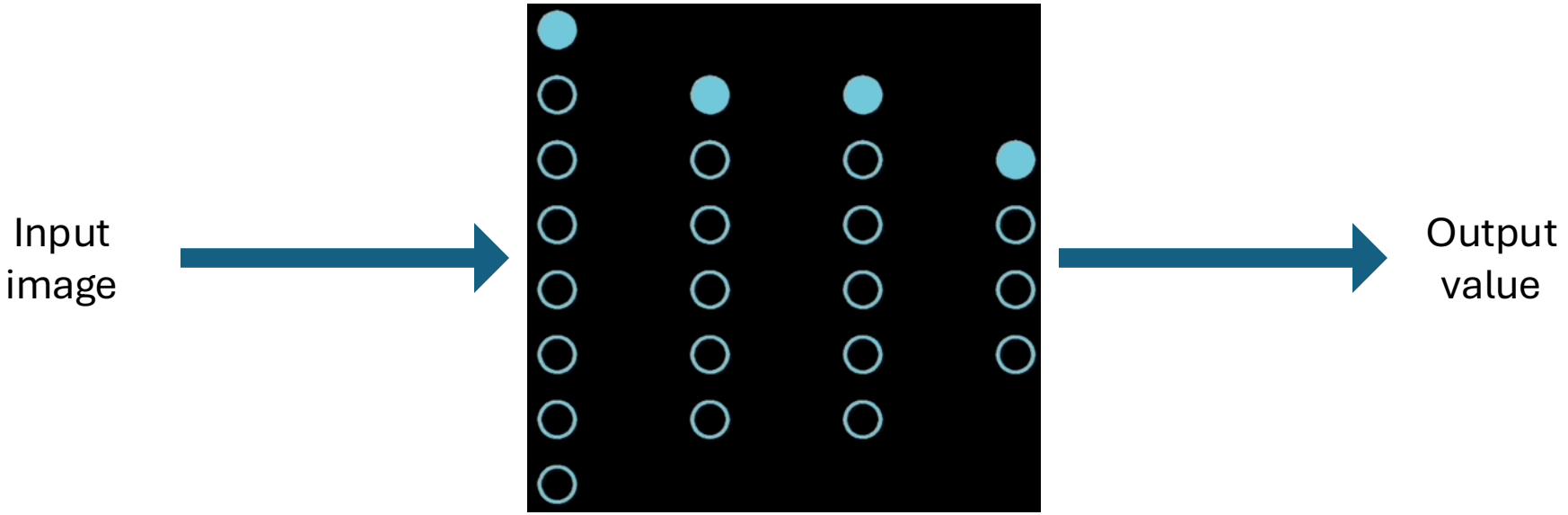
Neural Network architectures



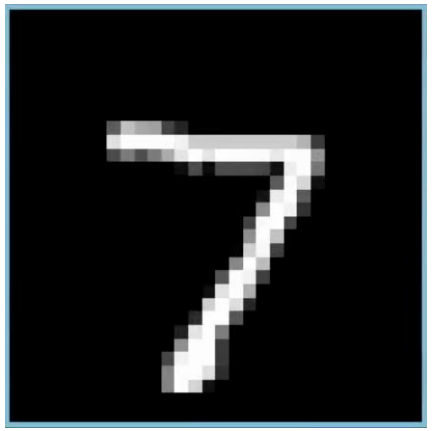
0 ?
1
2
3
4
5
6
7
8
9

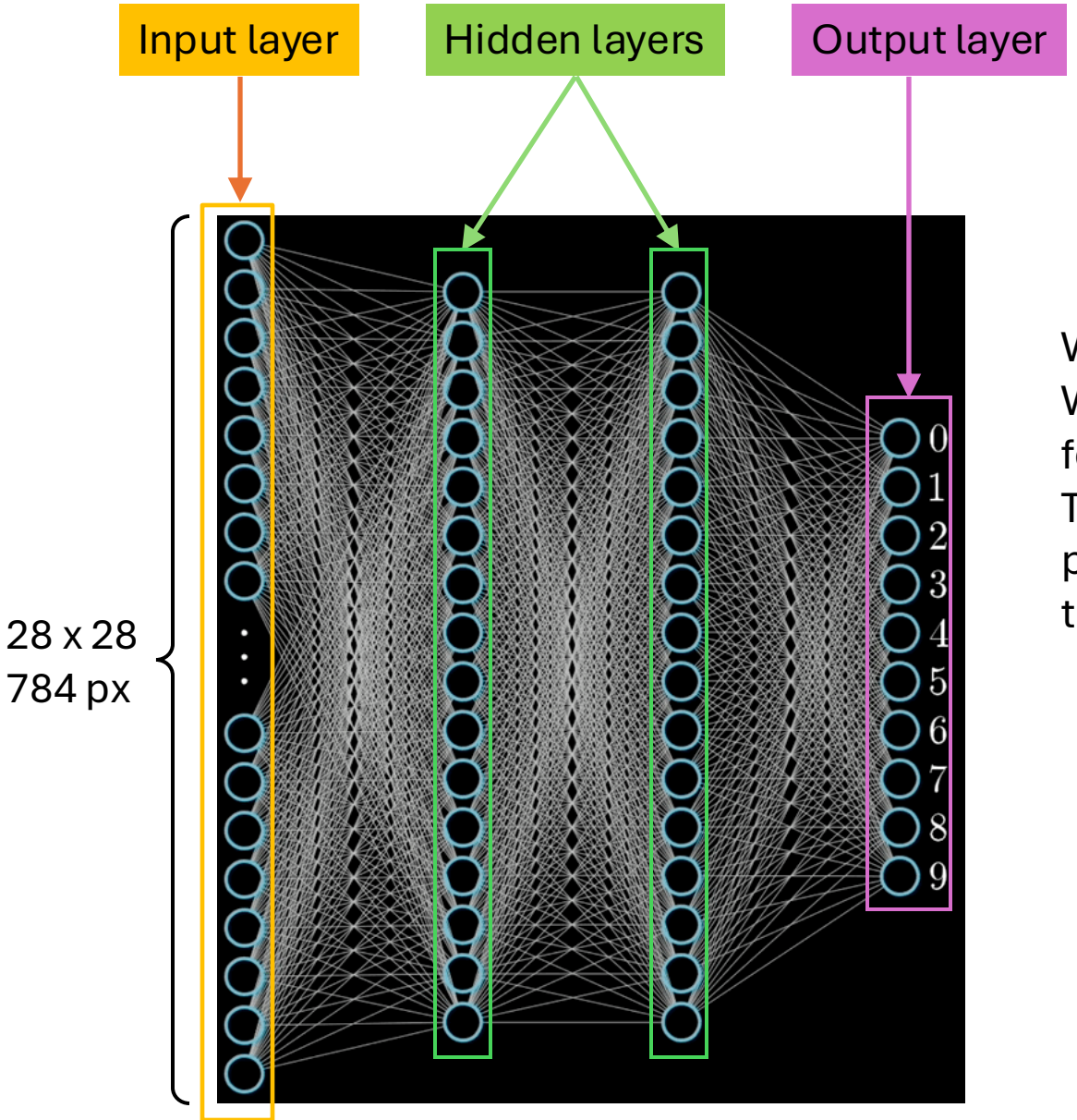
Neural Network architectures

How does a neural network solve this problem ?



Neural Network architectures





Why layered architecture ?
We hope that the model learn meaninful feature representation down the line.
The layered architecture transforms pixel-based (perceptual) representation to a domain of features

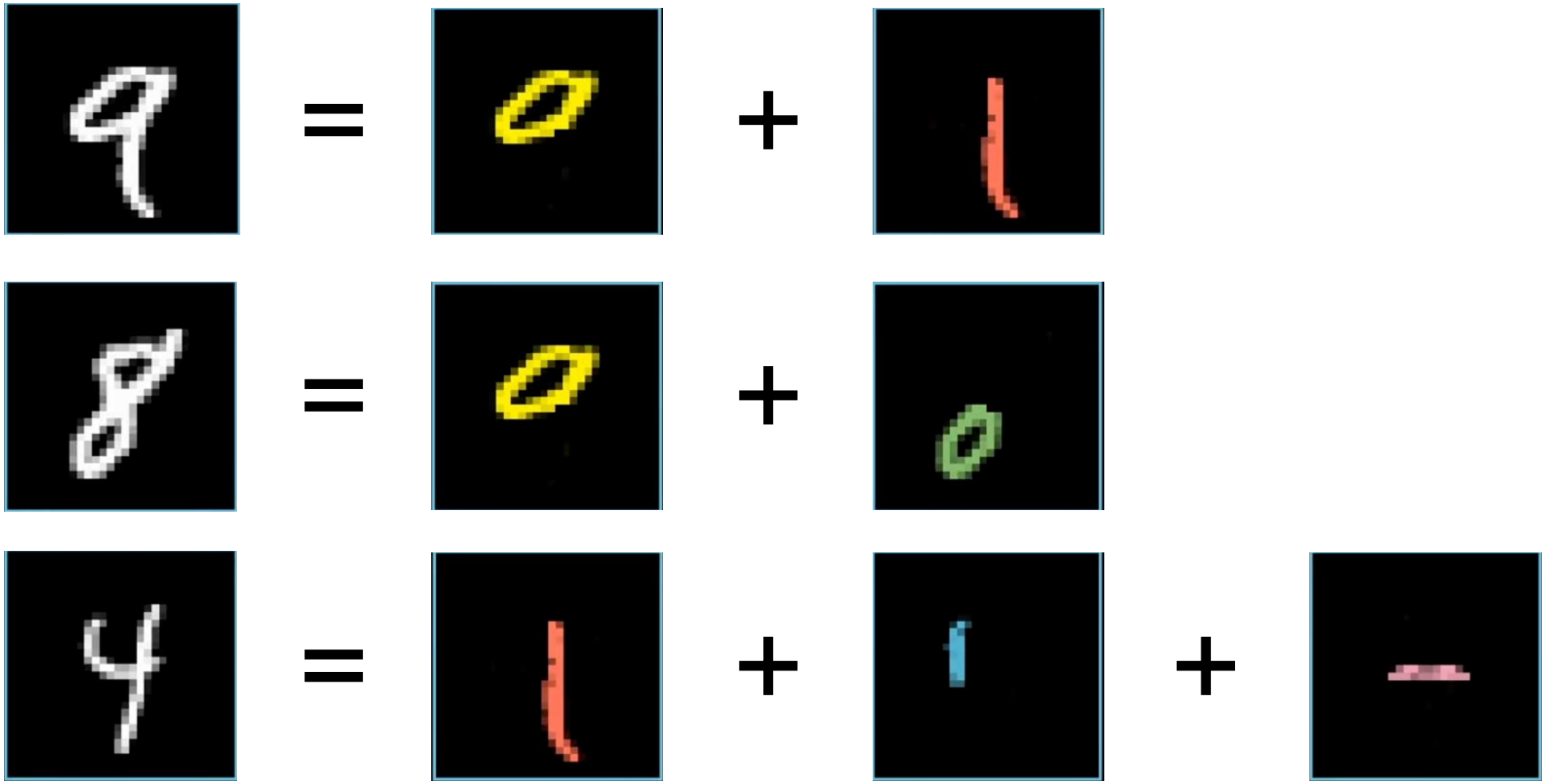
Neural Network architectures

=

=

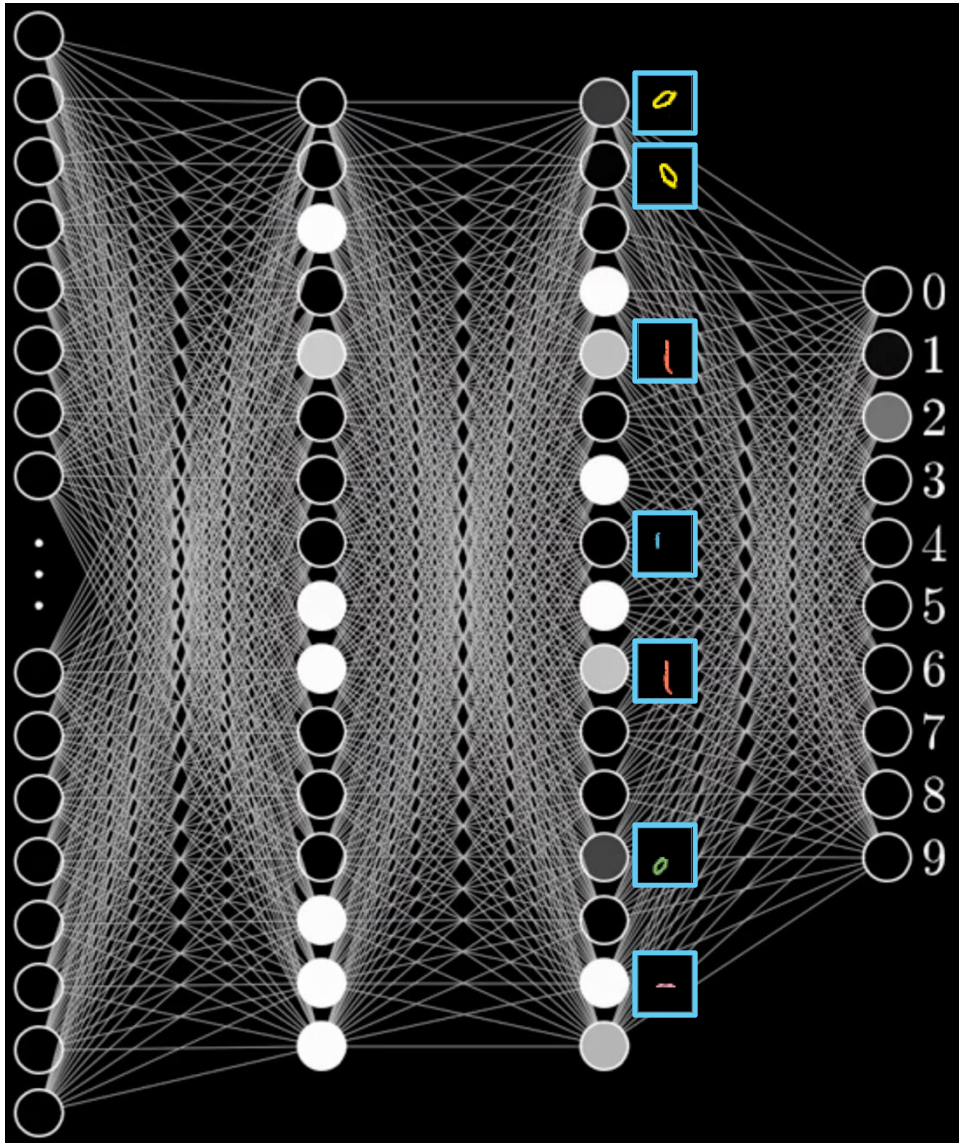
=

Neural Network architectures



Neural Network architectures

Hidden layers should learn hidden patterns beyond the perceptual aspect of the image



Neural Network architectures

Hidden layers should learn hidden patterns beyond the perceptual aspect of the image

Pixel

↓

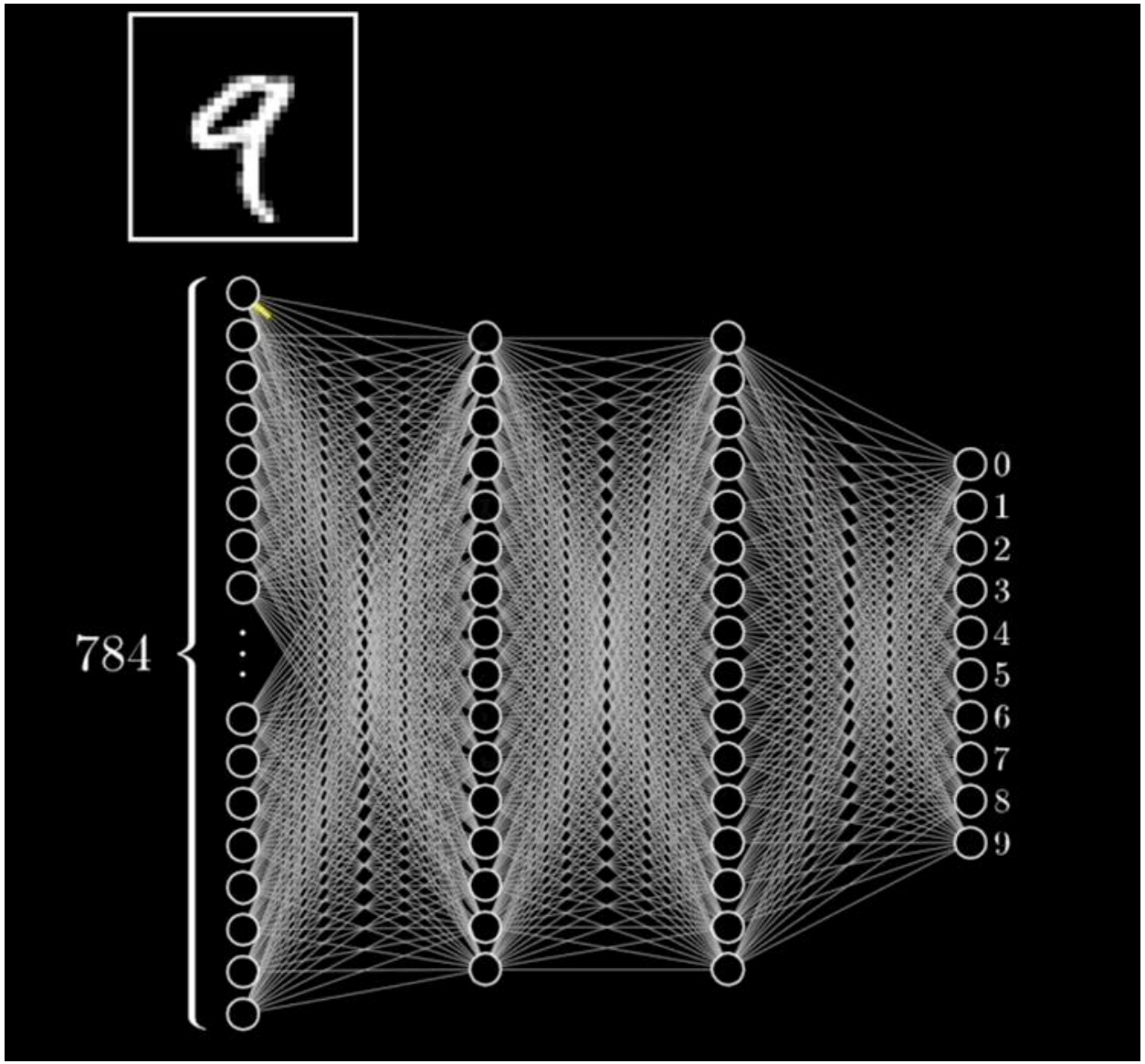
Edges

↓

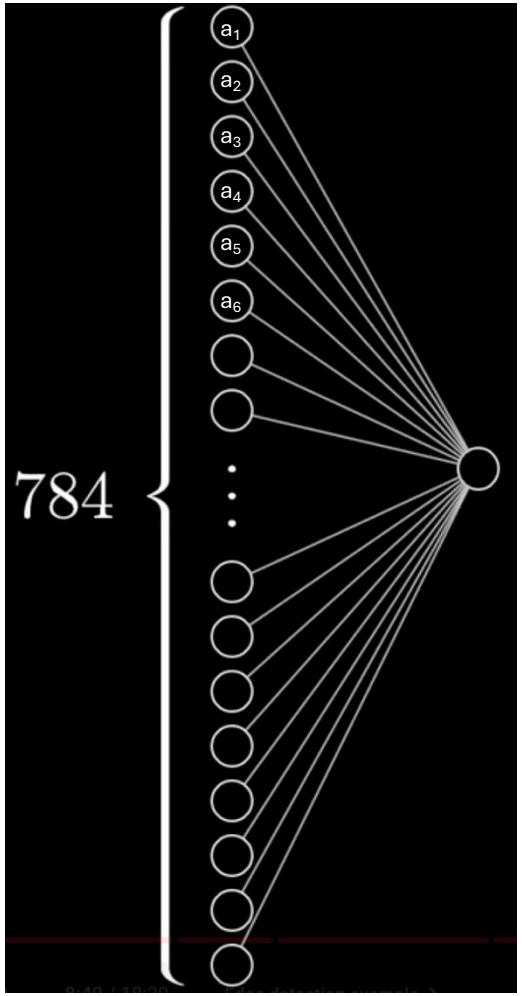
Pattern

↓

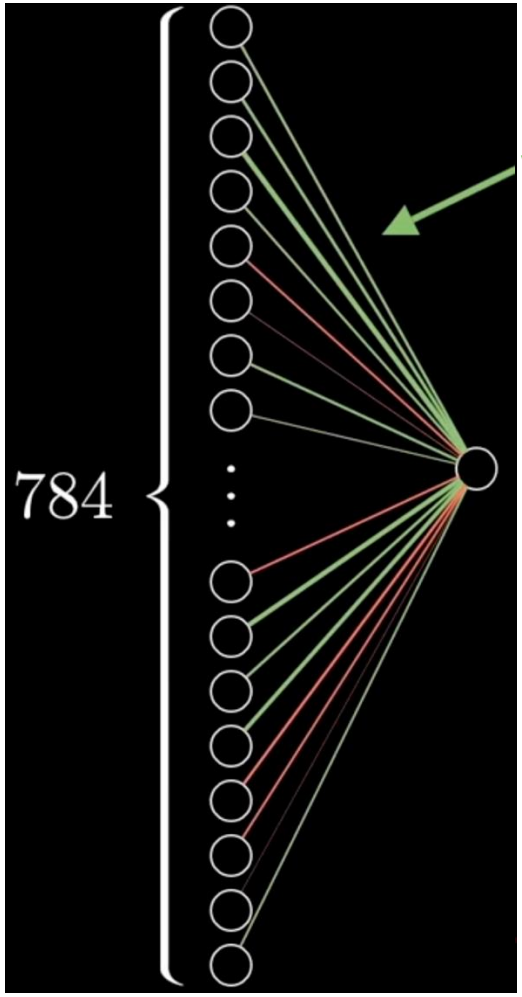
Digits



Neural Network architectures



Neural Network architectures



Weight $\sigma(\omega_1 a_1 + \omega_2 a_2 + \omega_3 a_3 + \omega_4 a_4 + \omega_5 a_5 + \omega_6 a_6 + \ldots) - b$

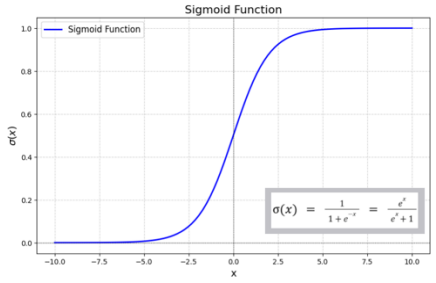
This weighted sum can results in any number –

But we want for the activation to have a value $\in [0, 1]$

Bias term to avoid activating usless neurons

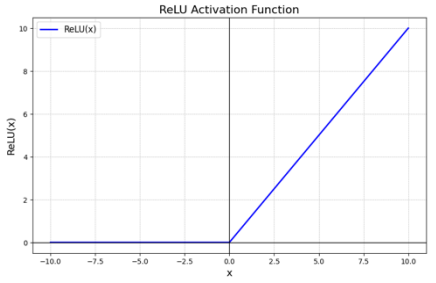
Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

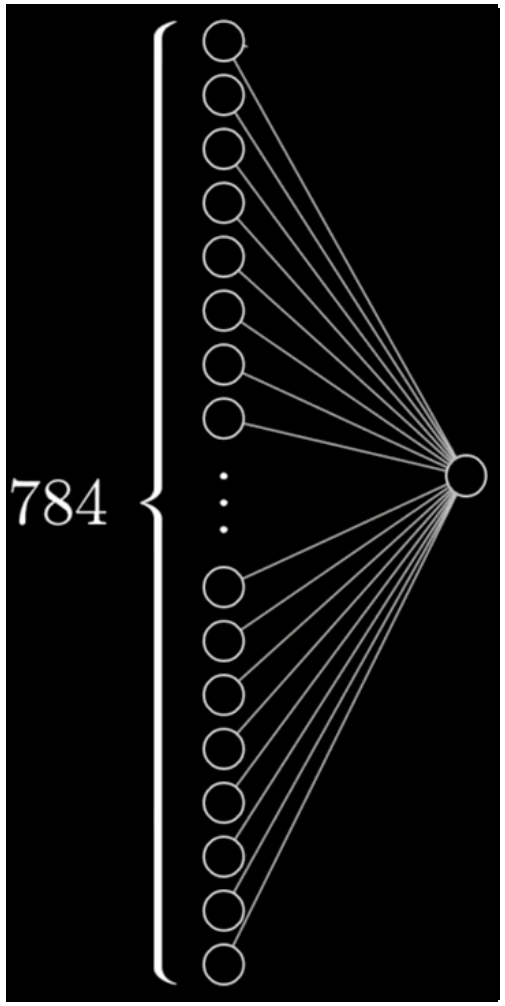


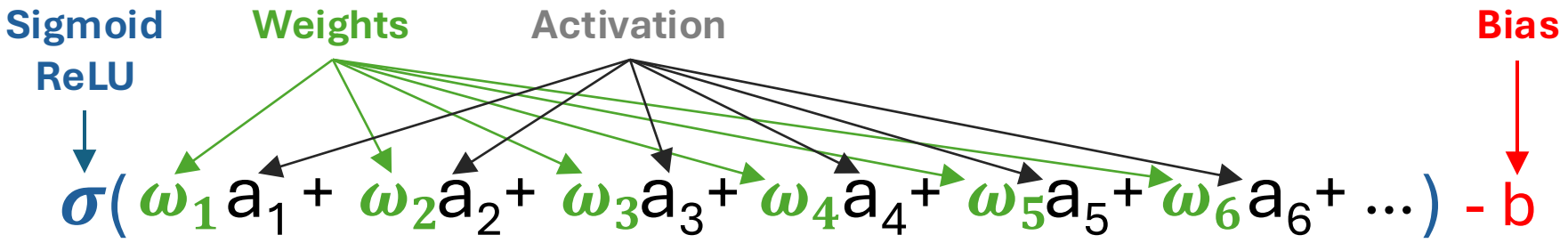
ReLU

$$f(x) = max(0, x)$$



Neural Network architectures

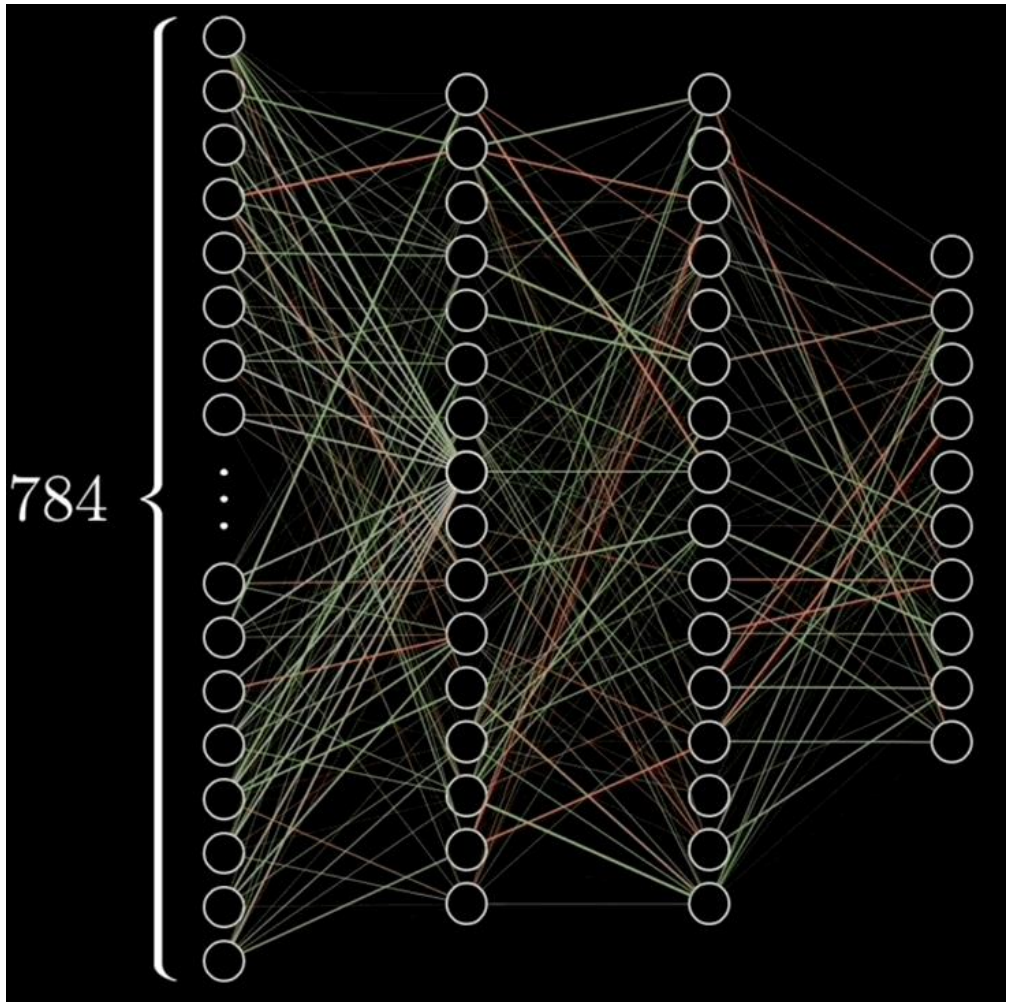




Each neuron is a function of 784 input activations

$784\ weights \times 16 \qquad 16\ bias$

Neural Network architectures



$$784 \times 16 + 16 \times 16 + 16 \times 10$$

Weights

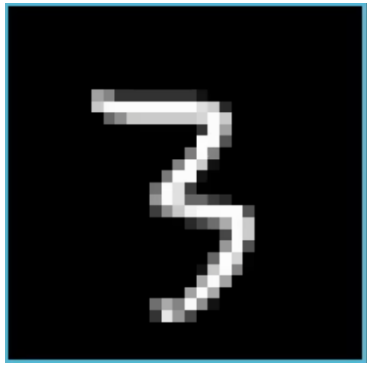
$$16 + 16 + 10$$

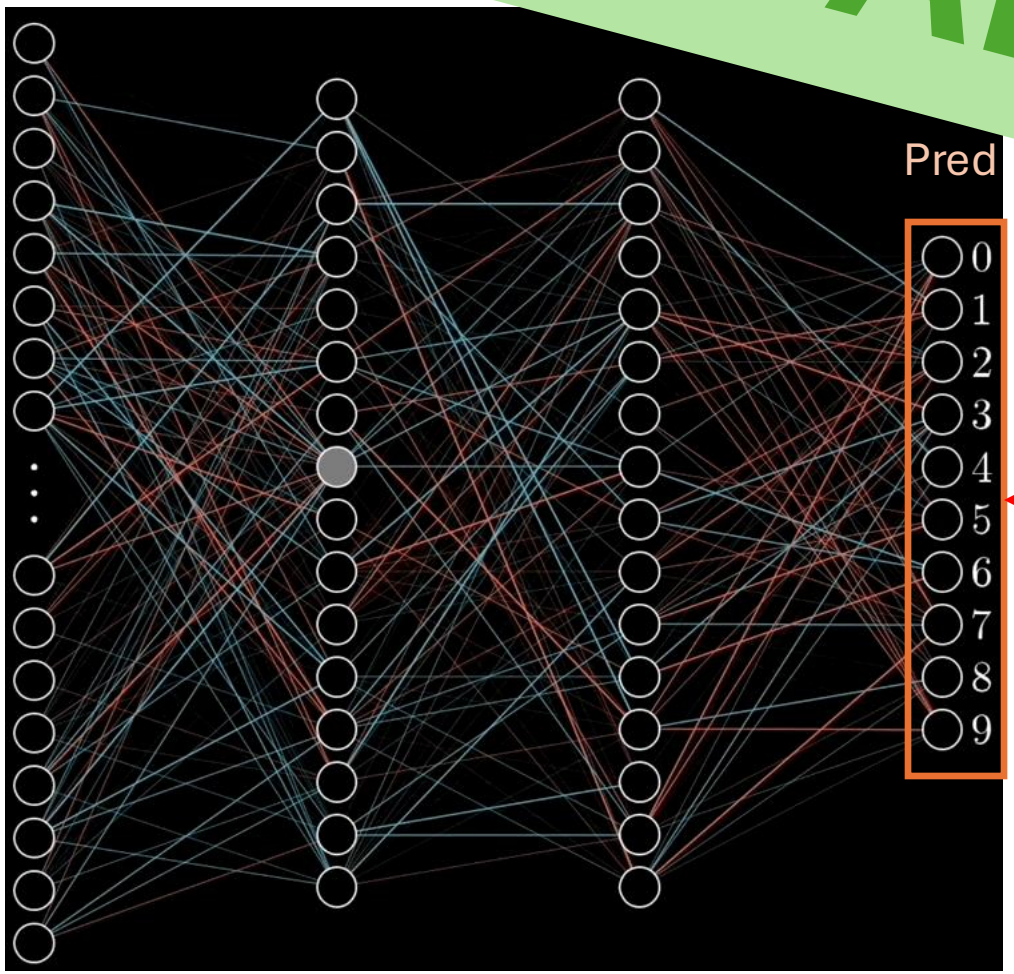
Biases

13 002 learnable parameters

Network architectures

FORWARD PASS





This is wh… …nes to

Cost function calculate the discrepancy between the network's predicted output and the true expected output

Neural Network architectures

Friendly reminder

$$\text{Loss } (\mathcal{L}) = \text{Measure of difference}(\text{Prediction, True Value})$$

The model need to adjust the weights and biases to make better prediction. This is performed by the calculation of Gradient. The **Gradient** is the actionable mathematical information derived directly from the Loss function.

$$\text{Gradient} = \nabla\mathcal{L} = \left(\frac{\partial\mathcal{L}}{\partial W_1}, \frac{\partial\mathcal{L}}{\partial W_2}, \dots\right)$$

The **sign** of the partial derivative (positive/negative) tells the **direction** to adjust the weight to decrease the loss.
The **magnitude** of the partial derivative tells the **sensitivity** (how big a step you should take).

Backpropagation

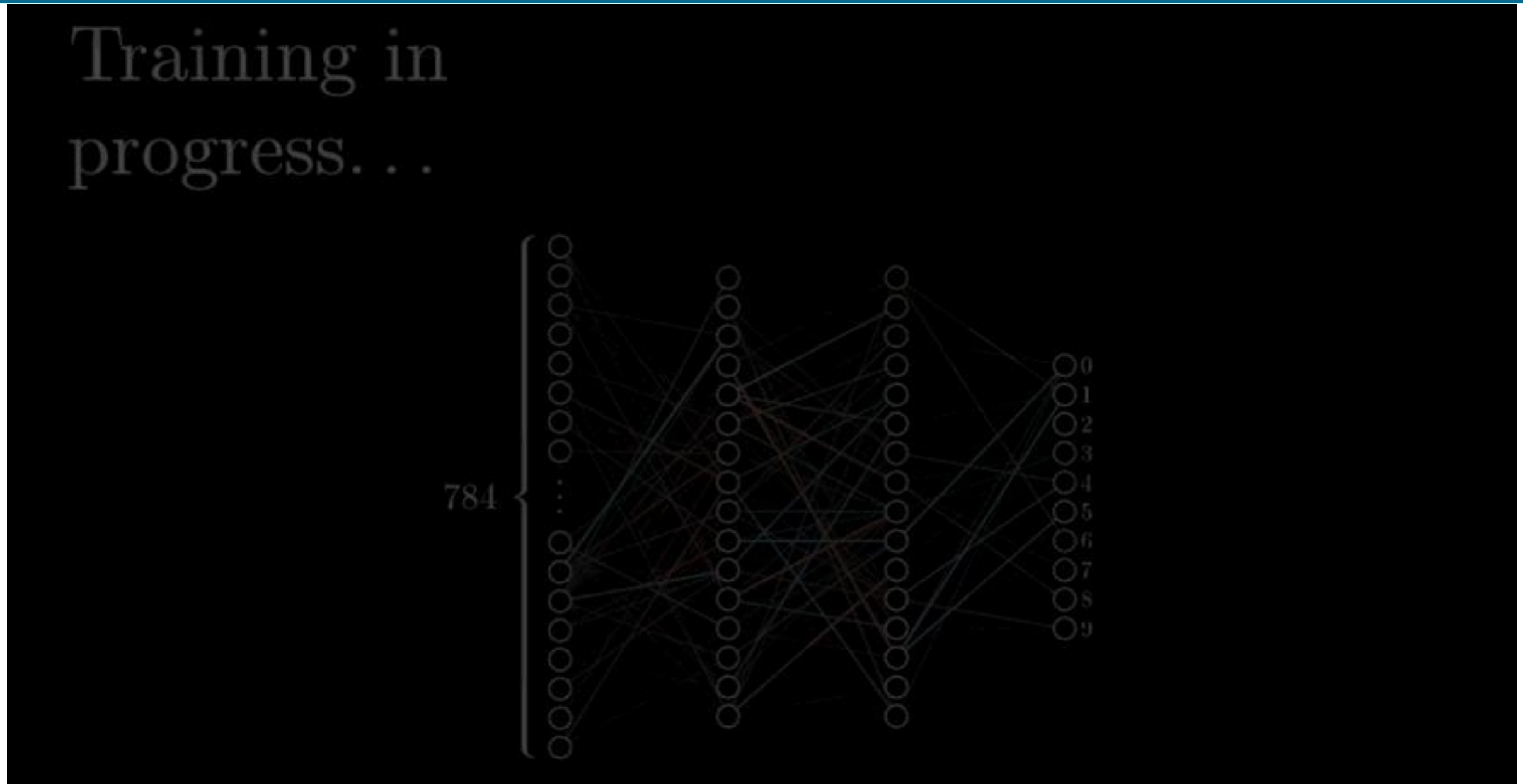
Backpropagation is the **calculator** for the Gradient. It applies the chain rule of calculus starting from the Loss (output) and works backward, layer by layer, to compute for *every* parameter

$$\frac{\partial\mathcal{L}}{\partial W_{\text{Layer } i}} = \frac{\partial\mathcal{L}}{\partial \text{Output}} \times \frac{\partial \text{Output}}{\partial \text{Layer } i} \times \cdots \times \frac{\partial \text{Layer } i}{\partial W_{\text{Layer } i}}$$

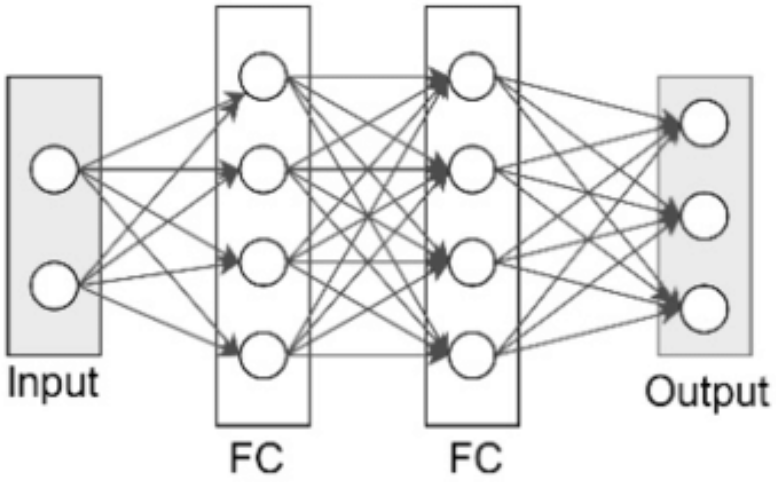
Once Backpropagation provides the complete Gradient vector, an optimizer uses it to execute the necessary parameter update

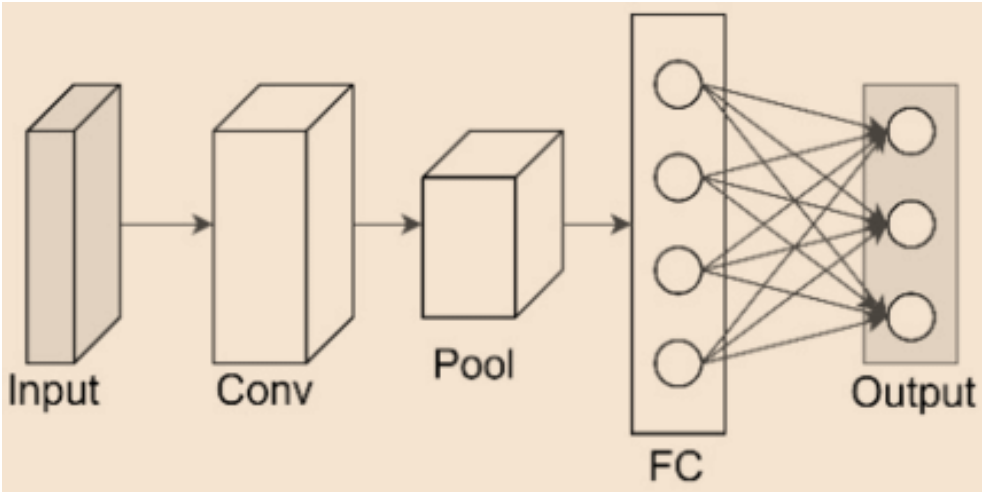
$$\text{New } W = \text{Old } W - \text{Learning Rate} \times \nabla\mathcal{L}$$

Neural Network architectures



HSLU Hochschule Luzern Neural Network architectures





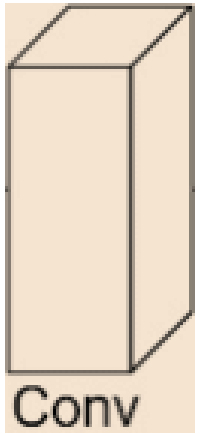
Aspect	Neural Networks (NN)	Convolutional Neural Networks (CNN)
Purpose	General ML tasks (regression, classification)	Image and spatial data processing
Input	Tabular data, vectors	Grid-like data (images)
Architecture	Fully connected layers	Convolutional + pooling layers
Feature Extraction	Manual preprocessing	Automatic via convolutional filters
Efficiency	High parameter count, less efficient	Fewer parameters, efficient for large inputs
Use Cases	Stock prediction, tabular data	Image recognition, object detection

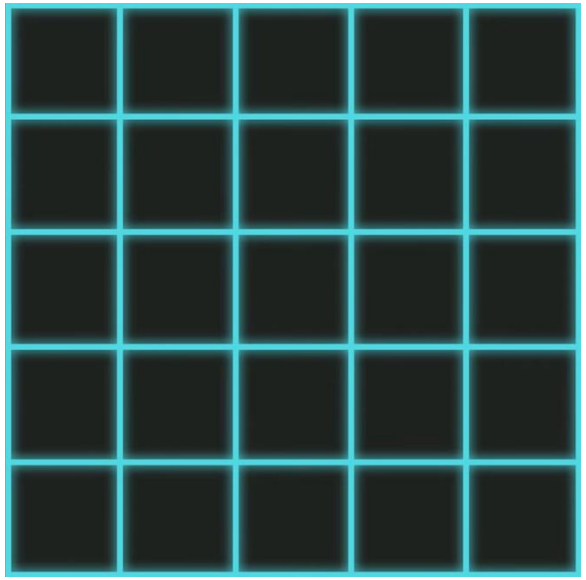
Neural Network architectures

What is Convolution:

The idea of convolution is to define a Kernel (Feature detector / Filter).

The kernel is a matrix of a defined size that moves around in the image to predict features relative to areas

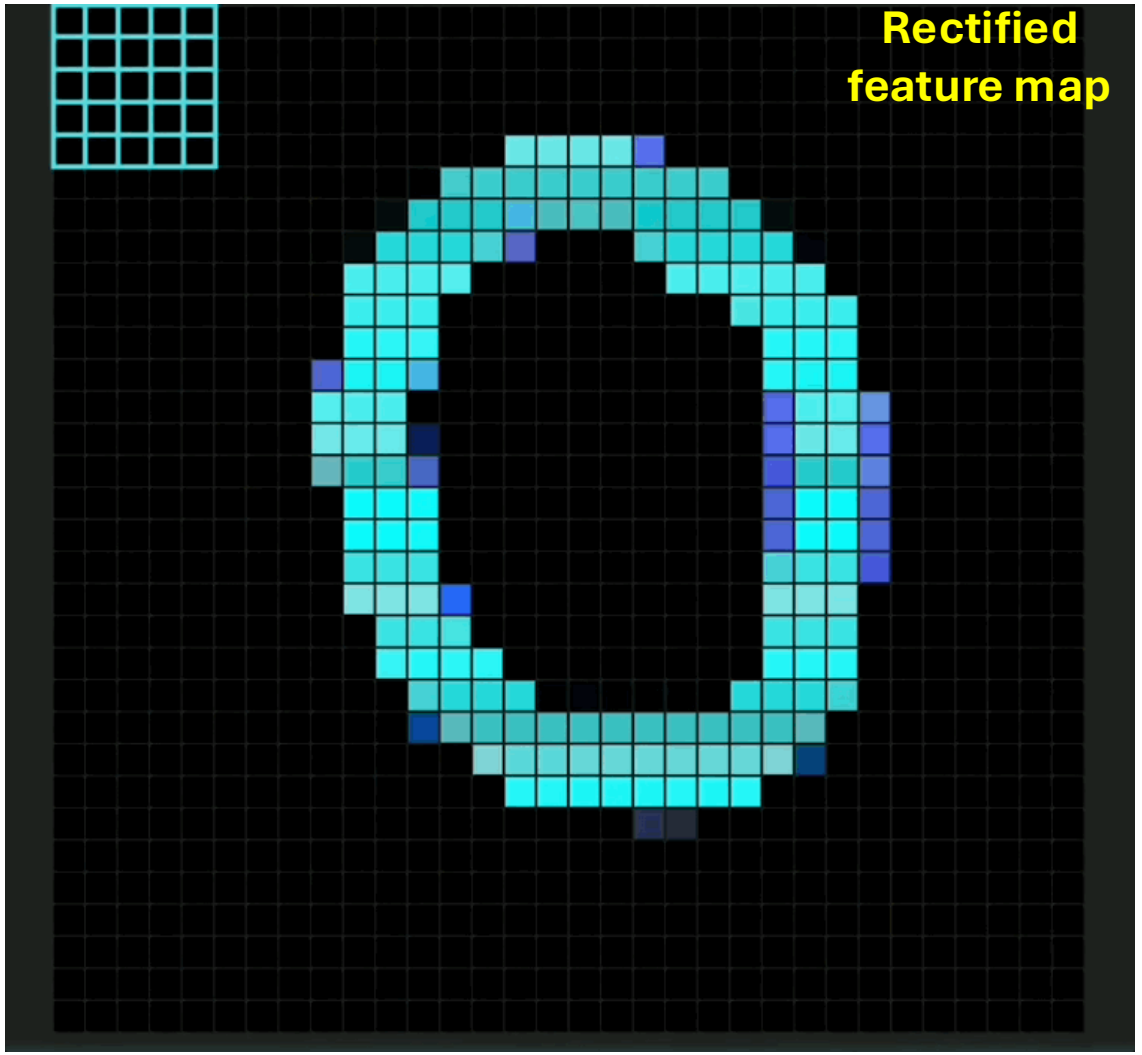


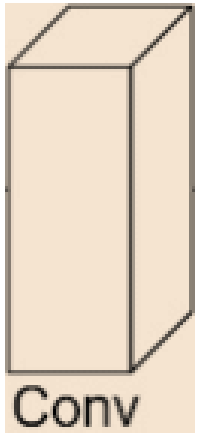


Neural Network architectures

What is Convolution:

Convolutional layers efficiently capture local spatial patterns and hierarchies in data like images through parameter sharing and local connectivity, reducing the number of parameters compared to fully connected layers while enabling translation invariance





For a 2D input feature map $\mathbf{X} \in \mathbb{R}^{H \times W \times C_{in}}$ and kernel $\mathbf{K} \in \mathbb{R}^{k_h \times k_w \times C_{in} \times C_{out}}$,the output feature map $\mathbf{Y}$ at position (i, j) for channel c is:

$$Y_{i,j,c} = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} \sum_{p=1}^{C_{in}} X_{i+m,j+n,p} \cdot K_{m,n,p,c} + b_c$$

Neural Network architectures

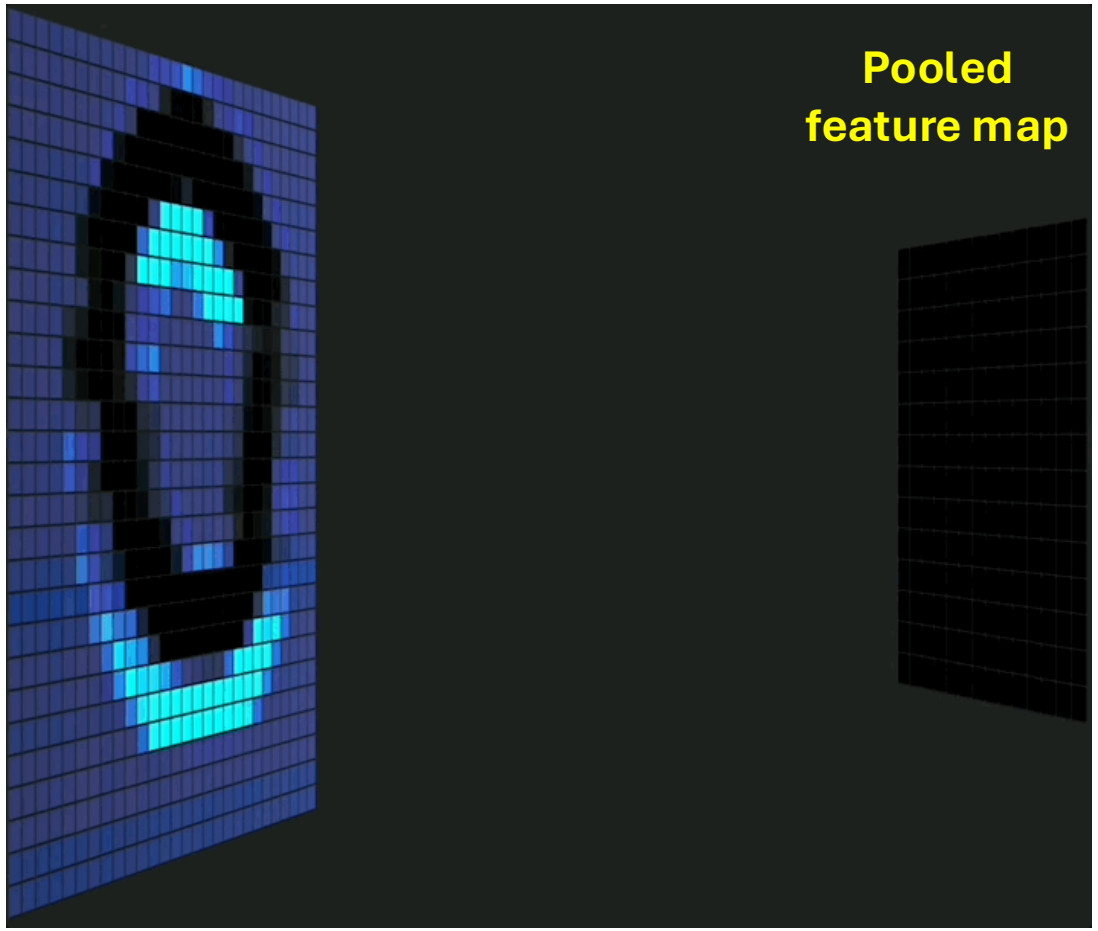
What is Pooling:

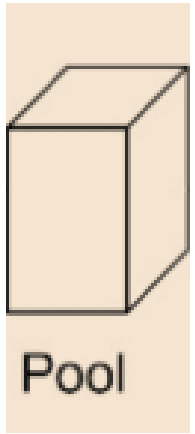
Pooling layers downsample feature maps to reduce computational complexity and spatial dimensions, enhancing translation invariance and helping prevent overfitting by summarizing essential features

For a 2D input *Rectified feature map* $\mathbf{Y} \in \mathbb{R}^{H' \times W' \times C}$, pooling window size $k_h \times k_w$ (2×2), and stride s such as 2 for non-overlapping), the output *Pooled feature map* $\mathbf{P}$ at position (i, j) for channel c

$$P_{i,j,c} = \max_{m=0}^{k_h-1} \max_{n=0}^{k_w-1} Y_{i \cdot s+m,\, j \cdot s+n,\, c}$$

No learnable parameters (unlike conv); it's a fixed operation for subsampling.





HSLU Hochschule Luzern

OpenCV

Hands-on:

Implementation of a Neural Networks

Matplotlib

HSLU Hochschule Luzern

Pretrained models

Approach	Description	Data Requirements	Compute/Resources	Performance Benefits	Applicability to End Tasks
Training from Scratch	Building and training a neural network entirely on your target dataset, initializing weights randomly (e.g., your ANN/CNN on MNIST). No prior knowledge used.	High: Needs large, diverse labeled data (e.g., millions of examples) to learn features effectively; small datasets like MNIST work but limit generalization.	High: Full training epochs required, often needing GPUs/TPUs for days/weeks on big models.	Baseline performance; good for simple tasks but prone to overfitting on small data and poor generalization to complex scenarios.	Versatile for all (classification, segmentation, detection), but inefficient for vision tasks needing hierarchical features (e.g., edges → objects). Best for custom architectures on abundant data.
Transfer Learning	Broad technique reusing a model trained on a source task/dataset (e.g., ImageNet) for a target task. Includes feature extraction (freeze base, add/train new layers) or full adaptation.	Low to Medium: Leverages pretrained features, so works well with smaller target datasets (e.g., 1K–10K samples vs. millions).	Medium: Less training time (e.g., hours vs. days); can freeze layers to reduce compute.	High: Transfers general features (e.g., edges/textures), boosting accuracy (e.g., +5–20% on small data) and convergence speed. Reduces overfitting.	Excellent for all vision tasks; foundational for adapting to classification (e.g., fine-tune ResNet), segmentation (e.g., U-Net variants), detection (e.g., YOLO with backbone).
Finetuning	A subset of transfer learning: Take a pretrained model and retrain (tune) all or select layers on your dataset, often with lower learning rates to preserve learned features.	Low: Ideal for domain shifts (e.g., medical images from ImageNet-pretrained model); 1K+ samples suffice.	Medium to High: Trains more parameters than feature extraction but faster than scratch; monitor for catastrophic forgetting.	Very High: Adapts to task specifics while retaining general knowledge; often state-of-the-art (e.g., 95%+ on CIFAR-10).	Broadly applicable; common for classification (e.g., adjust final layers), segmentation (e.g., DeepLab finetune), detection (e.g., Faster R-CNN backbone). Use for tasks with similar domains.
Models from the Hub	Pre-built, pretrained models loaded directly (e.g., torch.hub.load('pytorch/vision', 'resnet50', pretrained=True)). Ready for immediate use/adaptation.	Low: No need to train base; just your task data for finetuning/extraction.	Low to Medium: Download once, then quick setup; community-vetted for efficiency.	High: Proven on benchmarks (e.g., ImageNet top-1 ~80%); plug-and-play for quick prototypes.	Task-specific variants available: Classification (ResNet/ViT), Segmentation (DeepLab/FCN), Detection (Faster R-CNN/RetinaNet), plus keypoints/video. Great for rapid experimentation across tasks.

HSLU Hochschule Luzern

Pretrained models

Dataset	Primary Task(s)	Number of Images	Size (approx.)	Description	Reference
ImageNet-1K (ILSVRC 2012)	Image Classification	1 431 167 (1.28M train, 50K val, 100K test)	~150 GB	Foundational dataset with 1,000 diverse object categories; widely used for pretraining models like ResNet and VGG to learn general visual features.	ImageNet
ImageNet-21K	Image Classification	~14.2 million	~1.3 TB (raw; ~250 GB processed)	Larger variant with 21,841 categories; enables broader pretraining for better generalization in transfer learning.	ImageNet; HuggingFace
Open Images V7	Image Classification, Object Detection	~9 million (1.74M train subset for detection)	~561 GB (for train+val subset)	Google's large-scale dataset with image-level labels, bounding boxes, and visual relationships across 600+ classes; supports multi-task pretraining.	Open Images GitHub; Google Cloud
COCO (Common Objects in Context)	Object Detection, Instance Segmentation, Keypoint Detection, Captioning	330 000 (>200K labeled)	~20 GB	Comprehensive dataset with 80 object categories, 1.5M instances, and dense annotations (bounding boxes, masks, keypoints); benchmark for detection/segmentation models like Mask R-CNN.	COCO Dataset
Pascal VOC (2012)	Object Detection, Semantic Segmentation	~21,738 (11.5K trainval, 10.2K test)	~11 GB	Classic dataset with 20 object classes; provides bounding boxes and pixel-level segmentation; often used for evaluation and smaller-scale pretraining.	Pascal VOC; GluonCV Guide
Objects365	Object Detection	2 million	~500+ GB (metadata ~1.65 GB)	Massive dataset with 365 categories and 30M bounding boxes; designed for fine-grained object detection in the wild, improving robustness in pretrained detectors.	Objects365 Site; HuggingFace
ADE20K	Semantic Segmentation	~27 500 (20K train, 2K val, 3K test)	~15 GB (estimated)	Scene parsing dataset with 150 semantic categories (objects + stuff); annotated at pixel level for holistic scene understanding; used in models like DeepLab.	ADE20K Site; GitHub

HSLU Hochschule Luzern

OpenCV

Hands-on:

Using Pytorch Hub pretrained models

Matplotlib

HSLU Hochschule Luzern

Additional Frameworks

Framework	Description	Supported Tasks	Key Models
ultralytics	Open-source ecosystem for efficient, real-time YOLO-based vision AI.	Object detection, instance segmentation, pose estimation, classification	YOLOv8, YOLOv10, YOLOv11, SAM2, RT-DETR
Detectron2	Facebook's modular platform for advanced visual recognition tasks.	Object detection, instance segmentation, panoptic segmentation	Mask R-CNN, Cascade R-CNN, PointRend, ViTDet
MMDetection	OpenMMLab's comprehensive toolbox for detection benchmarks and research.	Object detection, instance segmentation, panoptic segmentation, rotated detection	RTMDet, Grounding DINO

AGENDA. DAY 2

- Pre-requisits
- Introduction to Generative AI
- Autoencoder
  - Introduction to autoencoders
  - Path to Variational Autoencoders
- Part 2.
  - From Adversarial Training to GANs
  - GAN's Architecture
  - GAN's objective
  - DCGANs
- Part 3.
  - From scratch hand-on implementation of DCGAN
  - Implementation of Conditional GAN

HSLU Hochschule Luzern

Pre-requisits

Mathematics and Statistics	Programming	ML Fundamentals
Linear Algebra Probability & Statistics Optimization (gradient descent, loss functions)	Python NumPy, Pandas, Matplotlib DL Frameworks (Pytorch, Keras, Tensorflow)	supervised/unsupervised learning, overfitting, evaluation metrics like accuracy/F1 feedforward, backpropagation CNNs

Generative AI in Vision

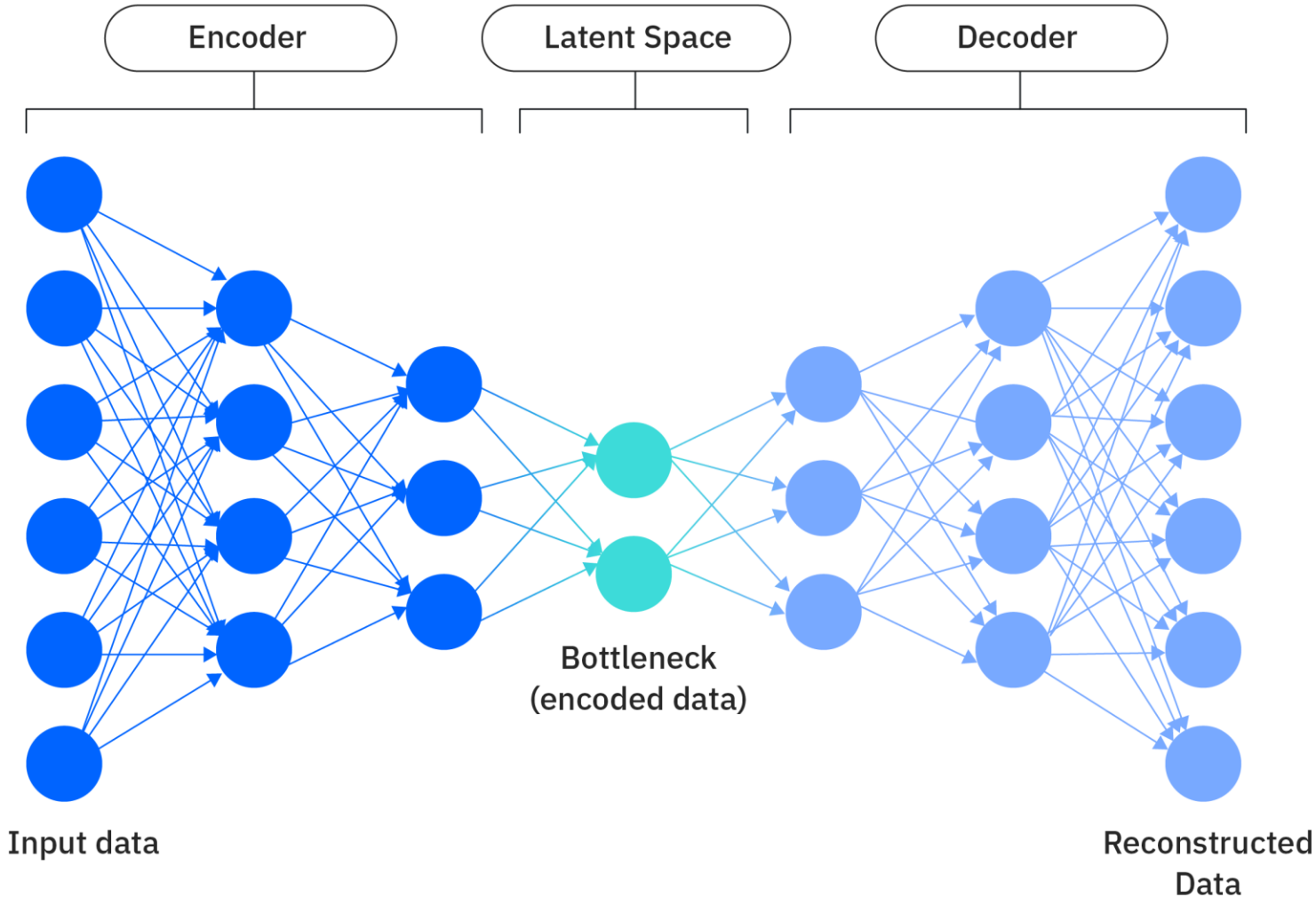
Generative AI :

transforming image synthesis, Generating image captions, enabling the creation of high-quality, diverse, and photorealistic visuals across industries like design, media, healthcare, and autonomous systems. Applications:

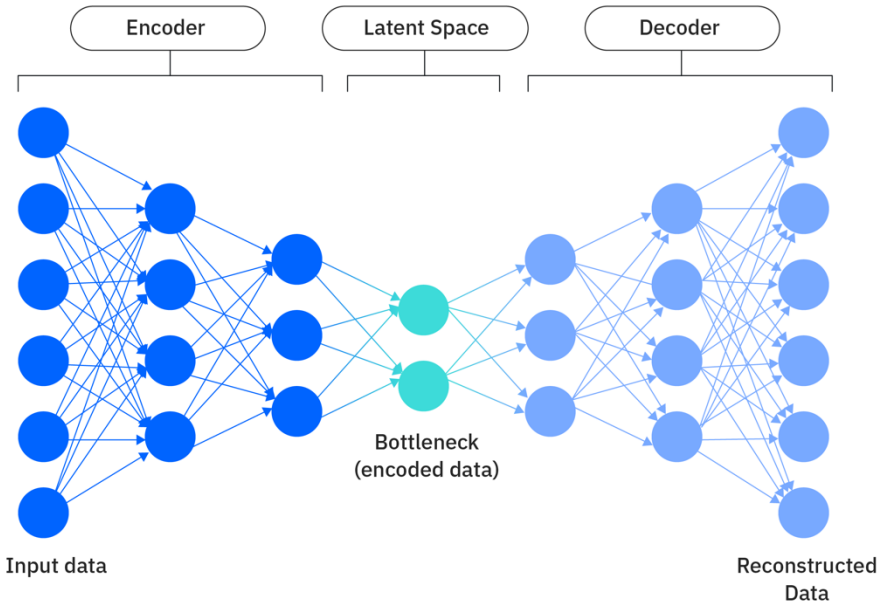
- image-to-image translation
- text-to-image generation,
- domain transfer
- multimodal alignment
- Image to text generation

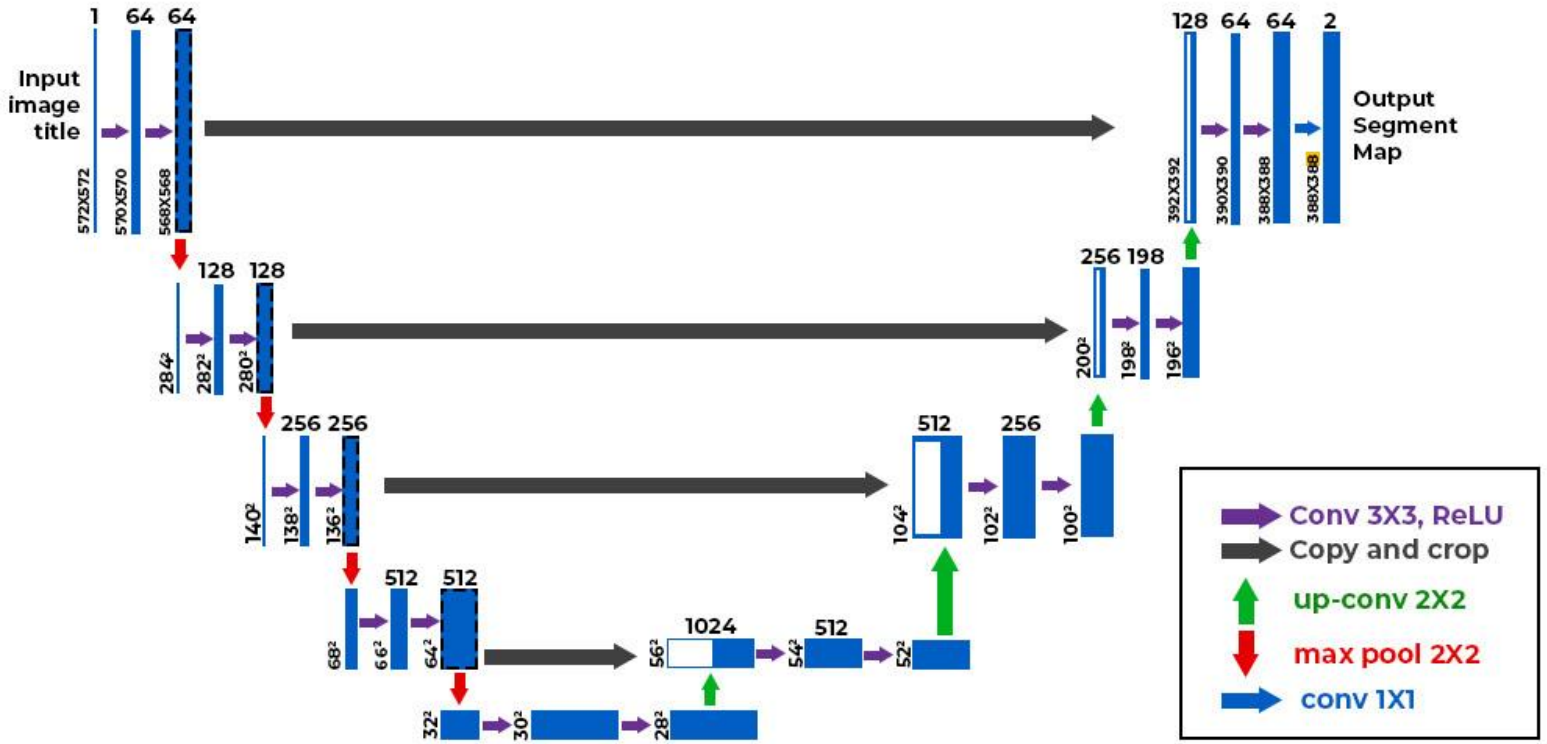
These advancements are driven by models like GANs, conditional frameworks, and diffusion-based approaches

Autoencoder

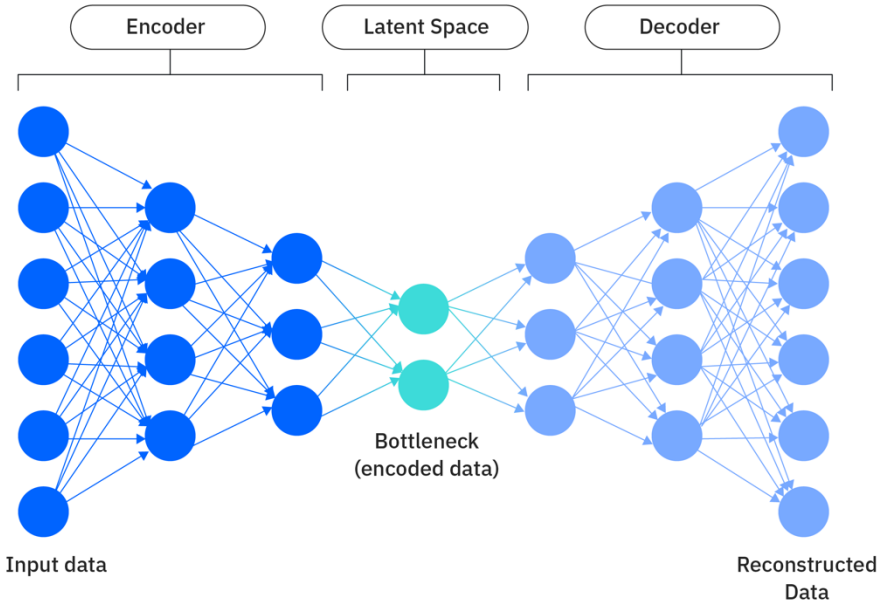


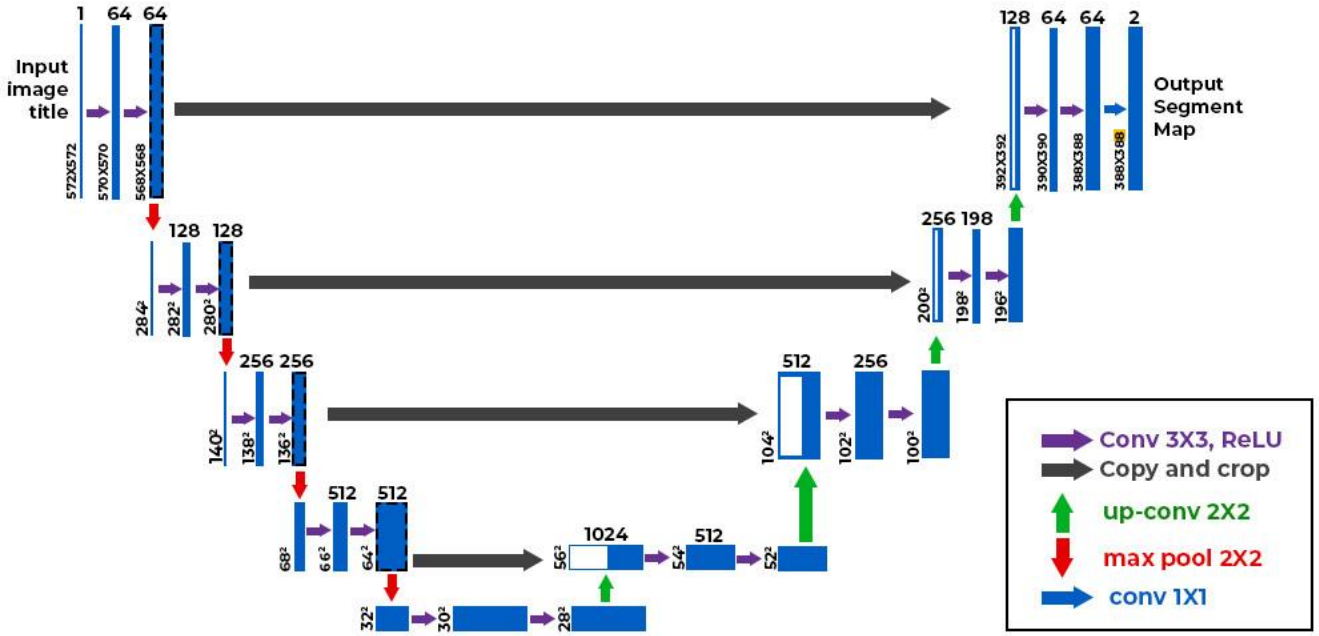
Autoencoder

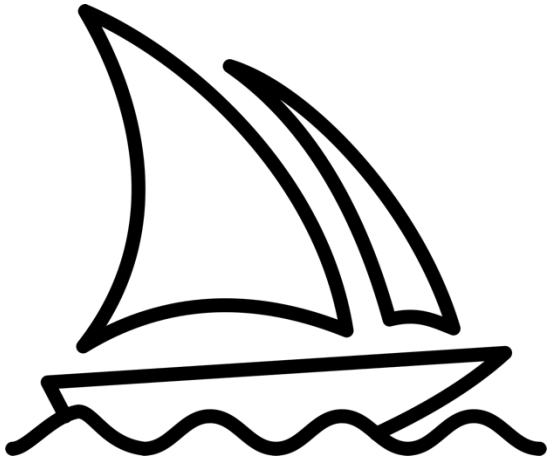




Autoencoder

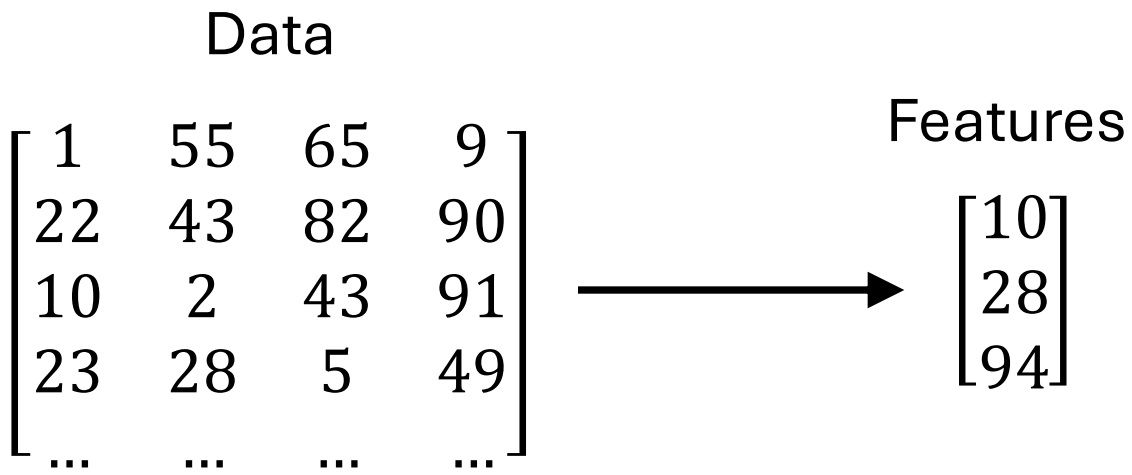






Autoencoder

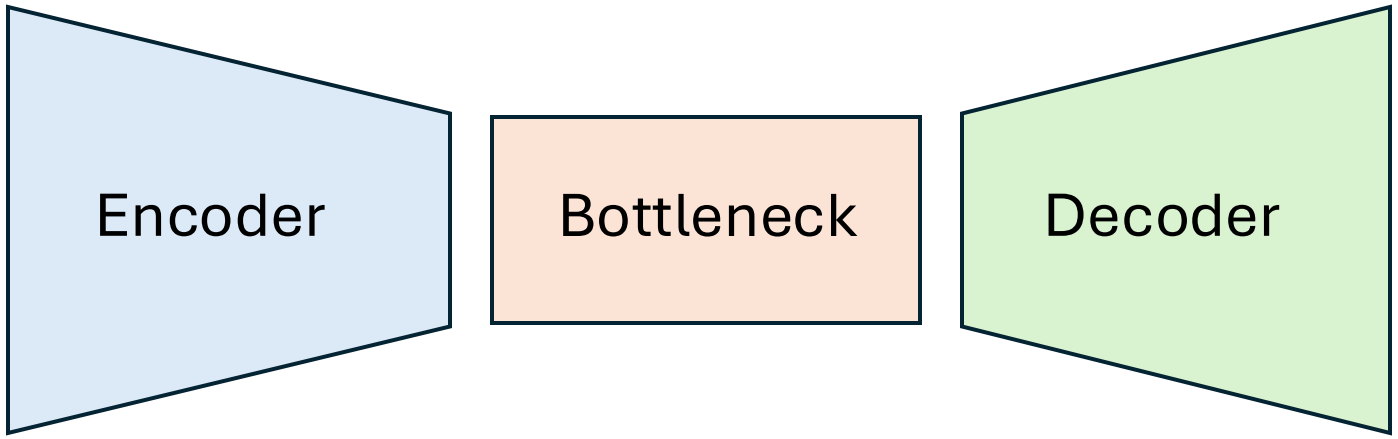
An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. The aim is to first learn encoded representations of our data and then generate the input data (as closely as possible) from the learned encoded representations. Thus, the output of an autoencoder is its prediction for the input.



$$\begin{bmatrix} 1 & 55 & 65 & 9 \\ 22 & 43 & 82 & 90 \\ 10 & 2 & 43 & 91 \\ 23 & 28 & 5 & 49 \\ \dots & \dots & \dots & \dots \end{bmatrix} \rightarrow \begin{bmatrix} 10 \\ 28 \\ 94 \end{bmatrix}$$

Autoencoder

An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. The aim is to first learn encoded representations of our data and then generate the input data (as closely as possible) from the learned encoded representations. Thus, the output of an autoencoder is its prediction for the input.



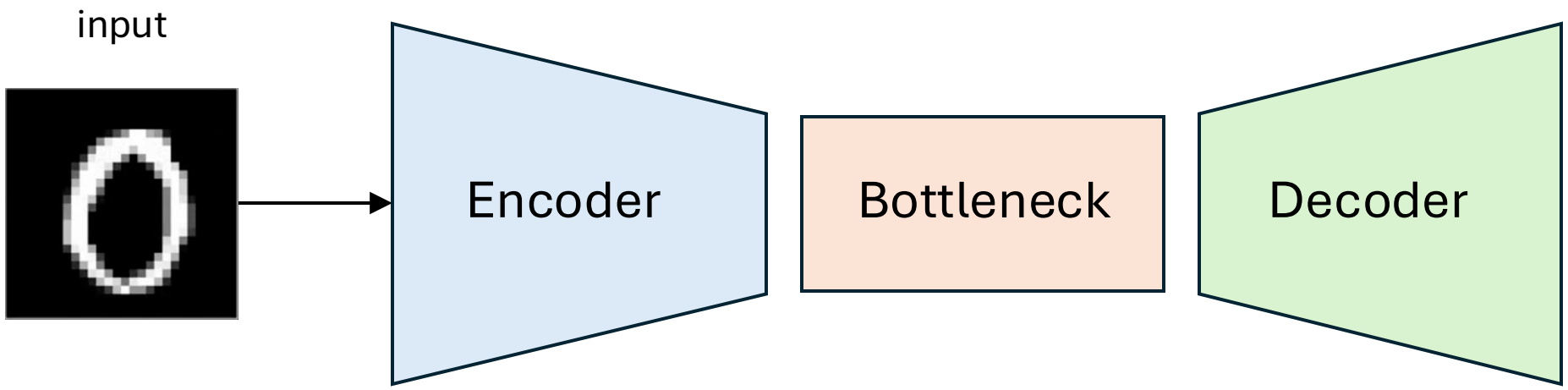
Compressor of the input data into a latent space representation.

Hold compressed data

Reconstruct input data produced by the Encoder and bottleneck

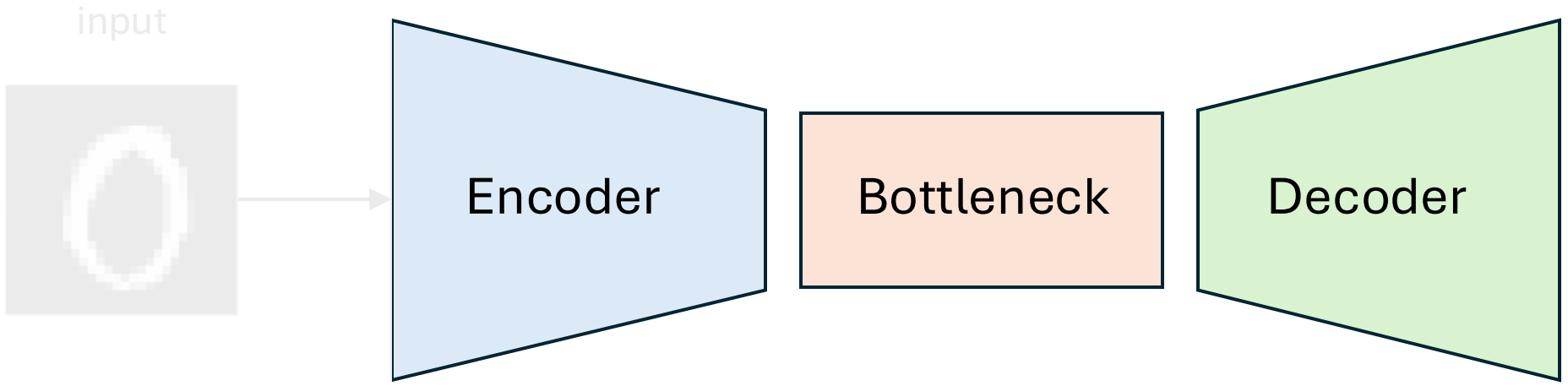
Autoencoder

An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. The aim is to first learn encoded representations of our data and then generate the input data (as closely as possible) from the learned encoded representations. Thus, the output of an autoencoder is its prediction for the input.



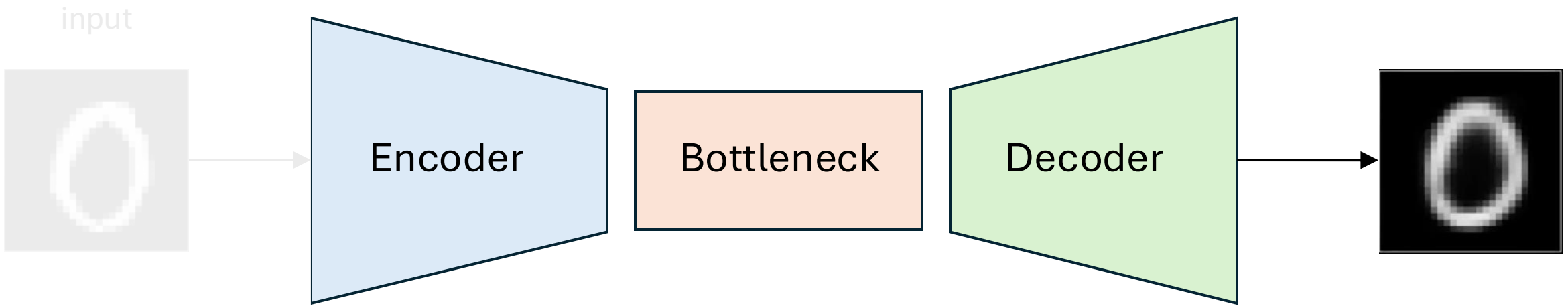
Autoencoder

An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. The aim is to first learn encoded representations of our data and then generate the input data (as closely as possible) from the learned encoded representations. Thus, the output of an autoencoder is its prediction for the input.



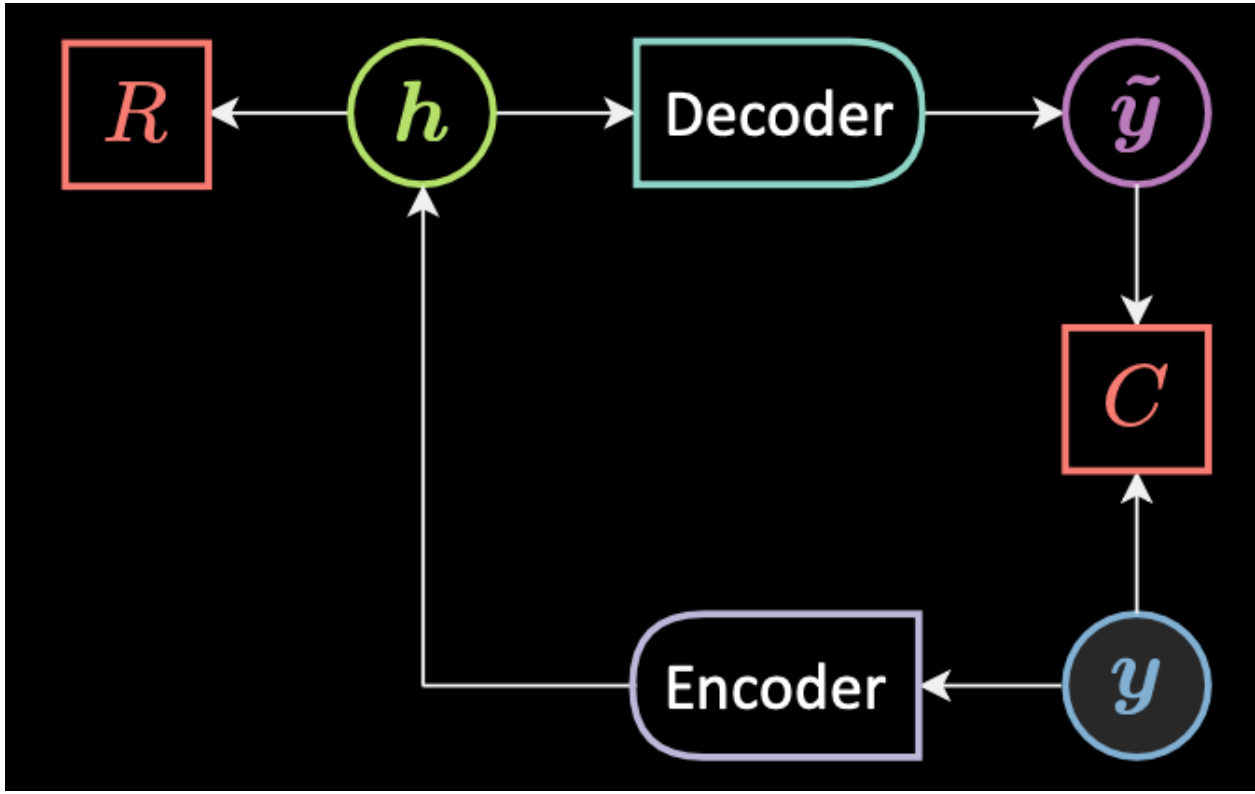
Autoencoder

An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. The aim is to first learn encoded representations of our data and then generate the input data (as closely as possible) from the learned encoded representations. Thus, the output of an autoencoder is its prediction for the input.



Autoencoder

- **Input (C):** Source image (RGB pixels in $\mathbb{R}^{H\times W\times 3}$)
- **Encoder:** Extracts hierarchical features (edges → textures → objects) into latent code y.
- **Hidden State (h):** Aggregated context (e.g., bottleneck features) bridging encoder-decoder.
- **Reference (R):** Ground-truth target
- **Decoder:** Upsamples/generates $\tilde{y}$ (reconstructed image) conditioned on h and R.



Autoencoder

• First, the autoencoder takes in an input and maps it to a hidden state through an affine transformation
$\boldsymbol{h} = f(\boldsymbol{W}_h \boldsymbol{y} + \boldsymbol{b}_h)$,where f is an (element-wise) activation function. This is the **encoder** stage. Note that $\boldsymbol{h}$ is also called the **code**.

• Next, $\tilde{\boldsymbol{y}} = g(\boldsymbol{W}_y \boldsymbol{y} + \boldsymbol{b}_y)$,where g is an activation function. This is the **decoder** stage.

• Finally, the energy is the sum of the reconstruction and the regularization, $F(\boldsymbol{y}) = C(\boldsymbol{y}, \tilde{\boldsymbol{y}}) + R(\boldsymbol{h})$.

Autoencoder

Reconstruction costs

When the input is real-valued we use mean squared error loss,

$$C(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \| \boldsymbol{y} - \tilde{\boldsymbol{y}} \|^2 = \| \boldsymbol{y} - Dec[Enc(\boldsymbol{y})] \|^2$$

When the input is categorical we use cross entropy loss,

$$C(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = -\sum_{i=1}^{n} \boldsymbol{y_i} \, \boldsymbol{log}(\tilde{\boldsymbol{y}}_{\boldsymbol{i}}) + (\boldsymbol{1} - \boldsymbol{y_i}) \, \boldsymbol{log}(\boldsymbol{1} - \tilde{\boldsymbol{y}}_{\boldsymbol{i}})$$

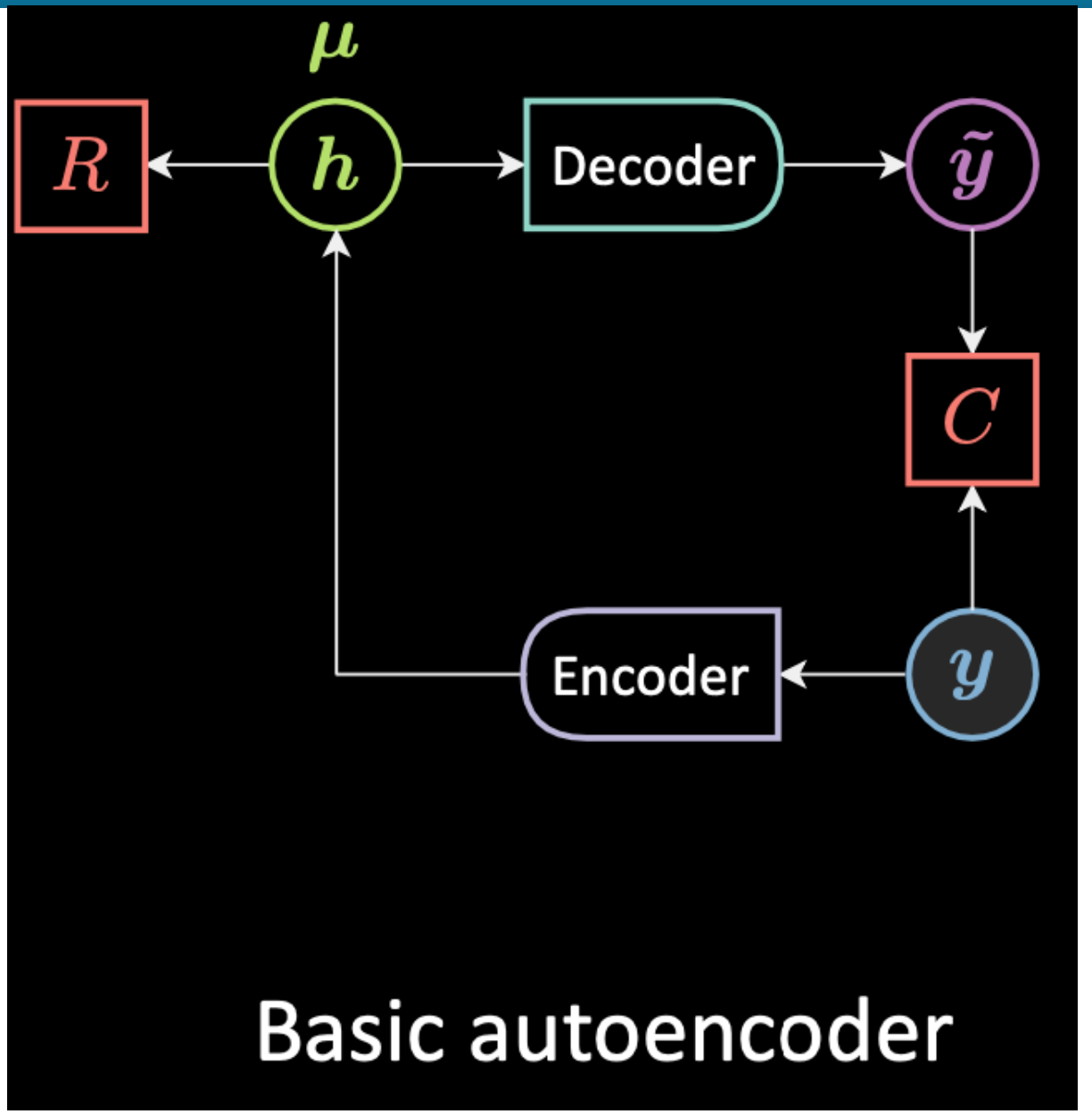
Loss functional

The loss functional is the average of the per sample loss.

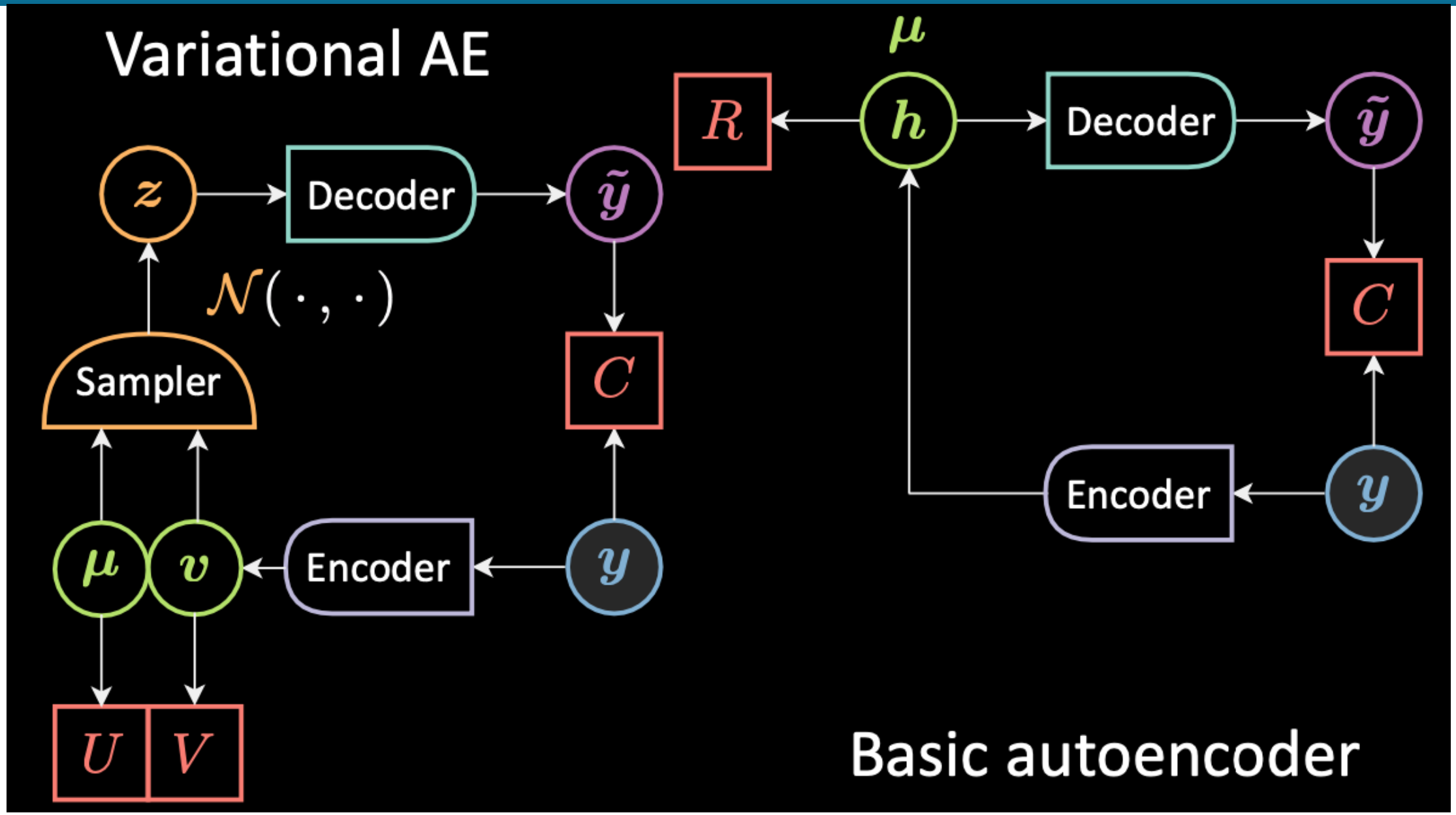
$$\mathcal{L}(F(\cdot), Y) = \frac{1}{m} \sum_{j=1}^{m} \ell\left(F(\cdot), y^{(j)}\right) \in \mathbb{R}$$

$$\ell_{energy}(F(\cdot), Y) = F(\boldsymbol{y})$$

Variational Autoencoder

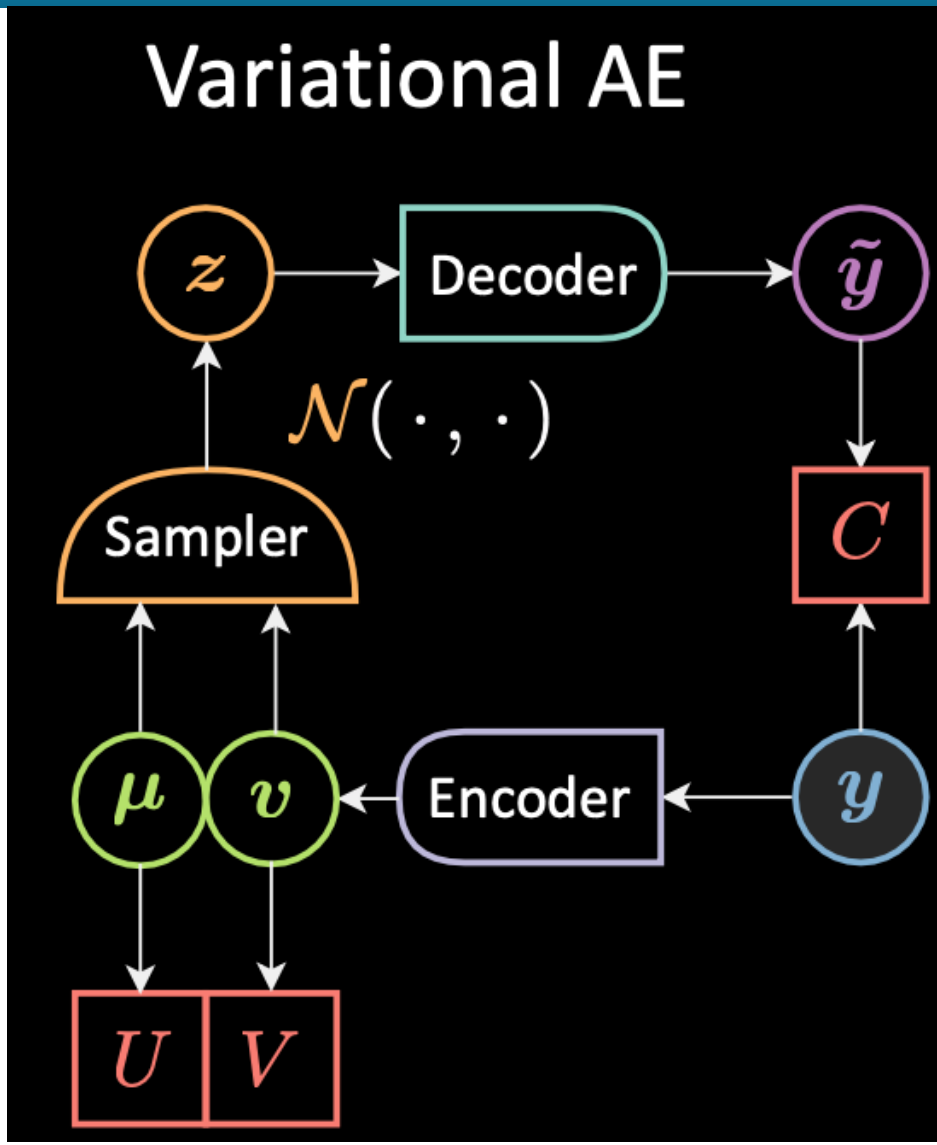


Variational Autoencoder



Variational Autoencoder

- First, the encoder stage: we pass the input $\boldsymbol{y}$ to the encoder. Instead of generating a hidden representation $\boldsymbol{h}$ in AE, the hidden representation in VAE comprises two parts: $\boldsymbol{\mu}$ and $\boldsymbol{v}$. And similar to using regularisation factor R for $\boldsymbol{h}$, we now use regularisation factors U and V for $\boldsymbol{\mu}$ and $\boldsymbol{v}$, respectively.
- Next, we use a sampler to sample $\boldsymbol{z}$ which is the latent random variable following a Gaussian distribution with $\boldsymbol{\mu}$ and $\boldsymbol{v}$. Note that people use Gaussian distributions as the encoded distribution in practice, but other distributions can be used as well.
- Finally, $\boldsymbol{z}$ is passed into the decoder to generate $\tilde{\boldsymbol{y}}$.
- The decoder will be a function from $\mathcal{Z}$ to $\mathbb{R}^n$:$\boldsymbol{z} \mapsto \tilde{\boldsymbol{y}}$.



HSLU Hochschule Luzern Variational Autoencoder

Input: Data sample C; here, C represents the observed data, such as image in $\mathbb{R}^n$.
Goal: Learn a low-dimensional latent space $\mathbb{Z} \subseteq \mathbb{R}^d$)with $d \ll n$) where data points are distributed according to a simple prior $p(z) = \mathcal{N}(0, I)$)standard Gaussian). This enables:

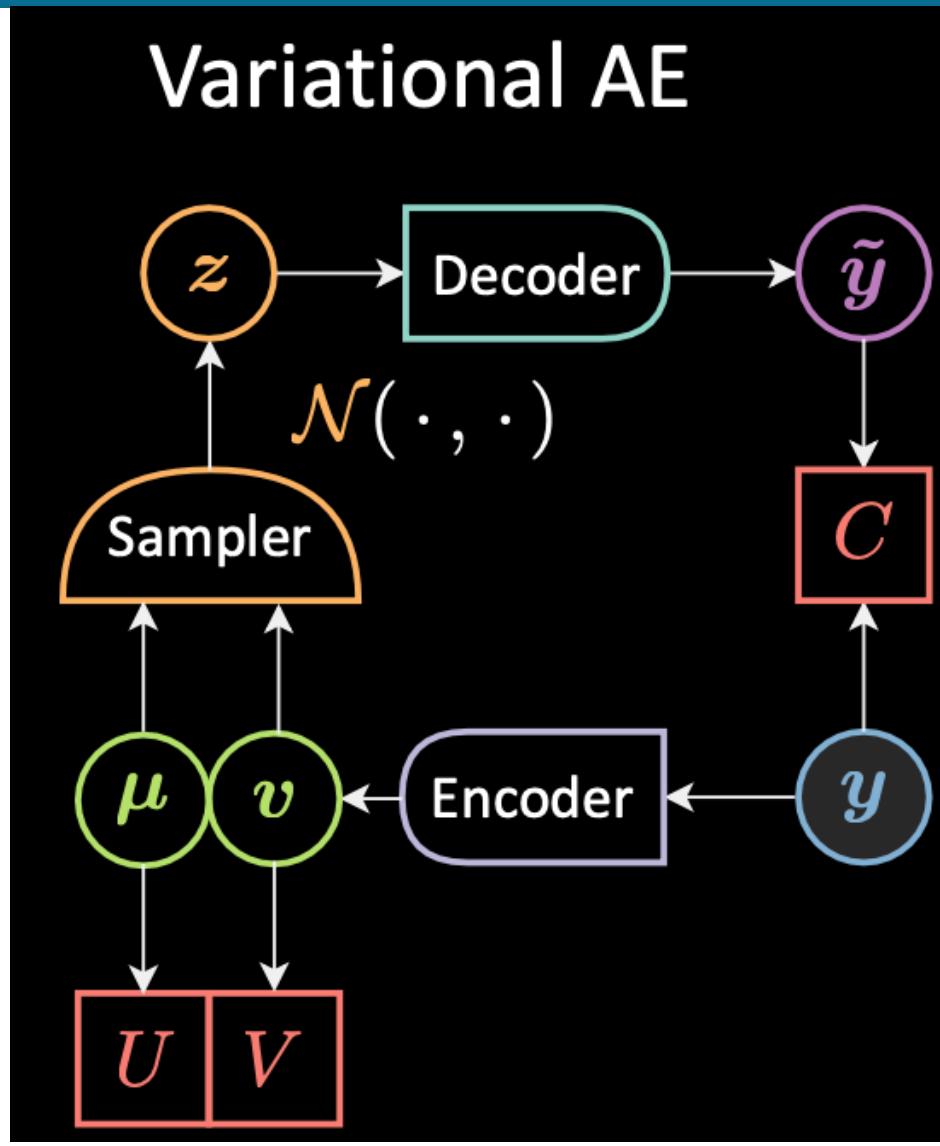
Reconstruction: $\tilde{y} \approx C$)denoising or compression).

Generation: Sample new $z \sim p(z)$,decode to novel $\tilde{y}$.

Training Objective: Maximize the Evidence Lower Bound (ELBO), a tractable proxy for the data log-likelihood:

$$\mathcal{L}(C; \theta; \phi) = \mathbb{E}_{q_\phi(z|C)}[\log p_\theta(\tilde{y} \mid z)] - D_{KL}\left(q_\phi(z \mid C) \parallel p(z)\right),$$

where $q_\phi(z \mid C)$is the approximate posterior (from encoder), $p_\theta(\tilde{y} \mid z)$is the decoder likelihood, $\theta$are decoder params, $\phi$are encoder params, and $D_{KL}$is Kullback-Leibler divergence (regularization term).



GAN Models

Generative
- Learn a generative model

Adversarial
- Trained in an adversarial setting

Networks
- Use Deep Neural Networks

} Unsupervised learning approach

GAN Models

Why generative ?

- Majority of models are discriminative :
  - Given an image X, predict a label Y
  - Estimates P(Y|X)
- Discriminative models have several key limitations:
  - Can’t model P(X), i.e. the probability of seeing a certain image
  - Thus, can’t sample from P(X), i.e. can’t generate new images
- Generative models can fill the gaps :
  - Can model P(X)
  - Can generate new images

GAN Models

Examples of GAN images

Ground truth / Real Image

Fake / Synthetic

GAN Models

Examples of GAN images

Ground truth / Real Image

Fake / Synthetic

GAN Models

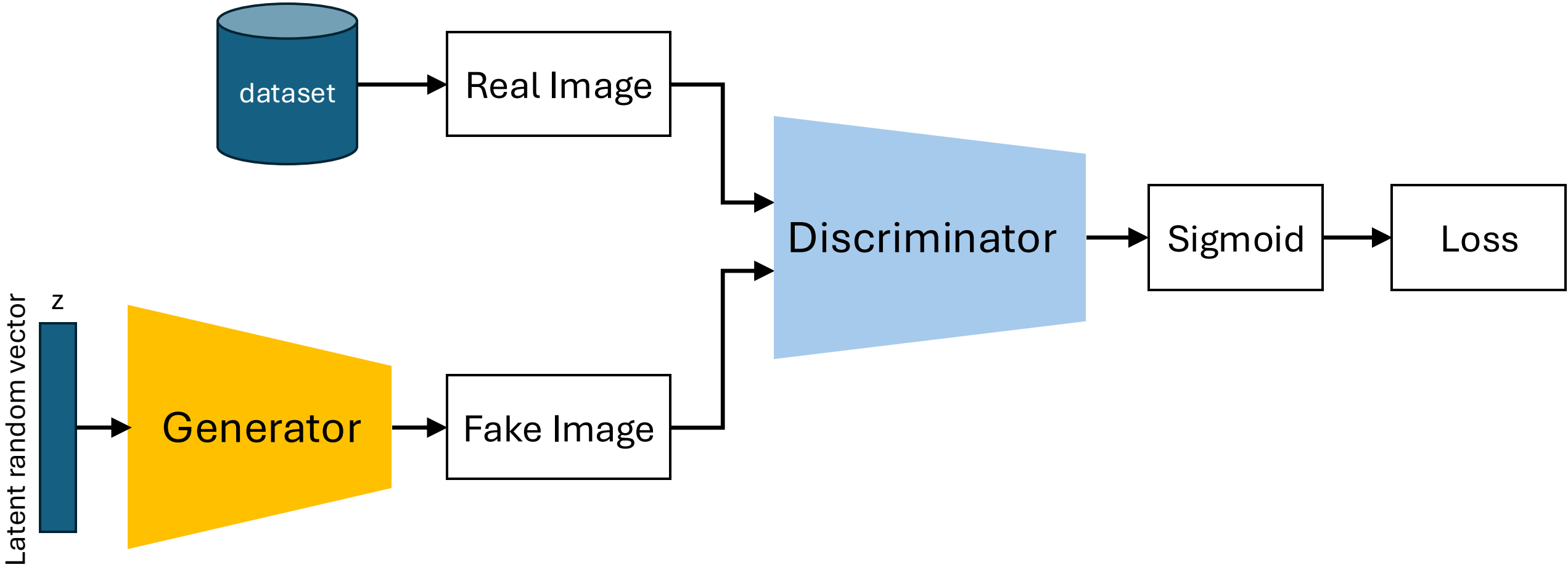
Examples of GAN images

Ground truth / Real Image

Fake / Synthetic

GAN Models

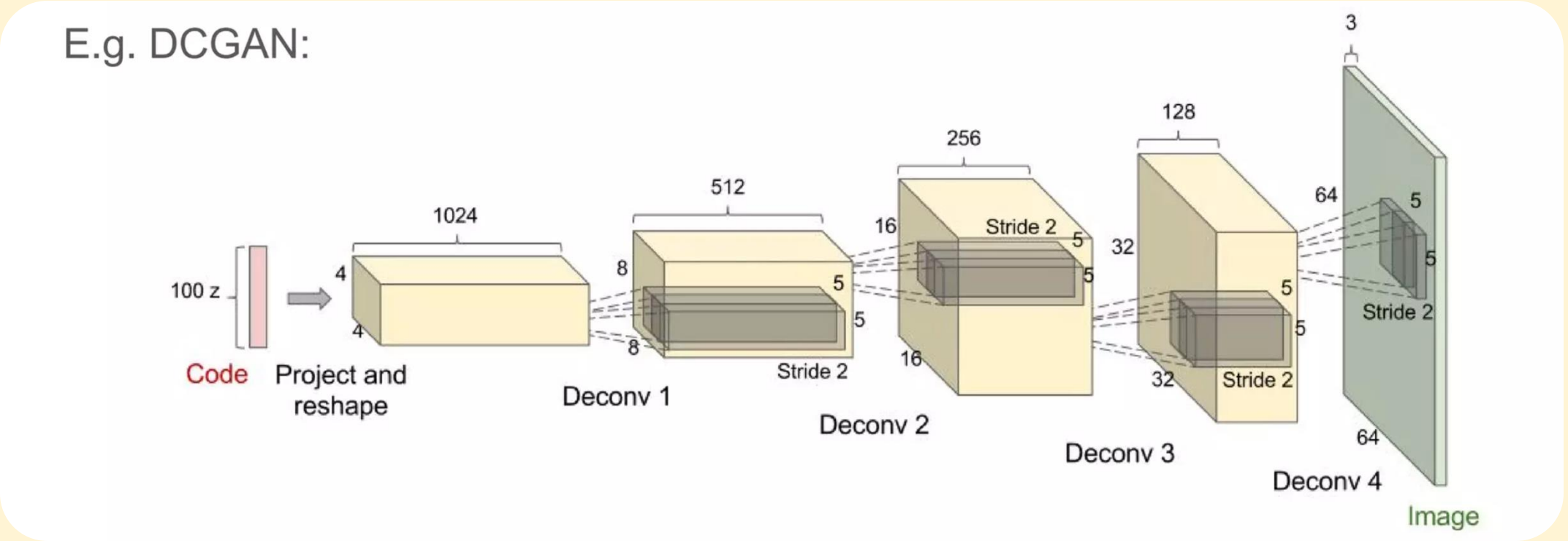
GAN architecture



GAN Models

Generator

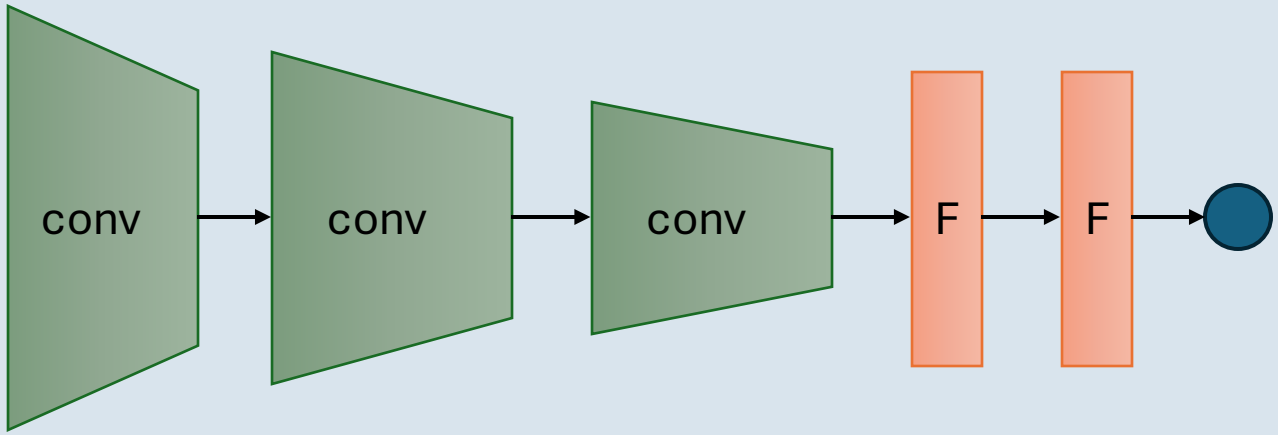
A deep neural network used for Deterministic mapping from a latent random vector to a sample q(x) ~ p(x)



GAN Models

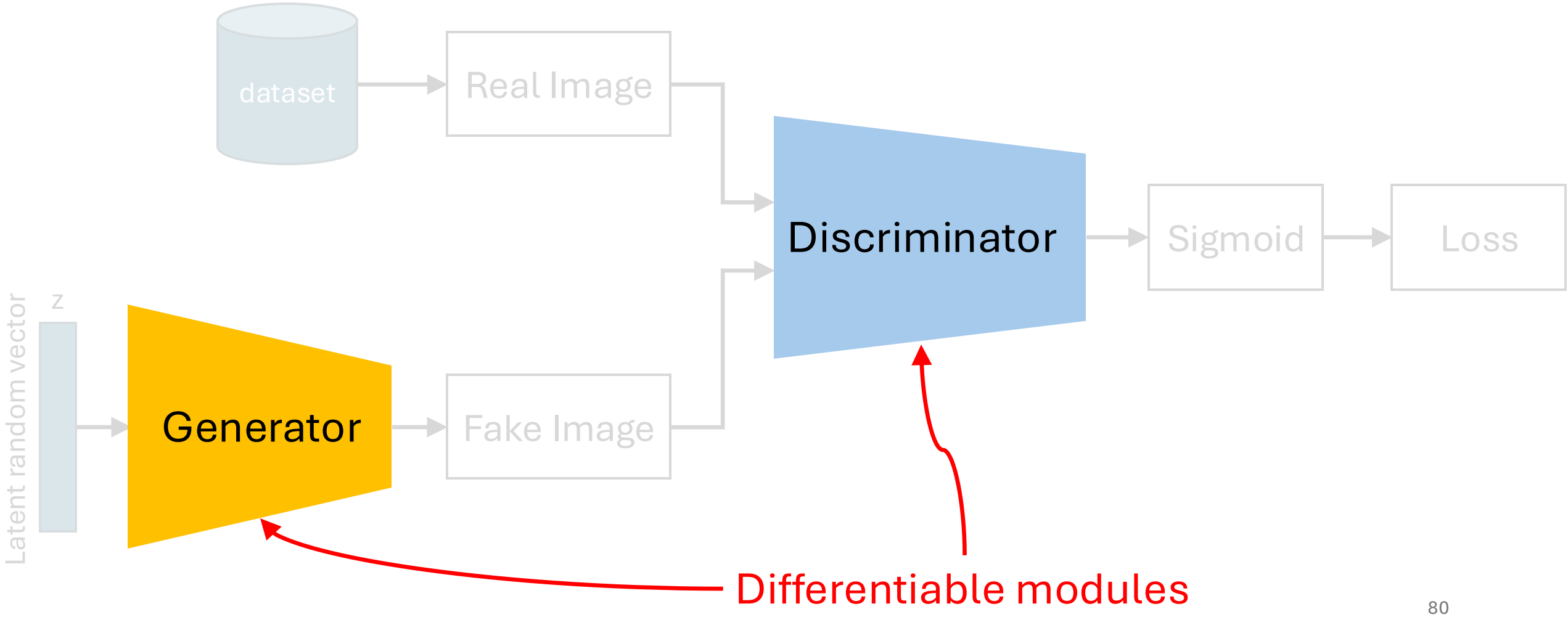
Discriminator

Parameterised function (Deep convolutional NN) that tries to distinguish between fake generated sample generated by the Generator q(x) and real samples p(x)



GAN Models

GAN architecture



GAN Models

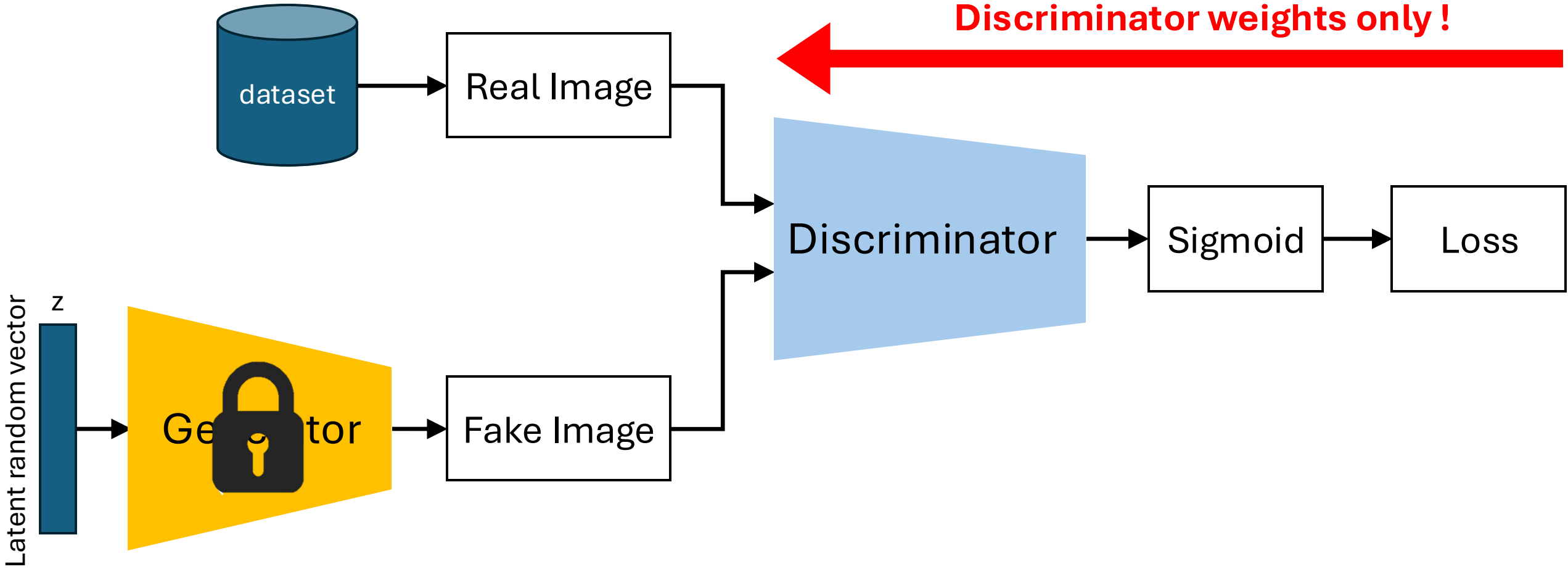
How is GAN trained

Since GAN is made of two neural network, the adopted training strategy is alternation

Meaning we alternate the training between Generator and discriminator

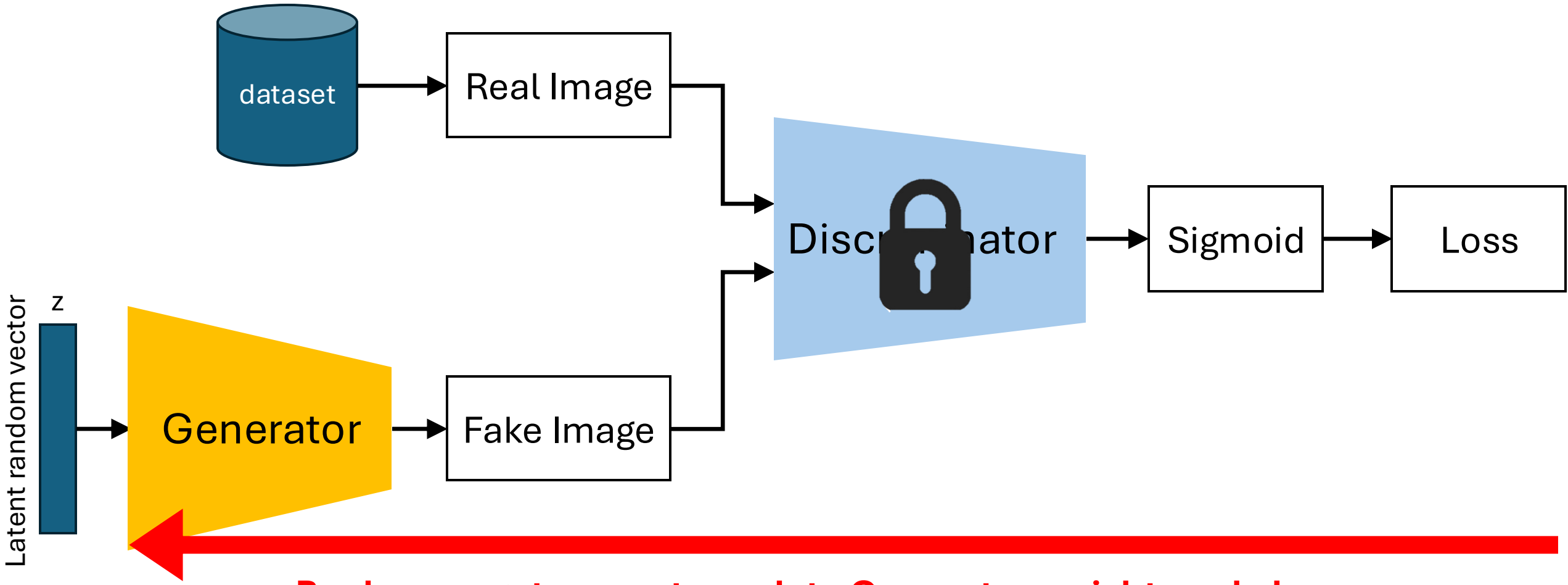
GAN Models

Training the **Discriminator**



GAN Models

Training the **Generator**



Backpropagate error to update Generator weights only !

GAN Models

Mathematical formulation of a GAN:

Two-player minimax game

$$\min_G \max_D \; V(D, G)$$

- **G** is the **Generator** network.
- **D** is the **Discriminator** network.
- **V(D, G)** is the **Value Function** (or objective function) that *D* tries to maximize and *G* tries to minimize.

GAN Models

Mathematical formulation of a GAN:

Two-player minimax game

$$\min_G \max_D V(D, G)$$

The optimization is a sequential process:

$\boldsymbol{max}_D$: The **Discriminator (D)** is trained to **maximize** the value function V(D, G). This corresponds to D maximizing its ability to correctly classify real data as real and generated data as fake.

$\boldsymbol{min}_G$: The **Generator (G)** is simultaneously trained to **minimize** the value function V(D, G). By minimizing V(D, G), G is trying to fool D, making D believe that the generated samples are real, thus minimizing D's performance (or maximizing D's loss).

GAN Models

Mathematical formulation of a GAN:

Two-player minimax game

$$\min_G \max_D V(D, G)$$

$$V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim q(z)}[\log (1 - D(G(z)))]$$

HSLU Hochschule Luzern

GAN Models

$$V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim q(z)}[\log(1 - D(G(z)))]$$

E Expectation The mathematical expectation (average value) of the term that follows, taken over the probability distribution.

x Real Data Sample A data point drawn from the true data distribution.

z Noise/Latent Vector A random vector (typically from a simple distribution like a uniform or Gaussian distribution) used as input to the Generator G.

$p_{data}(x)$ **Data Distribution** The unknown true probability distribution of the real data x. The GAN aims to learn this distribution.

q(z) Noise Distribution The probability distribution from which the input noise vector z is drawn

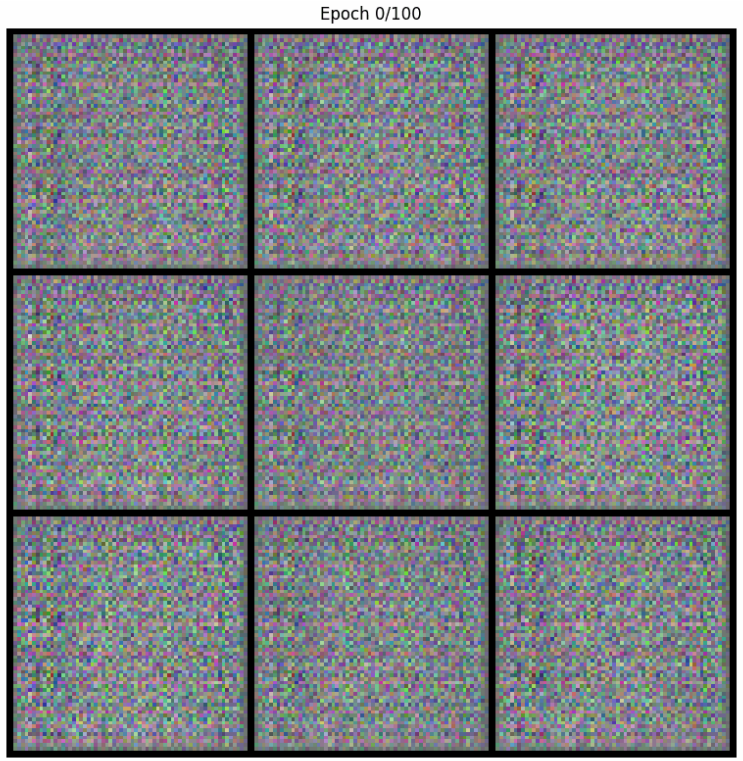
G(z) Generator Output The data sample generated by the Generator G from the noise input z. This output attempts to mimic the real data x.

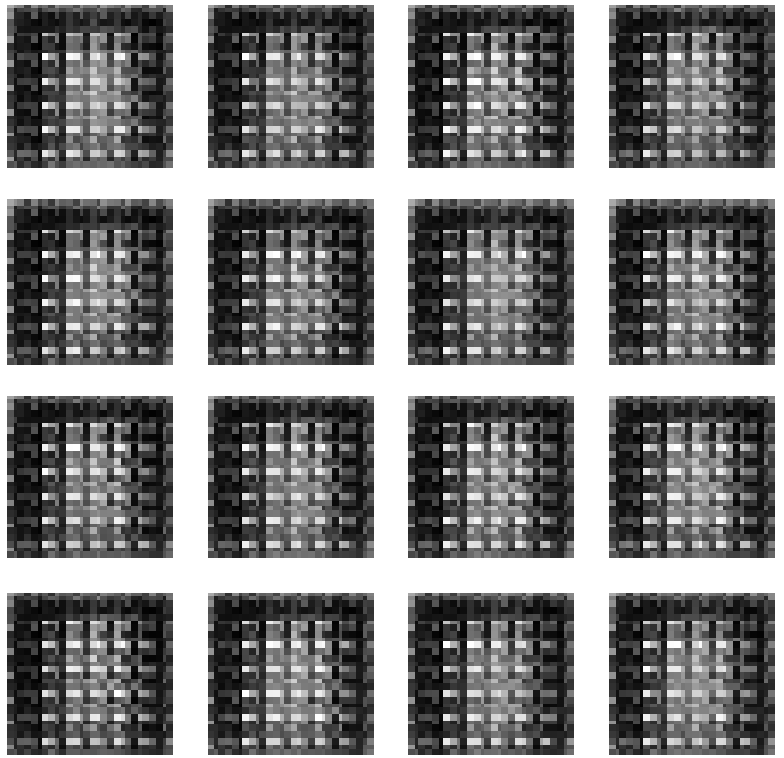
D(x) Discriminator Output (Real) The Discriminator's output (a scalar between 0 and 1) representing the probability that the real sample x came from the true data distribution $p_{data}(x)$.

D(G(z)) Discriminator Output (Fake) The Discriminator's output (a scalar between 0 and 1) representing the probability that the generated sample G(z) came from $p_{data}(x)$.

GAN Models

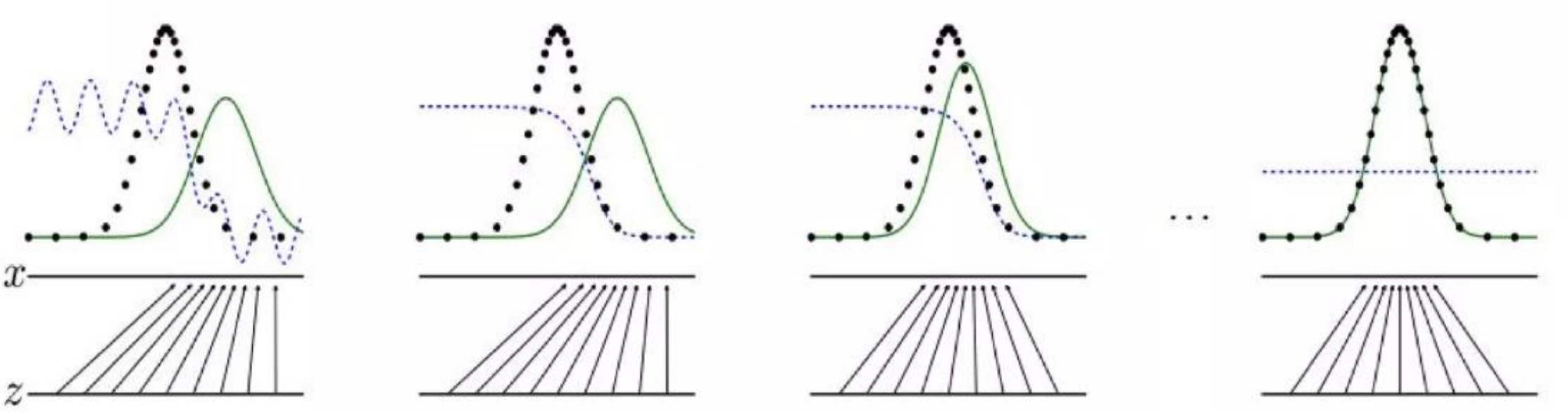
Generator in action : from random to clean images





Summary of GAN model

Training GANs involves repeating two steps until the model stabilizes, although unguaranteed:
1- First, the discriminator is updated to better distinguish between real images and those generated by the model. This means the discriminator improves its ability to tell real from fake.
2- Second, the generator is updated to better fool the current discriminator, enhancing its skill at creating realistic images.
Over time, the goal is for the generator to become so effective that the discriminator can no longer tell the difference between real and generated images. Ideally, when this happens, the discriminator's accuracy drops to 50%, indicating it is guessing randomly.



HSLU Hochschule Luzern

Hands-on:

(1) Implementation of DCGAN

HSLU Hochschule Luzern

Hands-on:

(1) Implementation of Conditional GAN

AGENDA. DAY 2

- Background & Motivations
- Transformers in vision (ViTs)
- Foundation models used in new generation of Generative AI models : CLIP and Zero-Shot Learning
- Hands-on:
  - Pre-trained classification model
  - Visualization of features map
  - Finetuning ViT on classification task
  - Inference CLIP for zero-shot classification
- Diffusion Model – introduction to Markov Chain
- Text-to-image
- Domain adaptation
- Inpainting
- Hands-on

Original Tranformer paper

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

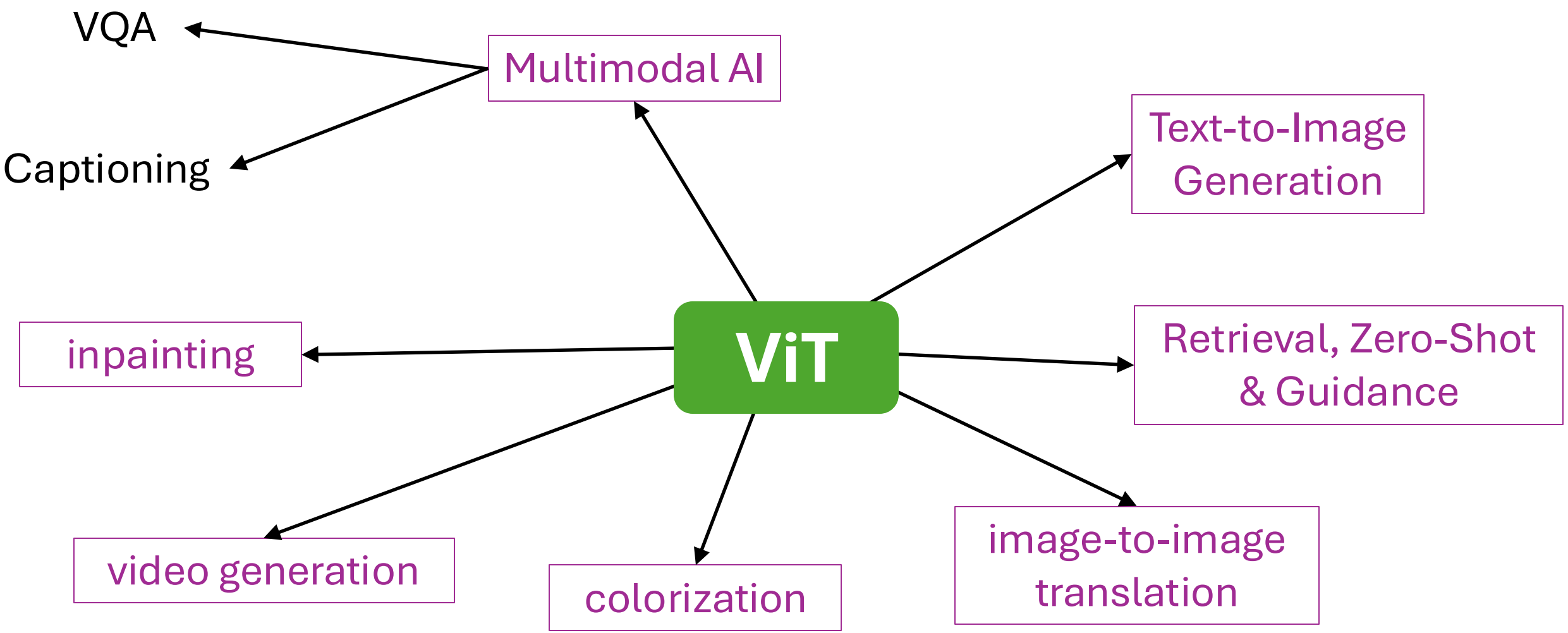
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

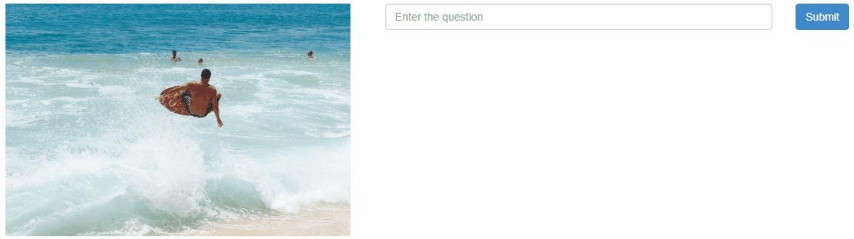
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Applications in Generative AI



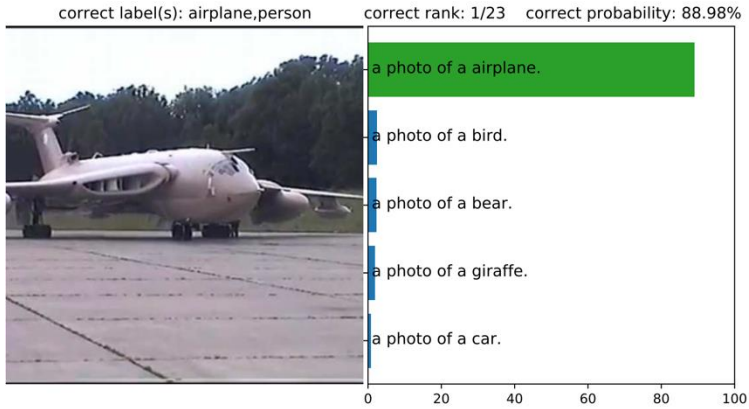
HSLU Hochschule Luzern Applications in Generative AI

Multimodal AI: VQA



Text-to-Image Generation

Text-Image Alignment



inpainting

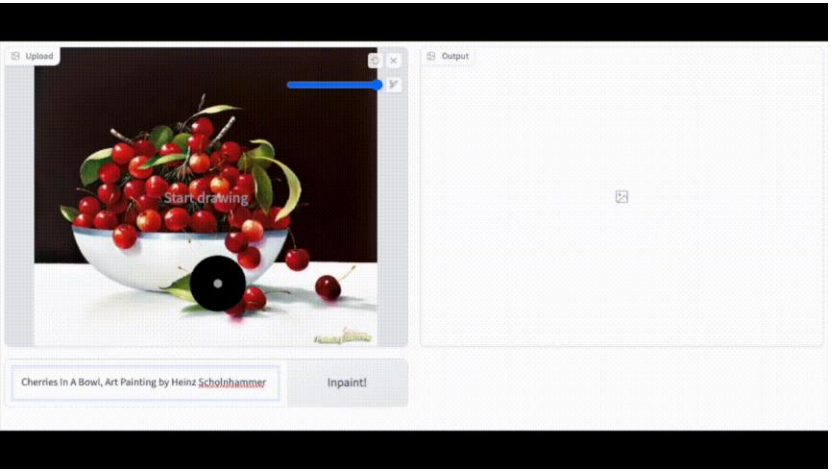
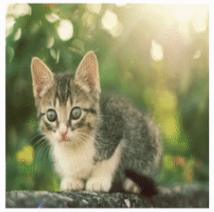
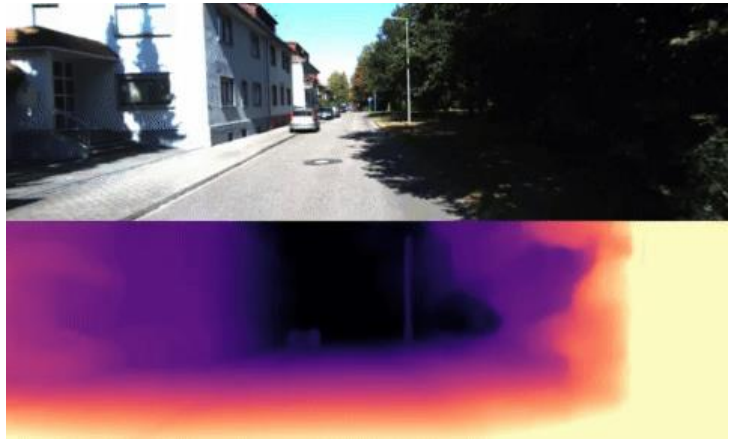


image-to-image translation



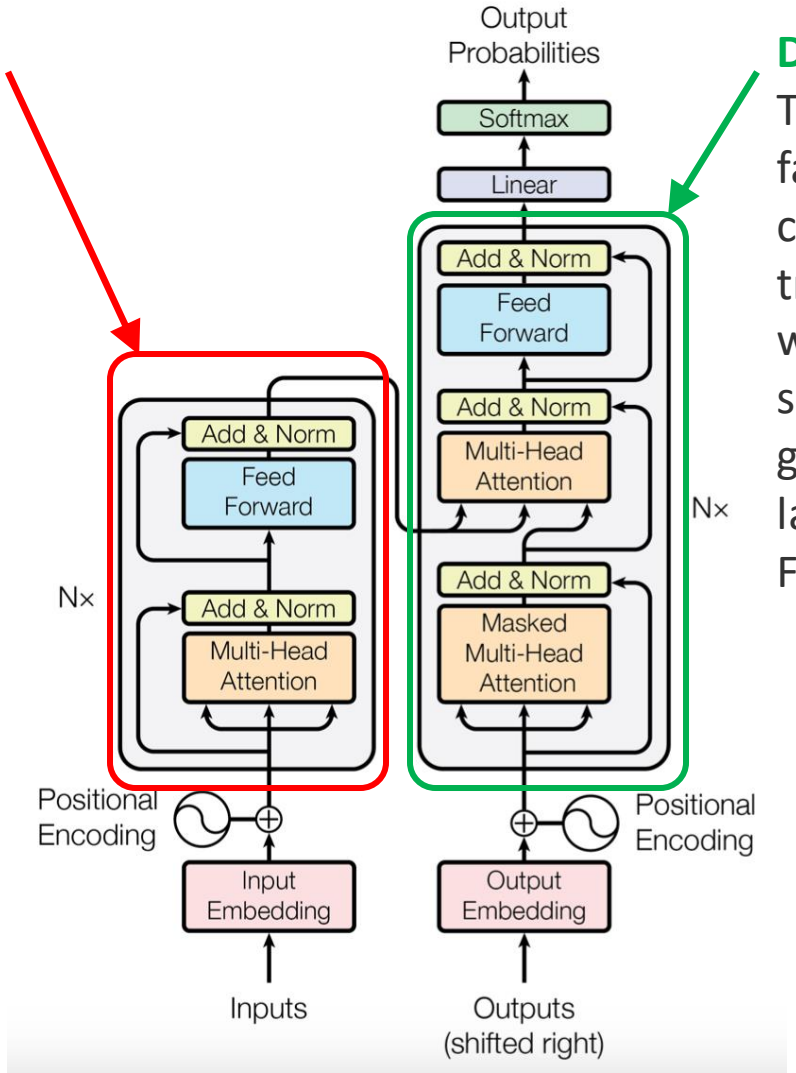
Monocular depth



Transformers in vision (ViTs)

Encoder

The encoder acts as the initial quality check in this data factory. It examines the raw materials (your sentence or paragraph) and prepares a detailed report (context or memory) for the next stage. For example, if the input is the sentence “I love ice cream,” the encoder processes each word and its relationship to the others, creating a ‘context’ that captures the sentiment and subject of the sentence.



Decoder

The decoder is the craftsman of this factory. It takes the ‘context’ and crafts the final product. In a translation task, for instance, it would take the ‘context’ from the sentence “I love ice cream” and generate its equivalent in another language, like “J’adore la glace” in French.

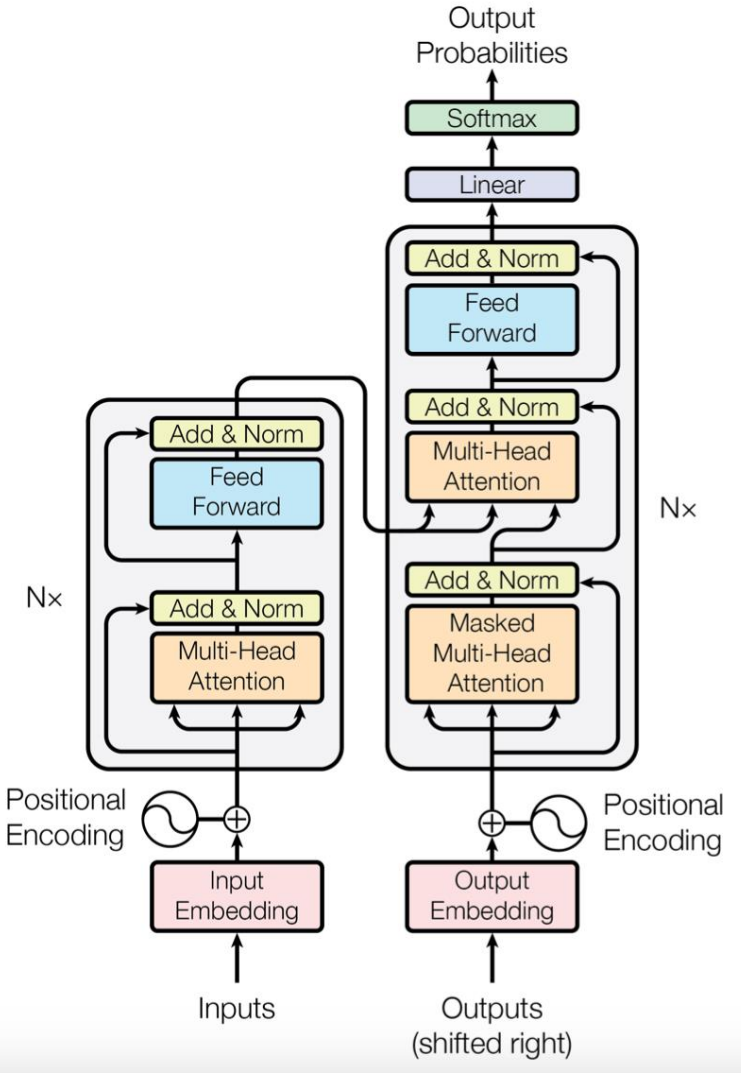
Transformers in vision (ViTs)

Query, Key, Value Vectors

In self-attention, each word in the input sentence is transformed into Query, Key, and Value vectors. These vectors are computed in a way that allows the model to focus on specific parts of the input. For example, in the sentence “The cat sat on the mat,” the word “cat” would have a Query vector querying the other words, Key vectors that serve as keys to unlock those queries, and Value vectors that contain the actual content to focus on.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



Transformers in vision (ViTs)

First ViT paper

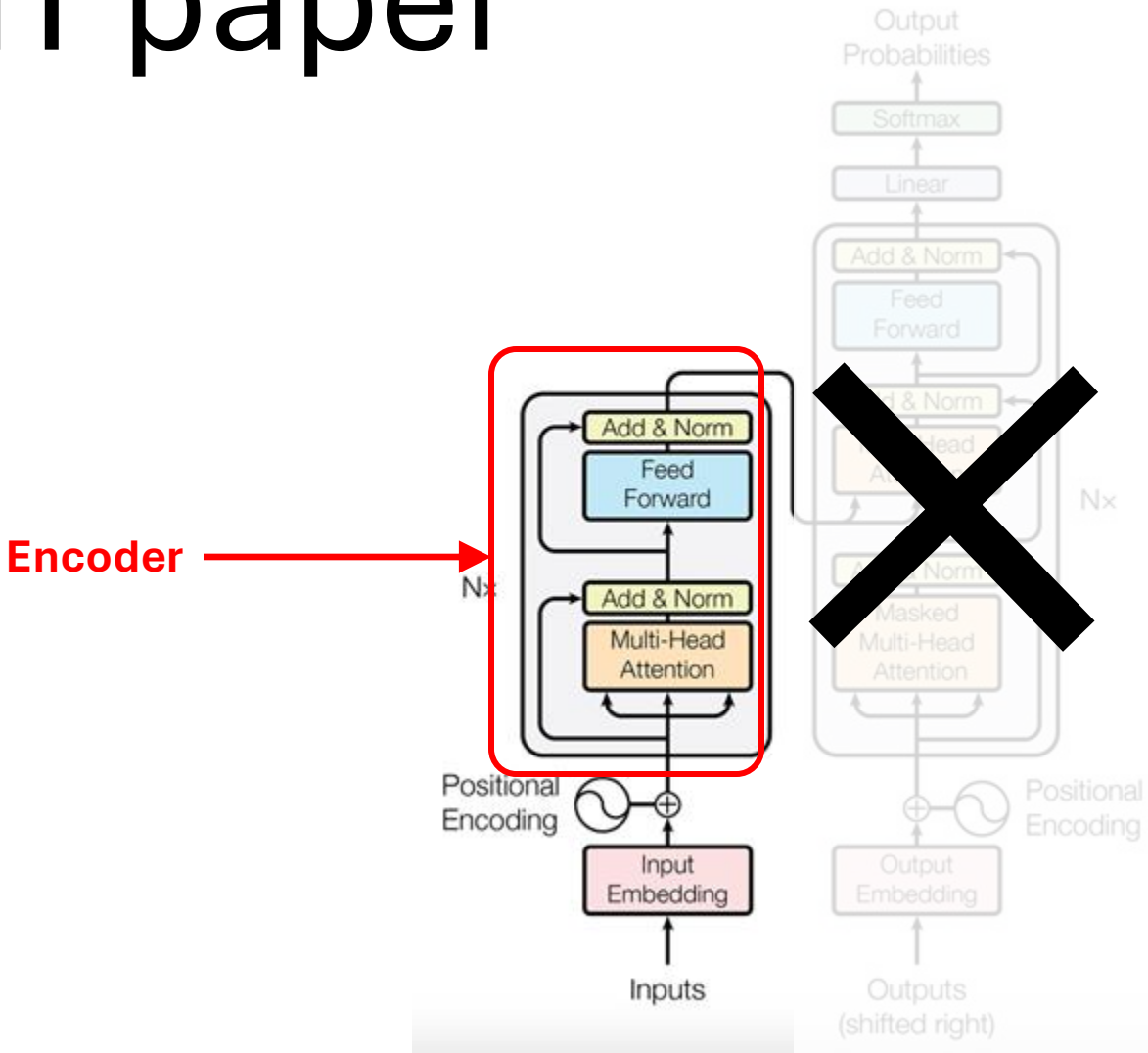
AN IMAGE IS WORTH 16X16 WORDS:
TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*],
Xiaohua Zhai[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,**
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby[*,†]

[*]equal technical contribution, [†]equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

Transformers in vision (ViTs)

First ViT paper



Transformers in vision (ViTs)

Objectives:

1. **Demonstrate the Applicability of Pure Transformers to Image Recognition**: Show that a standard Transformer architecture can be applied directly to sequences of image patches for classification tasks, without relying on convolutional neural networks (CNNs), thereby challenging CNN dominance in computer vision.
2. **Evaluate Performance on Large-Scale Pre-Training and Transfer Learning**: Assess the Vision Transformer (ViT) model's performance when pre-trained on massive datasets (e.g., ImageNet-21k or JFT-300M) and fine-tuned on various benchmarks (e.g., ImageNet, CIFAR-100, VTAB), comparing it to state-of-the-art CNNs.
3. **Investigate the Impact of Dataset Scale on Transformer Performance**: Explore how training dataset size influences ViT's effectiveness, emphasizing that large-scale data can compensate for Transformers' lack of built-in inductive biases like those in CNNs (e.g., translation equivariance and locality).
4. **Compare Computational Efficiency and Resource Requirements**: Highlight that ViT achieves strong results with significantly lower computational costs during training compared to leading CNNs, using metrics such as TPUv3-core-days.

Transformers in vision (ViTs)

Aspect	NLP Tokenization	Vision Tokenization
Definition	Process of breaking down text into smaller units (tokens) such as words, subwords, or characters.	Process of dividing an image into fixed-size patches or regions to process as tokens.
Normalization	- Converts text to a standard format (e.g., lowercasing, removing accents). - Removes punctuation or special characters. - Handles contractions (e.g., "don't" → "do not").	- Normalizes pixel values (e.g., scaling to [0,1] or standardizing with mean and std). - Adjusts color channels (e.g., RGB normalization). - Resizes images to a fixed resolution.
Segmentation/Splitting	- Splits text into tokens based on whitespace, punctuation, or subword units (e.g., WordPiece, BPE). - Example: "I am running" → ["I", "am", "running"]. - Subword tokenization for rare words (e.g., "unhappiness" → ["un", "##happiness"]).	- Divides image into fixed-size patches (e.g., 16x16 pixels). - Each patch is treated as a token. - Example: A 224x224 image with 16x16 patches → 196 patches (tokens).
Cleanup	- Removes irrelevant tokens (e.g., extra whitespace, stop words like "the"). - Filters out low-frequency tokens or replaces them with [UNK]. - Handles encoding issues (e.g., UTF-8 normalization).	- Removes or adjusts noisy patches (e.g., handling corrupted pixels). - Applies data augmentation (e.g., random cropping, flipping) to enhance robustness. - Filters out irrelevant regions (e.g., background suppression in some cases).
Token Representation	- Tokens are mapped to vocabulary indices or embeddings (e.g., Word2Vec, BERT embeddings). - Each token is a discrete unit in a vocabulary.	- Patches are flattened and projected to a fixed-dimensional embedding space (e.g., linear projection in ViT). - Each patch is a vector in a continuous space.
Handling Variability	- Handles linguistic variations (e.g., synonyms, misspellings, multilingual text). - Uses subword tokenization to manage out-of-vocabulary words.	- Handles variations in lighting, orientation

Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:

Transformers in vision (ViTs)

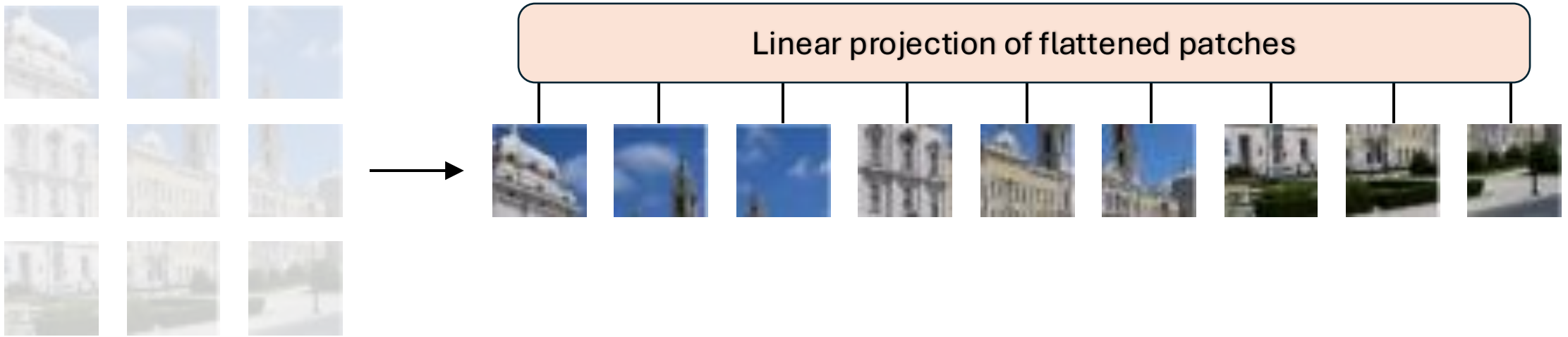
OVERVIEW OF THE ViT-ARCHITECTURE:

Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:

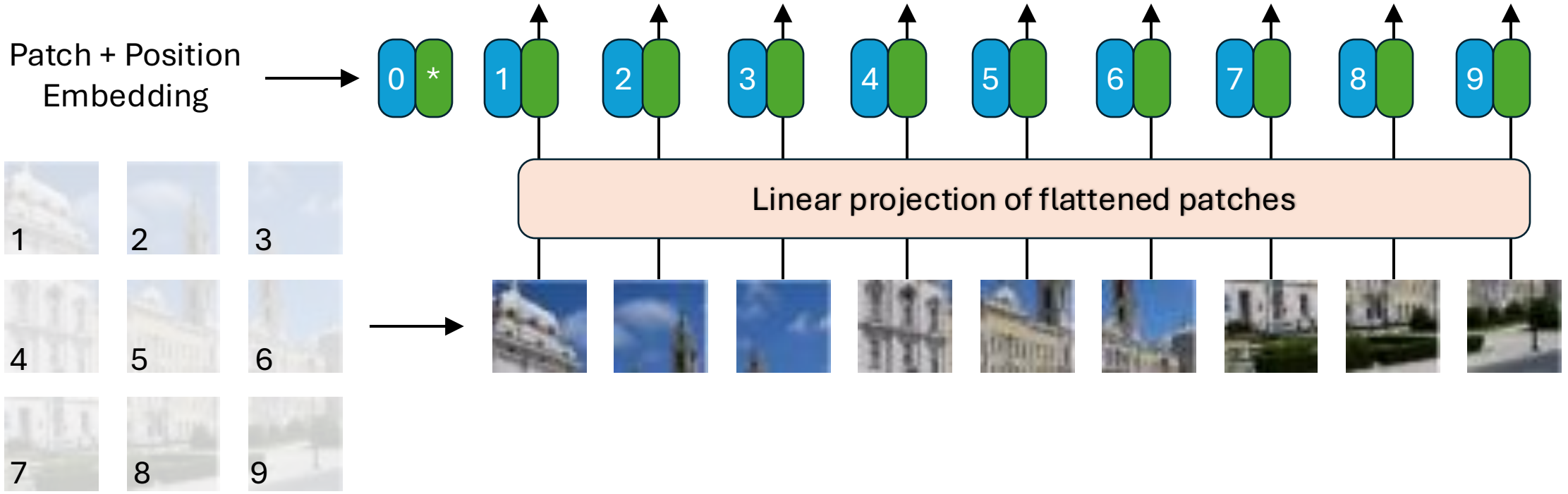
Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:



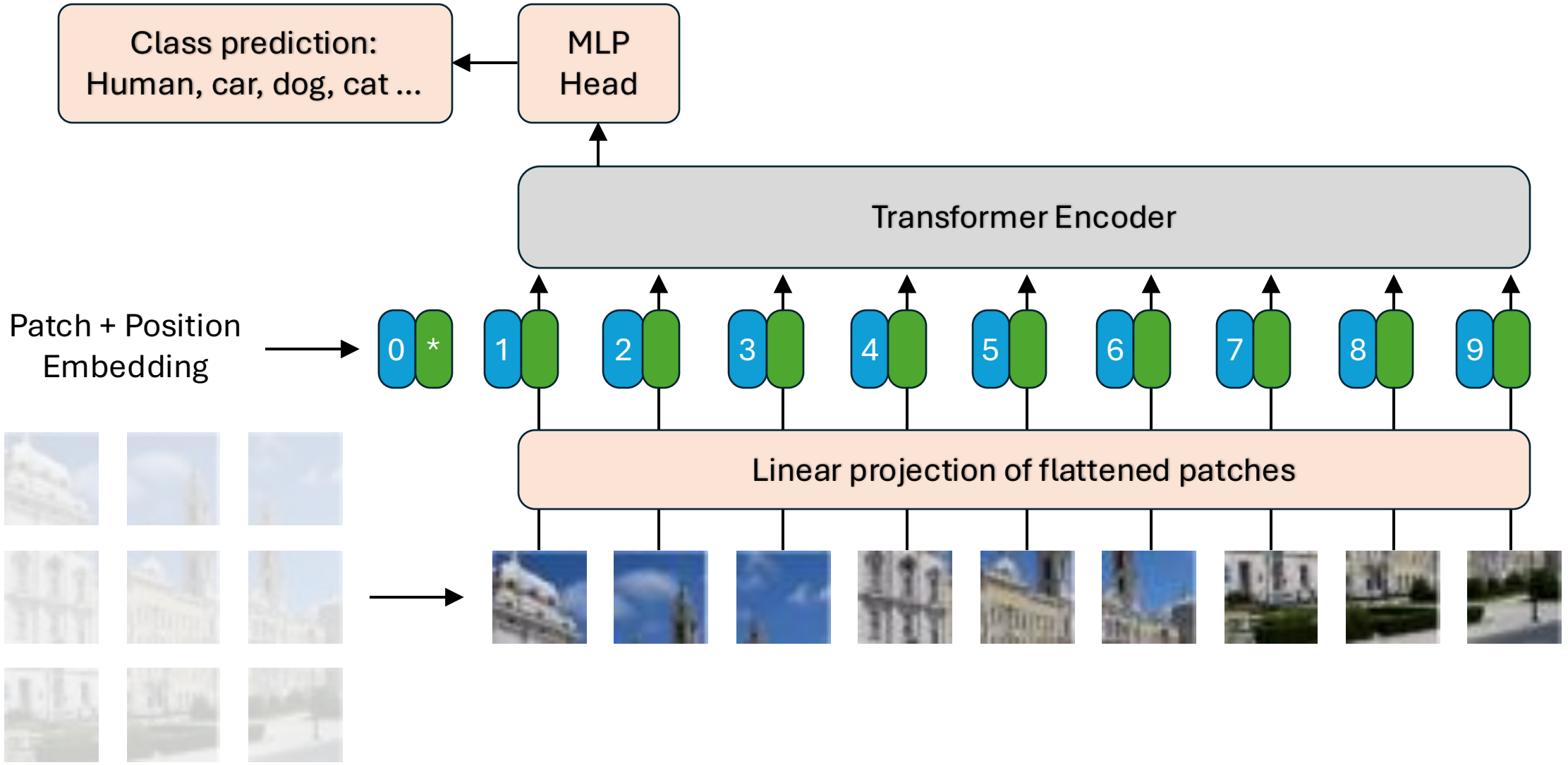
Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:



Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:

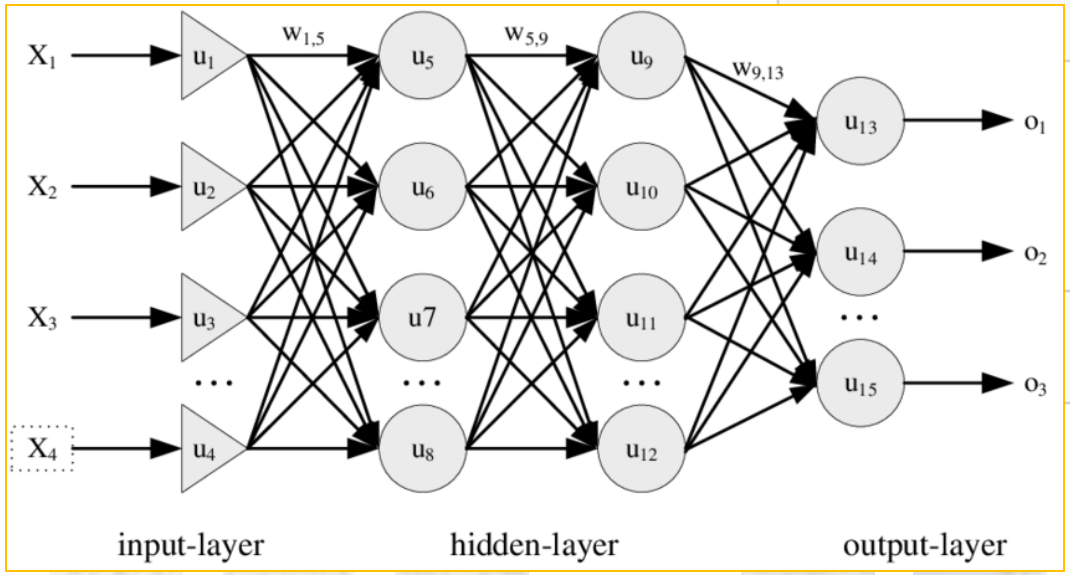


Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:

Class prediction:
Human, car, dog, cat ...

MLP Head

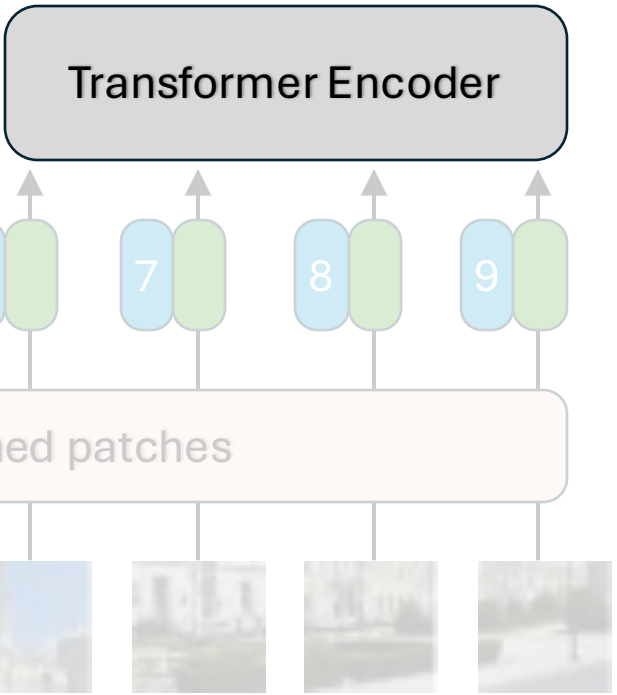


In DL, a multilayer perceptron (MLP) is a name for a modern feedforward neural network consisting of fully connected neurons with nonlinear activation functions, organized in layers, notable for being able to distinguish data that is not linearly separable.

The MLP consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes. Since MLPs are fully connected, each node in one layer connects with a certain weight w_{ij} to every node in the following layer.

Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:



What is the "Transformer Encoder" how different / similar to NLP is it?

Both encoders are fundamentally similar in architecture and operation, but there are key differences due to the nature of the input data

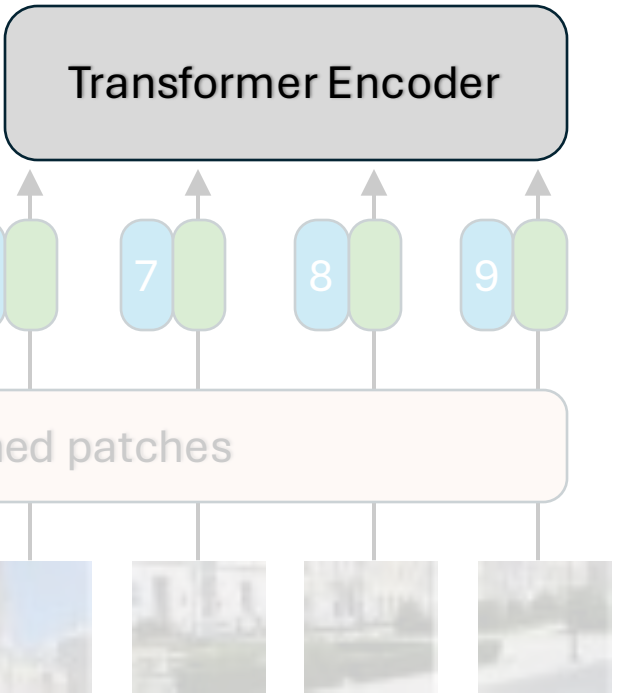
Core Similarity

- **Multi-Head Self-Attention**: Computes relationships between all input tokens/patches, capturing global dependencies.
- **Feed-Forward Networks (FFNs)**: Applied independently to each token/patch for feature transformation.
- **Layer Normalization and Residual Connections**: Stabilize training and improve gradient flow.
- **Positional Encoding/Embeddings**: Preserve sequence or spatial order.

The encoder processes a sequence of input tokens (or patches in ViTs) and produces contextualized representations by attending to all elements in the sequence simultaneously.

Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:



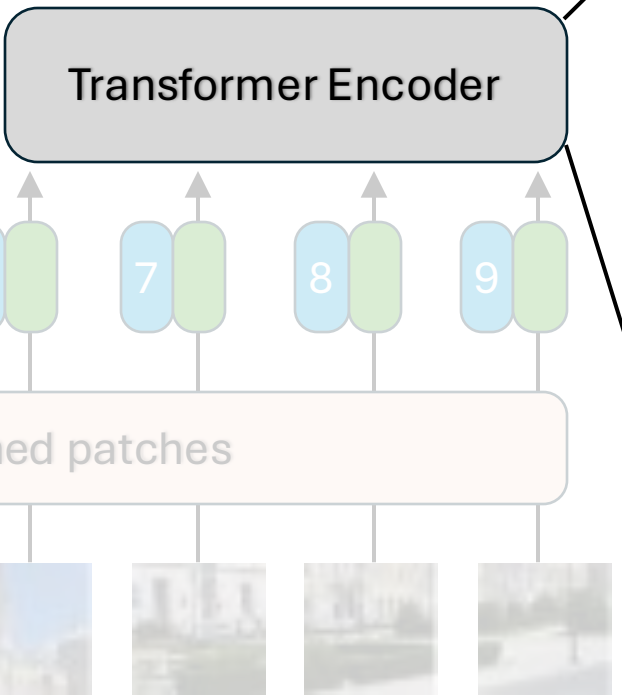
What is the "Transformer Encoder" how different / similar to NLP is it?

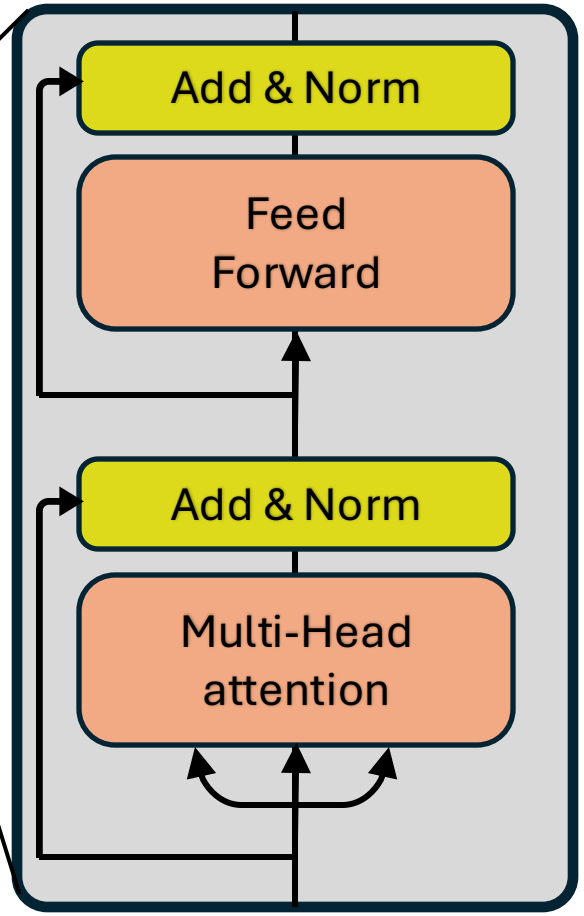
Differences

- **Input**
  - **NLP**: Takes a sequence of tokenized text (such as words, subwords) and tokens are mapped to embeddings via a vocabulary lookup (e.g., WordPiece)
  - **Vision**: Takes a sequence of image patches (e.g., 16x16 pixel patches). And the patches are flattened and linearly projected to a fixed-dimensional embedding space.
- **Positional Encoding**
  - **NLP**: In natural language, the input is a 1D sequence of words. The position can be represented by a single index, like 1, 2, 3, and so on. The positional encoding is therefore a 1D vector corresponding to each word's index.
  - **Vision**: Images are 2D. The position of a patch is defined by its row and column coordinates. While some ViTs use a flattened 1D sequence of patches (e.g., from top-left to bottom-right), more advanced methods might use a 2D encoding that explicitly models the row and column positions

Transformers in vision (ViTs)

OVERVIEW OF THE ViT-ARCHITECTURE:





Residual Connection & **Layer Normalization**
(Skip connections)

$$head_i = Attention(Q * W_i^Q, K * W_i^K, V * W_i^V)$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V$$

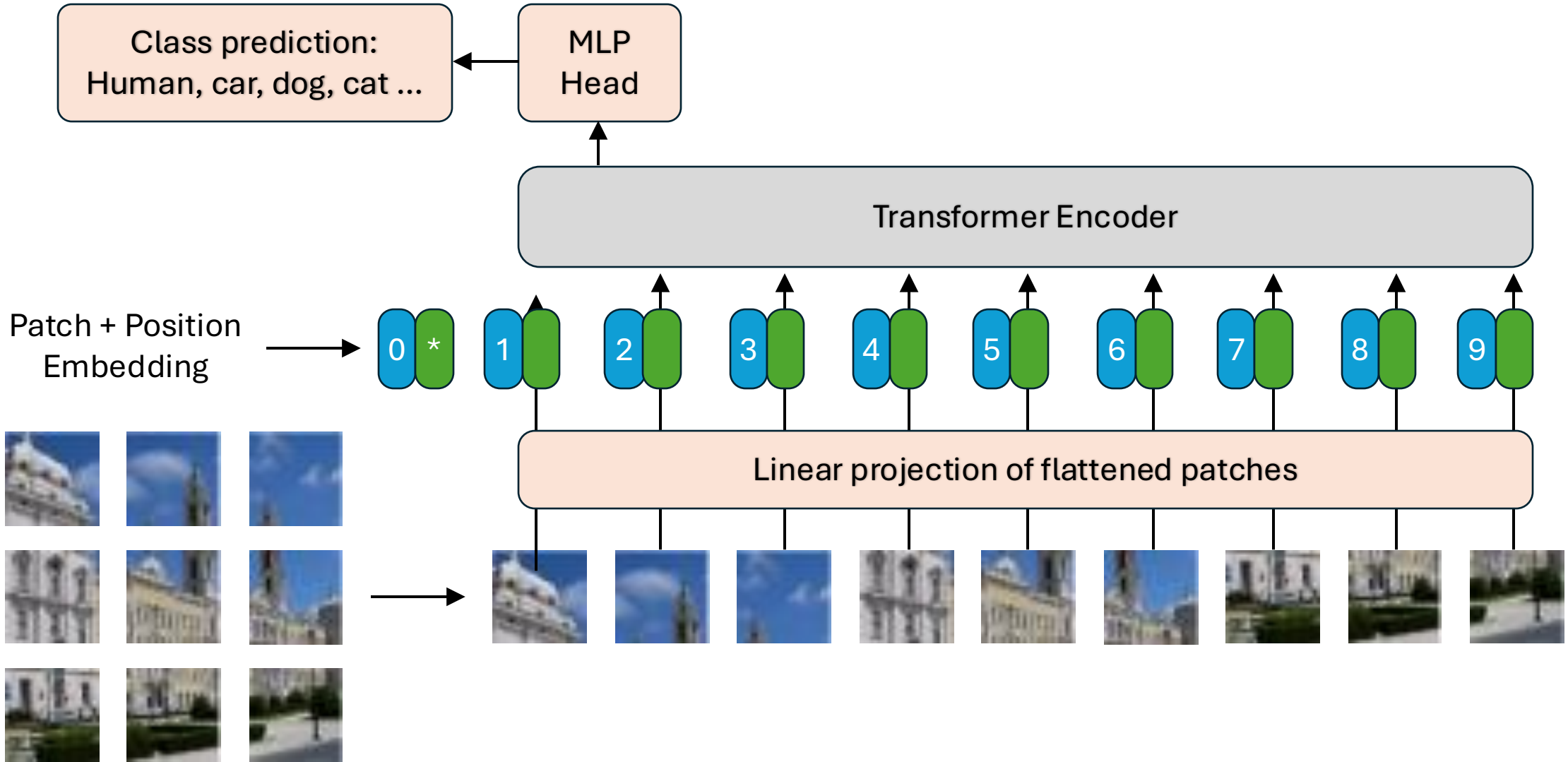
Q (Query), K (Key), and V (Value)

HSLU Hochschule Luzern

Divers ViTs architectures

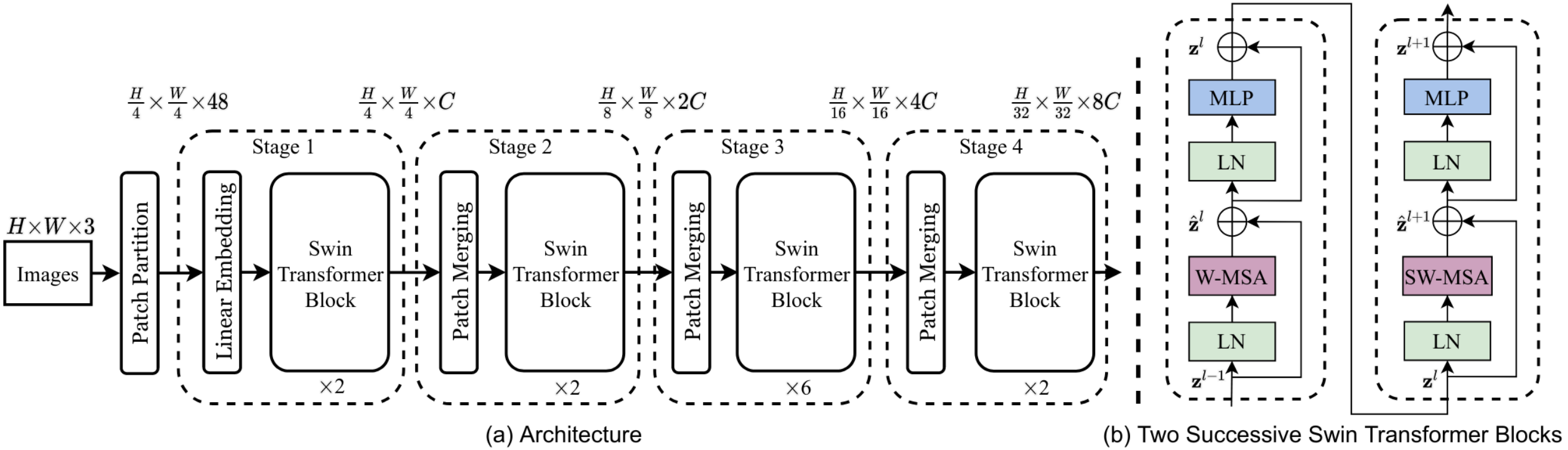
Divers ViTs architectures

ViT : Google - Vision Transformer (paper)



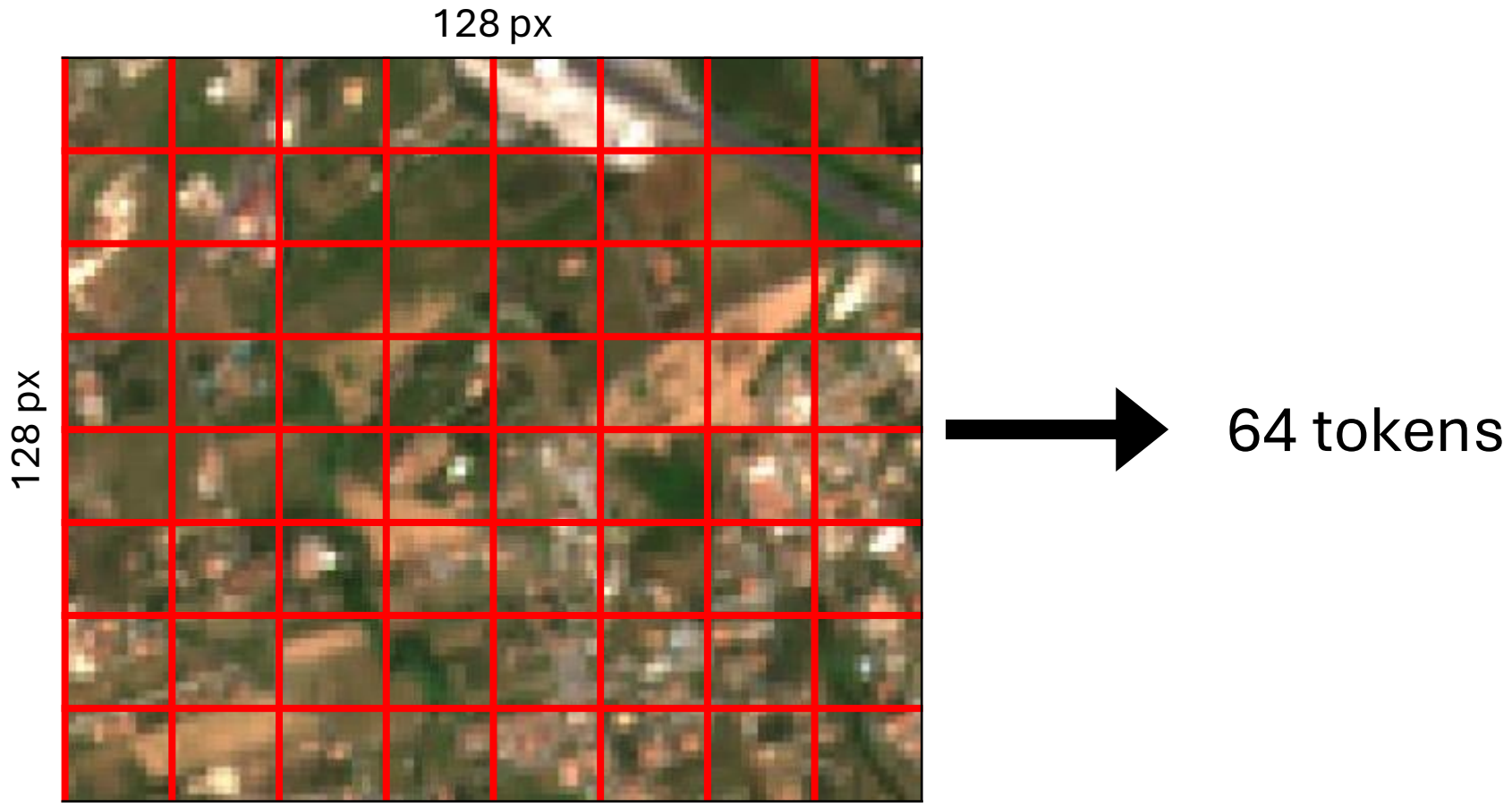
Divers ViTs architectures

Microsoft – Swin-Transformer (paper)



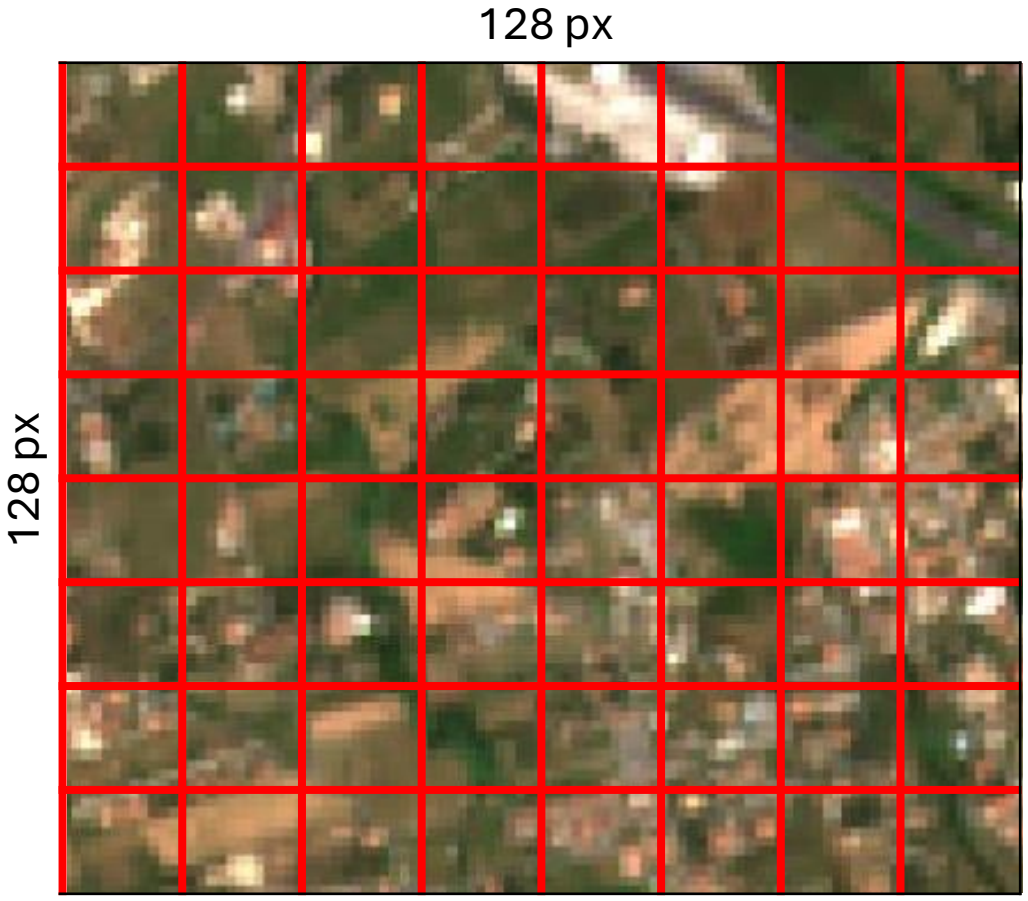
Divers ViTs architectures

Microsoft – Swin-Transformer (paper)

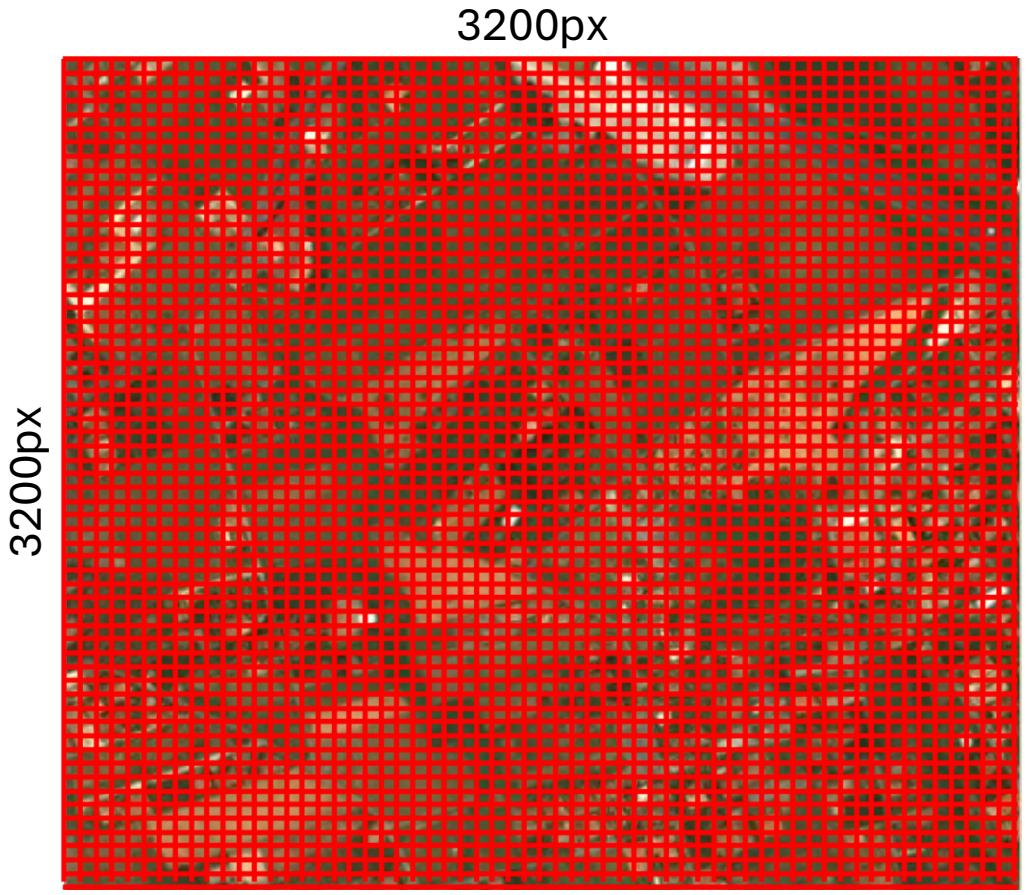


Divers ViTs architectures

Microsoft – Swin-Transformer (paper)



(128 / 16 x 128 /16) = 64 tokens



(3200 / 16 x 3200 /16) = 40000 tokens

Divers ViTs architectures

Microsoft – Swin-Transformer (paper)

The Swin Transformer was developed to solve two main problems with the original Vision Transformer (ViT):

quadratic computational complexity

The original ViT treats an image as a sequence of non-overlapping patches and applies a standard Transformer encoder to them. A key operation within the Transformer is **self-attention**, which calculates the relationship between every patch and every other patch. The computational cost of this operation grows quadratically with the number of patches ($O(n^2)$), where n is the number of patches.

- A high-resolution image needs more patches
- The quadratic cost makes the model very slow and memory-intensive, impractical for tasks like object detection or semantic segmentation

lack of hierarchical feature representation.

The ViT architecture processes patches at a single, fixed resolution throughout the network. It doesn't have a hierarchical structure like a Convolutional Neural Network (CNN), which builds a pyramid of features from low-level details (edges, textures) to high-level semantic concepts. This makes ViT less suitable for "dense prediction" tasks like object detection and segmentation, where objects of different sizes need to be detected and a multi-scale representation is crucial.

Divers ViTs architectures

Microsoft – Swin-Transformer (paper)

The Swin Transformer solutions

quadratic computational complexity

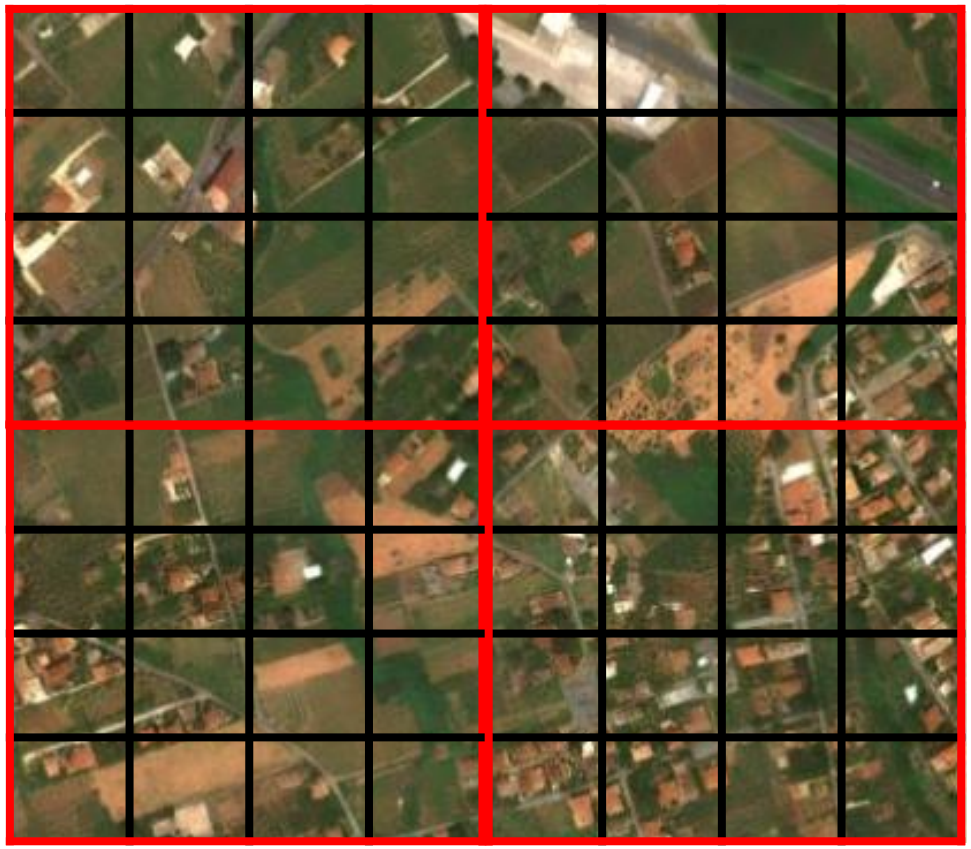
The Swin Transformer limits the self-attention computation to **non-overlapping local windows**. This changes the complexity from quadratic to **linear** with respect to the number of patches. To ensure information can still flow between these windows, Swin introduces a **shifted window** mechanism. In alternating layers, the windows are shifted, allowing patches to interact with different neighbors and learn global context over the network's depth.

lack of hierarchical feature representation.

The Swin Transformer introduces a **hierarchical architecture** that produces multi-scale feature maps, just like a CNN. It starts with small patches and then gradually merges adjacent patches in deeper layers. This downsampling process creates a feature pyramid that can capture both fine-grained details for small objects and high-level semantic information for large objects.

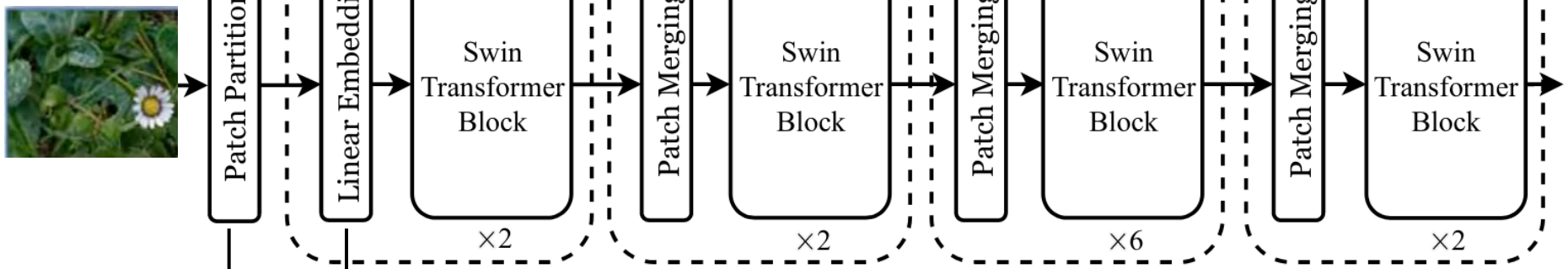
Divers ViTs architectures

Microsoft – Swin-Transformer (paper)

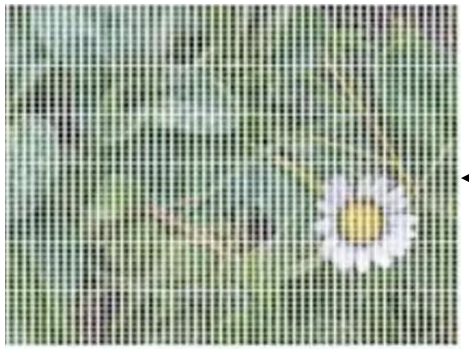


Divers ViTs architectures

Microsoft – Swin-Transformer (paper)



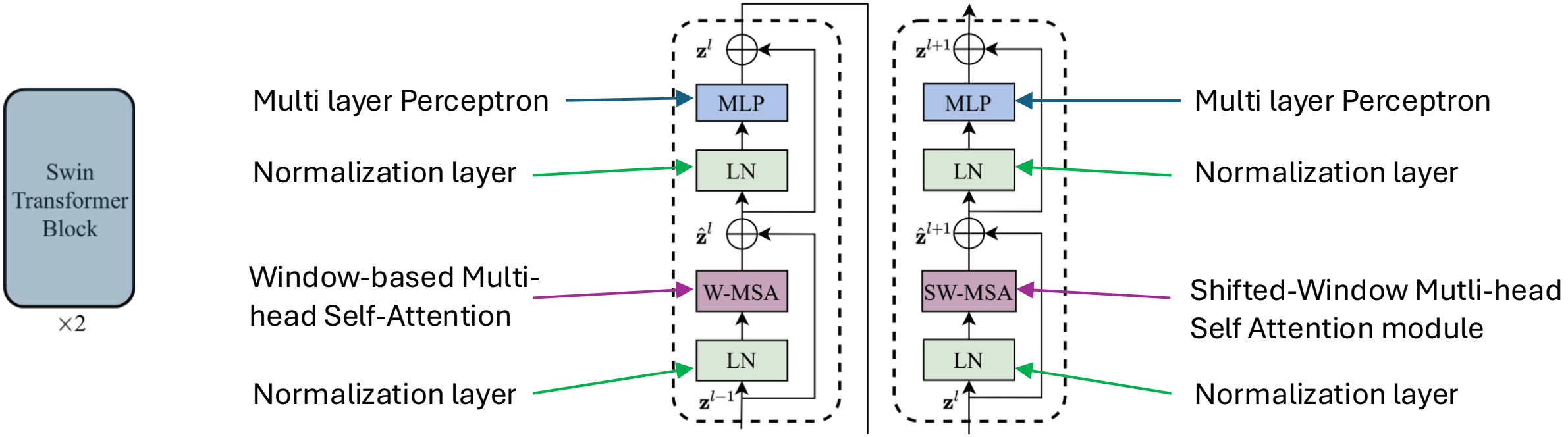
4x4 px



```
First patch embedding vector: tensor([ 0.0872,  0.0503, -0.1719,  0.1464,  0.1546, -0.0358,  0.0016, -0.3824,
       -0.2313, -0.1955,  0.1172, -0.0174, -0.1295, -0.0316,  0.0104, -0.0479,
        0.0975, -0.0499, -0.0125, -0.1404, -0.2292,  0.1697, -0.2276, -0.0672,
        0.0015, -0.0895,  0.1411, -0.0210,  0.0128,  0.1669,  0.0921, -0.1980,
       -0.2184,  0.0075,  0.1579,  0.0842,  0.0635, -0.0160,  0.2636,  0.1215,
        0.1866, -0.2069,  0.0335,  0.0802,  0.1717, -0.1648,  0.2139,  0.1257,
       -0.1655, -0.0129, -0.1318, -0.1114,  0.1215, -0.0470,  0.0056, -0.2260,
       -0.0460,  0.0295, -0.0118, -0.4155,  0.0999, -0.0171,  0.1431, -0.0716,
       -0.0708,  0.0945, -0.1236,  0.0800, -0.1344,  0.0319,  0.0184,  0.1027,
        0.0112, -0.0439, -0.2414,  0.0518,  0.1925, -0.0069,  0.0818,  0.0732,
        0.1824,  0.1271,  0.0092,  0.0964, -0.0937,  0.1673, -0.2281, -0.0448,
       -0.1030, -0.0190,  0.1538, -0.0324,  0.2129,  0.1216,  0.0420, -0.2114],
```

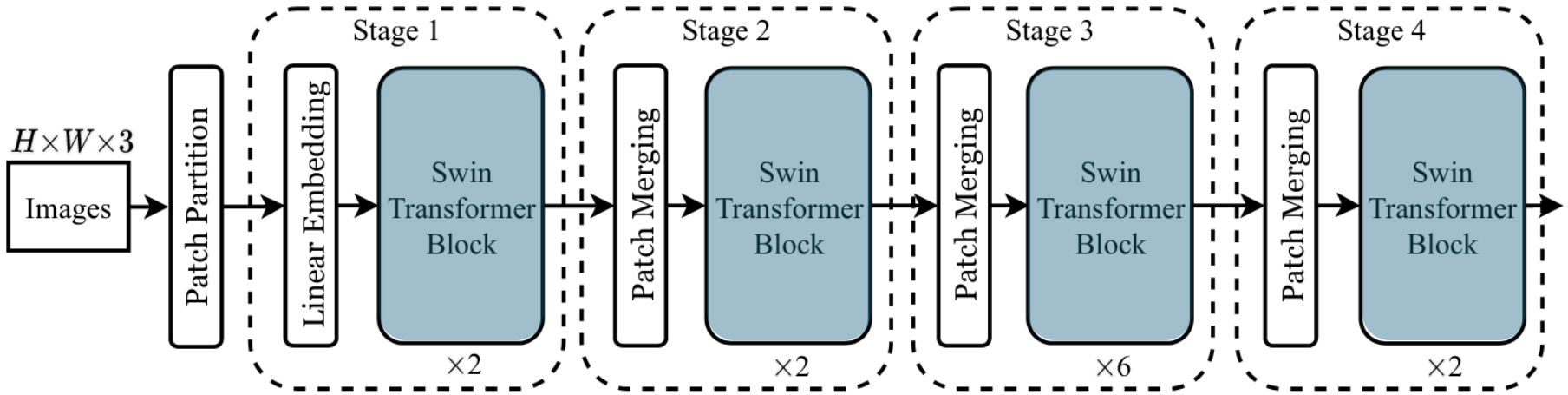
Divers ViTs architectures

Microsoft – Swin-Transformer (paper)



Divers ViTs architectures

Microsoft – Swin-Transformer (paper)



HSLU Hochschule Luzern

CLIP and zero-shot Learning

CLIP and zero-shot Learning

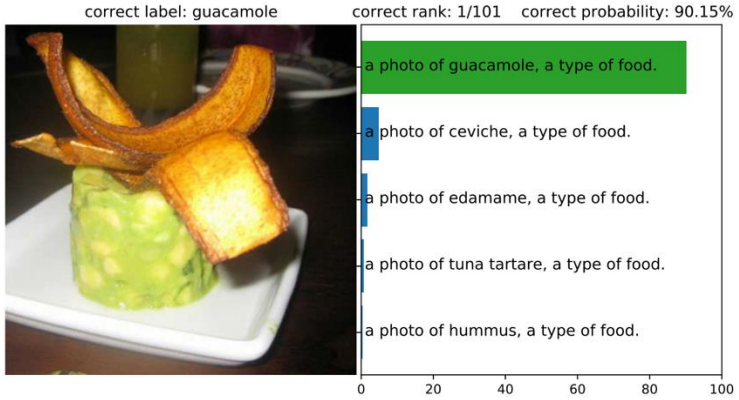
CLIP: Learning Transferable Visual Models From Natural Language Supervision (Paper)

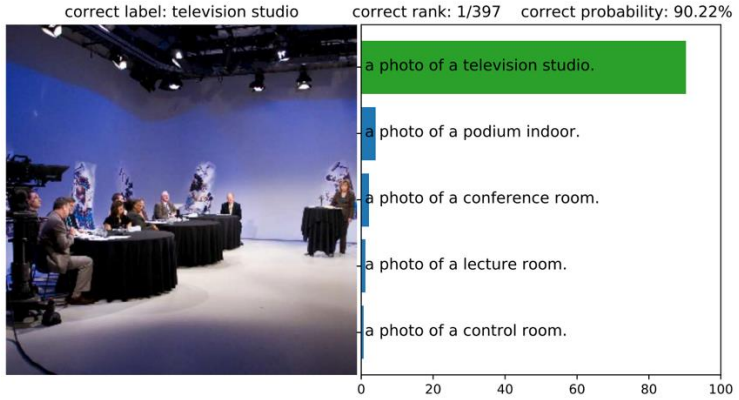
- CLIP (Contrastive Language-Image Pretraining) is a **multimodal AI** model developed by OpenAI and introduced in their 2021. It is designed to understand and align images and text in a shared embedding space, enabling tasks like **zero-shot image classification**, **image-text retrieval**, and more without task-specific training.

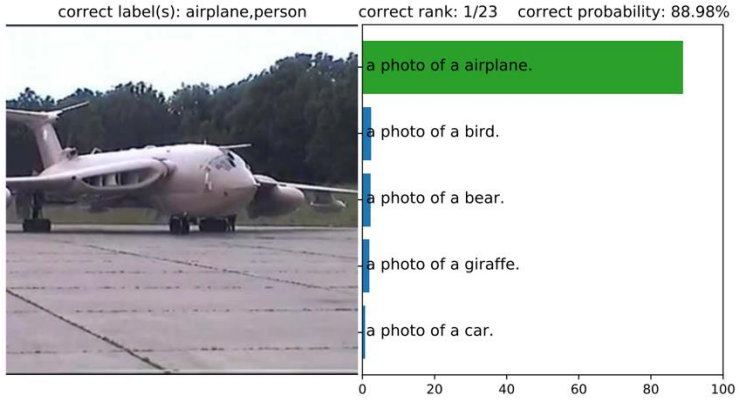
- CLIP learns from vast, noisy web-scraped image-text pairs (about 400 million in the original dataset), using natural language descriptions as supervision.

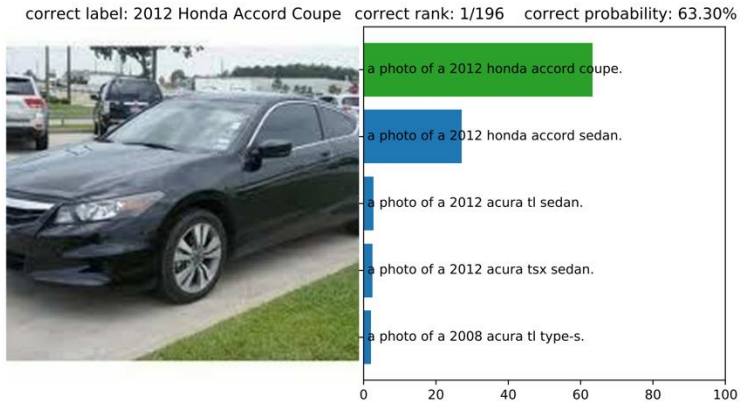
CLIP and zero-shot Learning

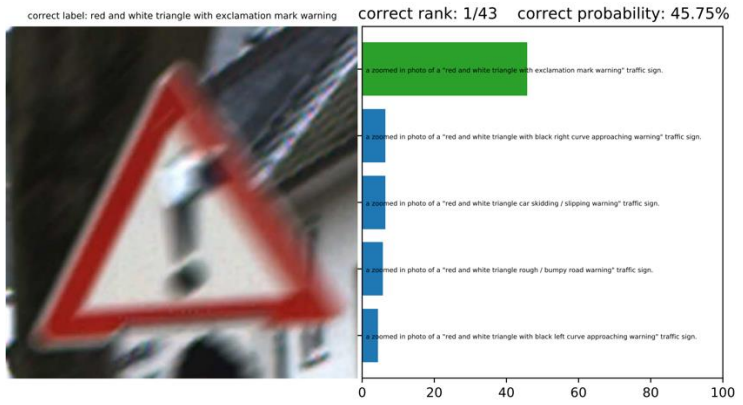
CLIP: Learning Transferable Visual Models From Natural Language Supervision (Paper)

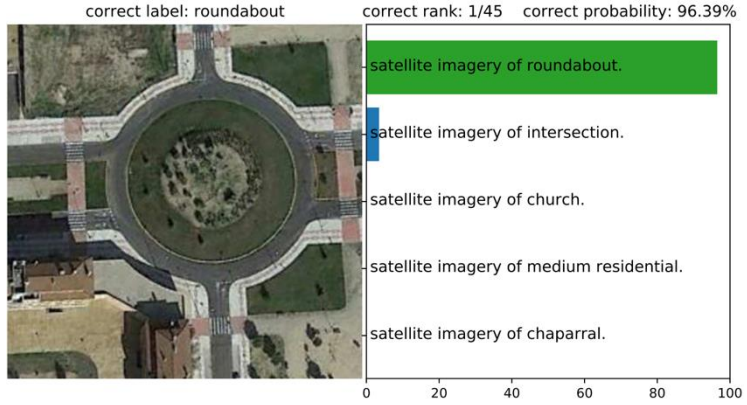






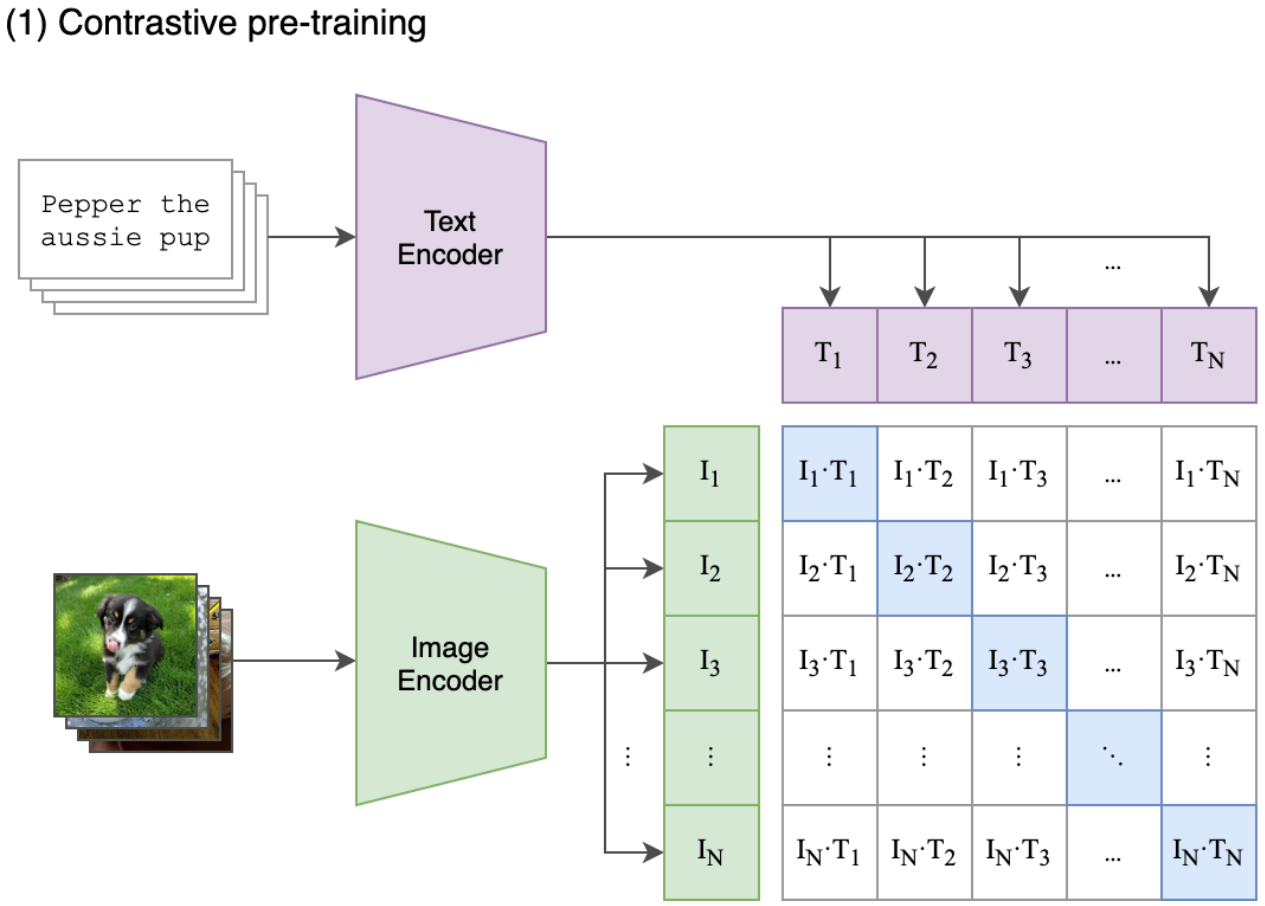


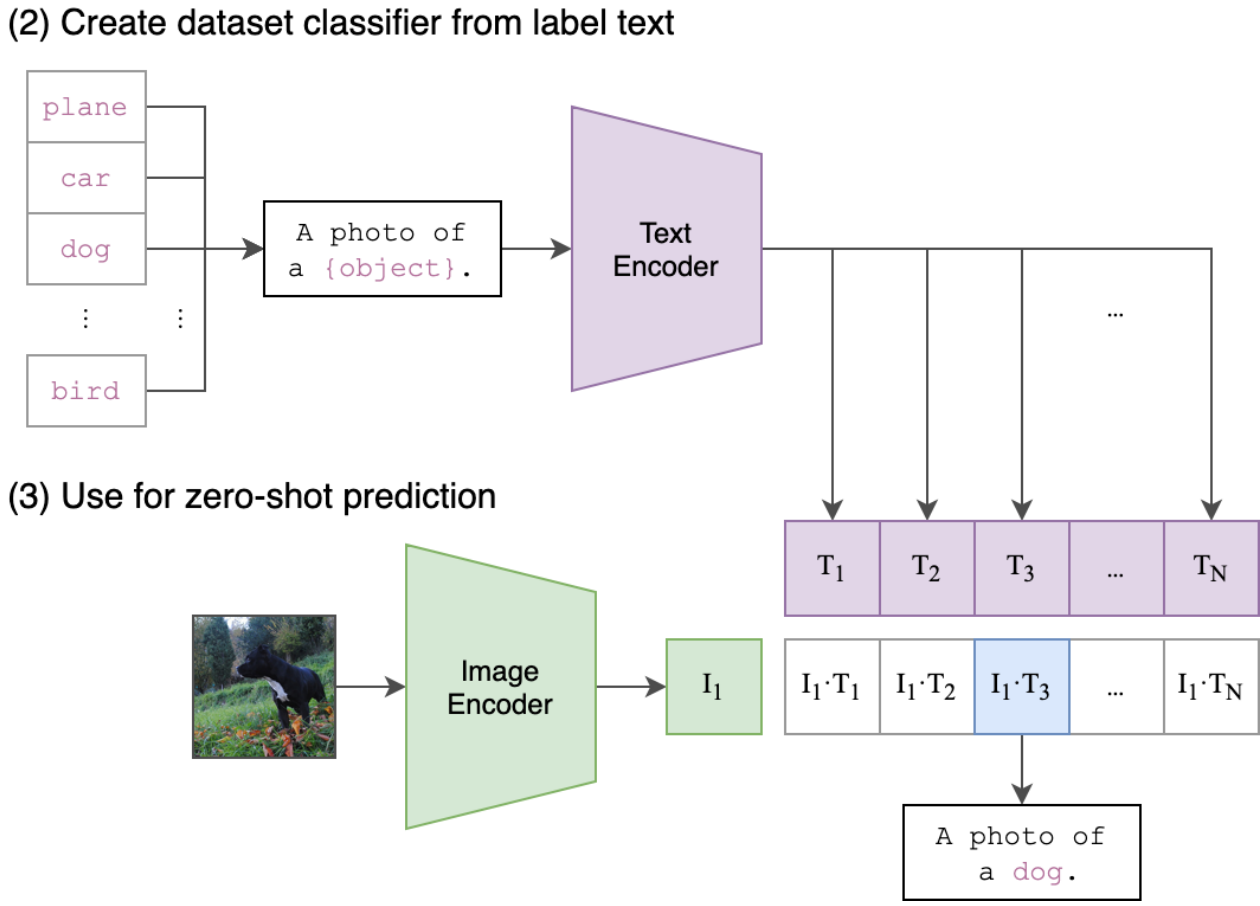




CLIP and zero-shot Learning

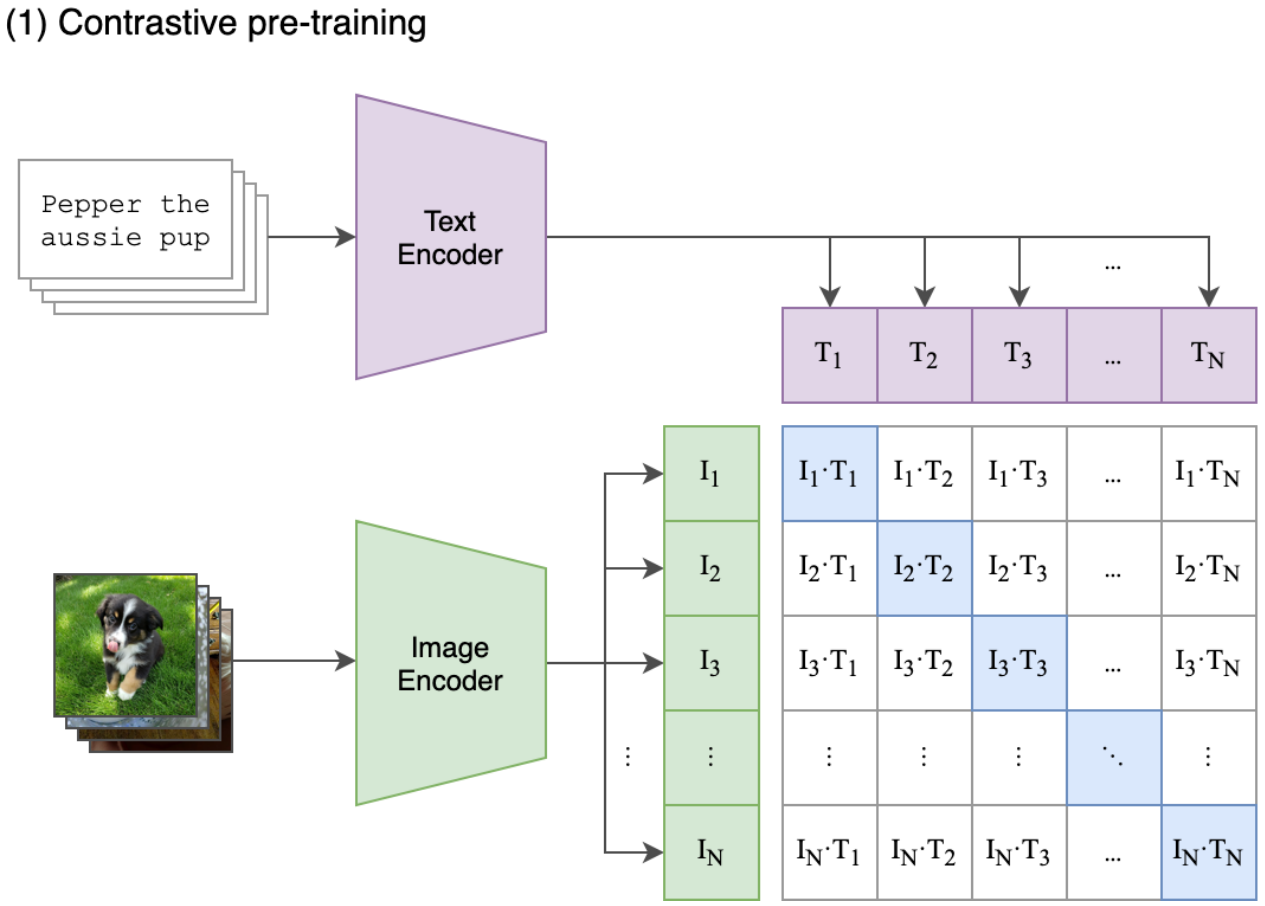
CLIP: Learning Transferable Visual Models From Natural Language Supervision (Paper)





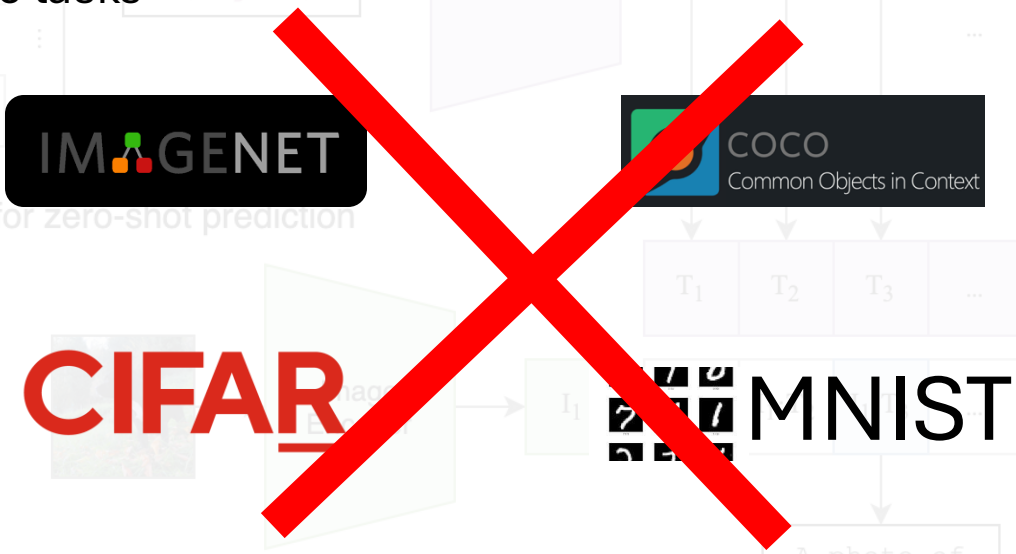
CLIP and zero-shot Learning

CLIP: Learning Transferable Visual Models From Natural Language Supervision (Paper)



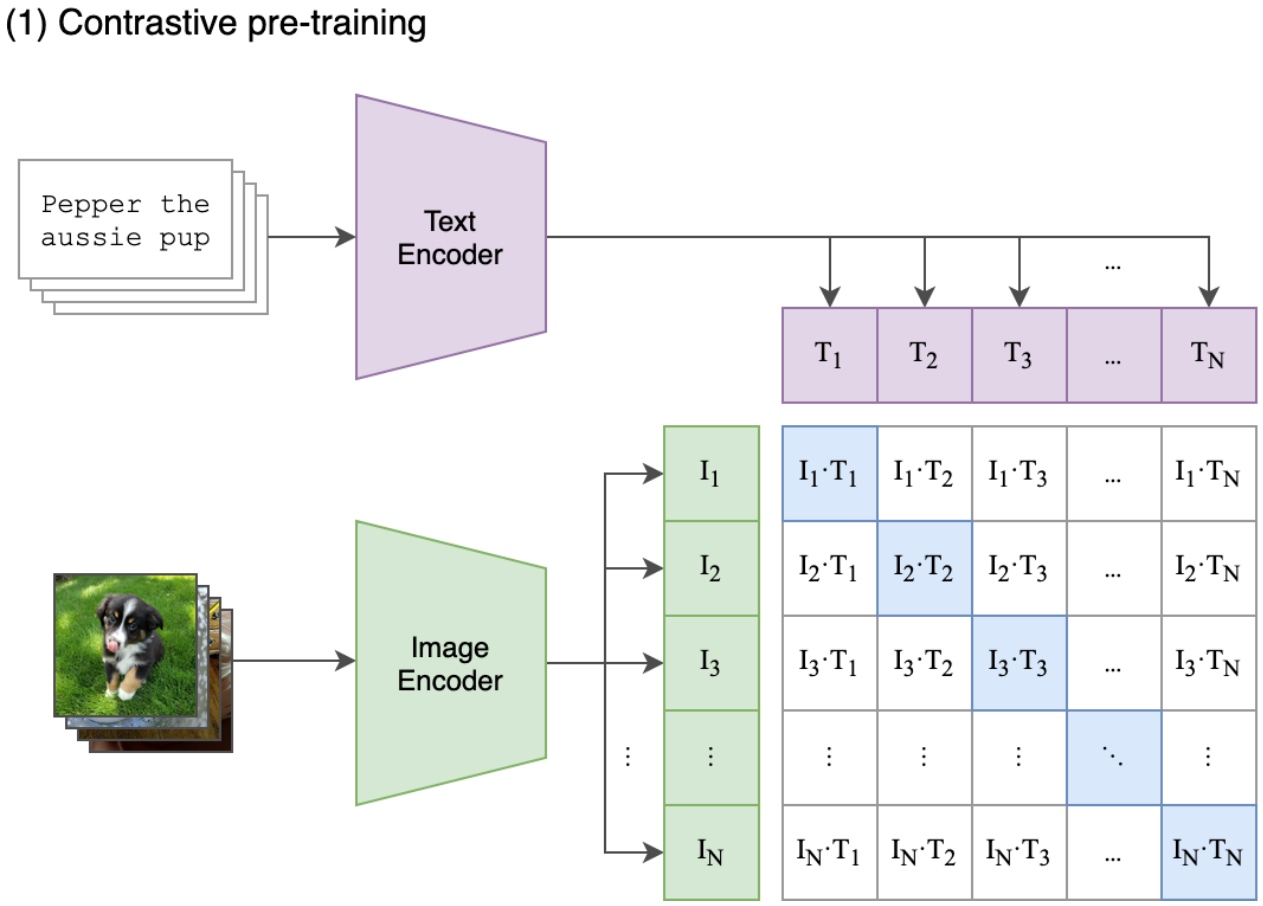
Foundational stage of CLIP's training process

Goal: learning to align images and texts in a shared embedding space without needing explicit labels for specific tasks



CLIP and zero-shot Learning

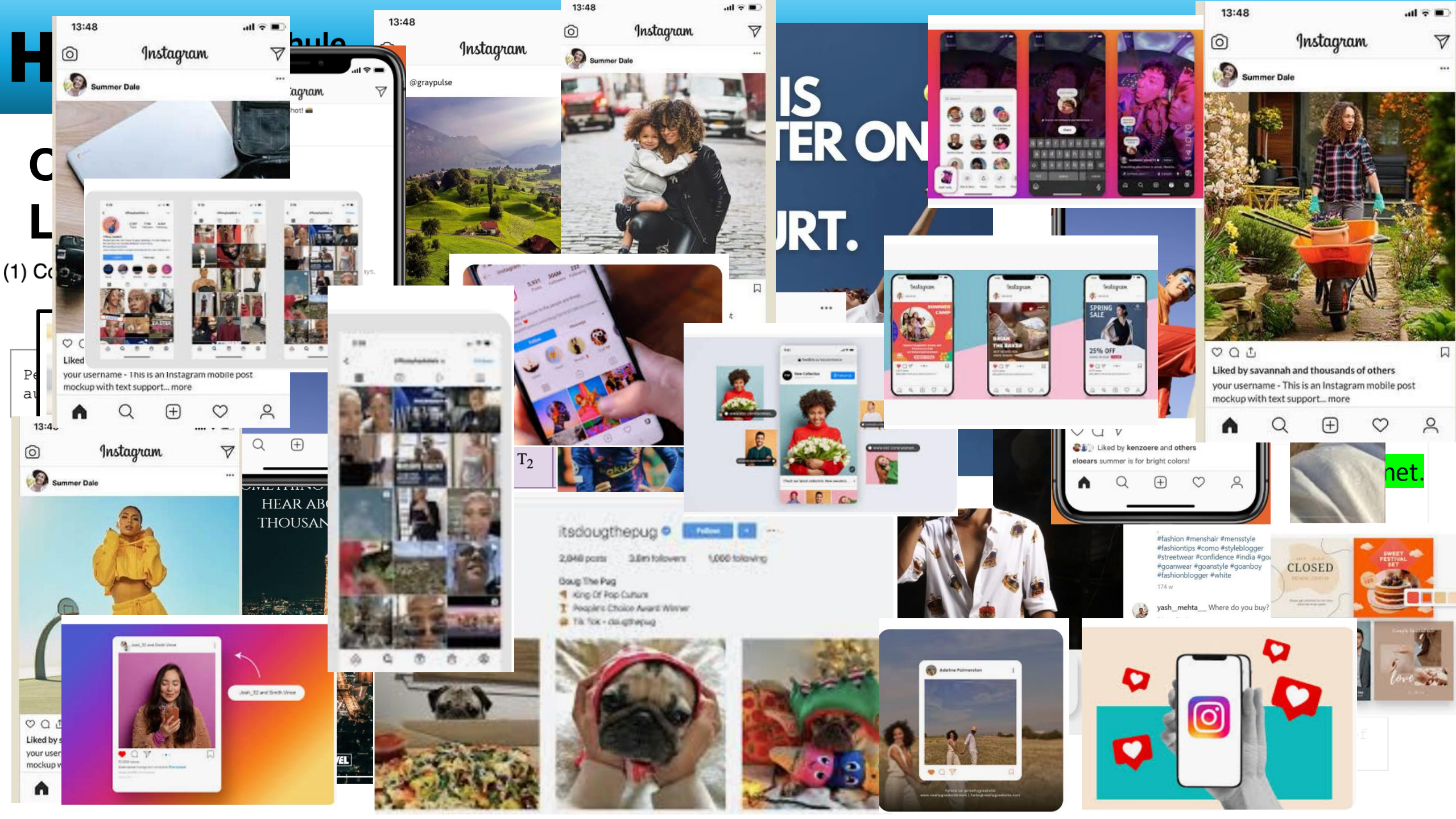
CLIP: Learning Transferable Visual Models From Natural Language Supervision (Paper)



Foundational stage of CLIP's training process

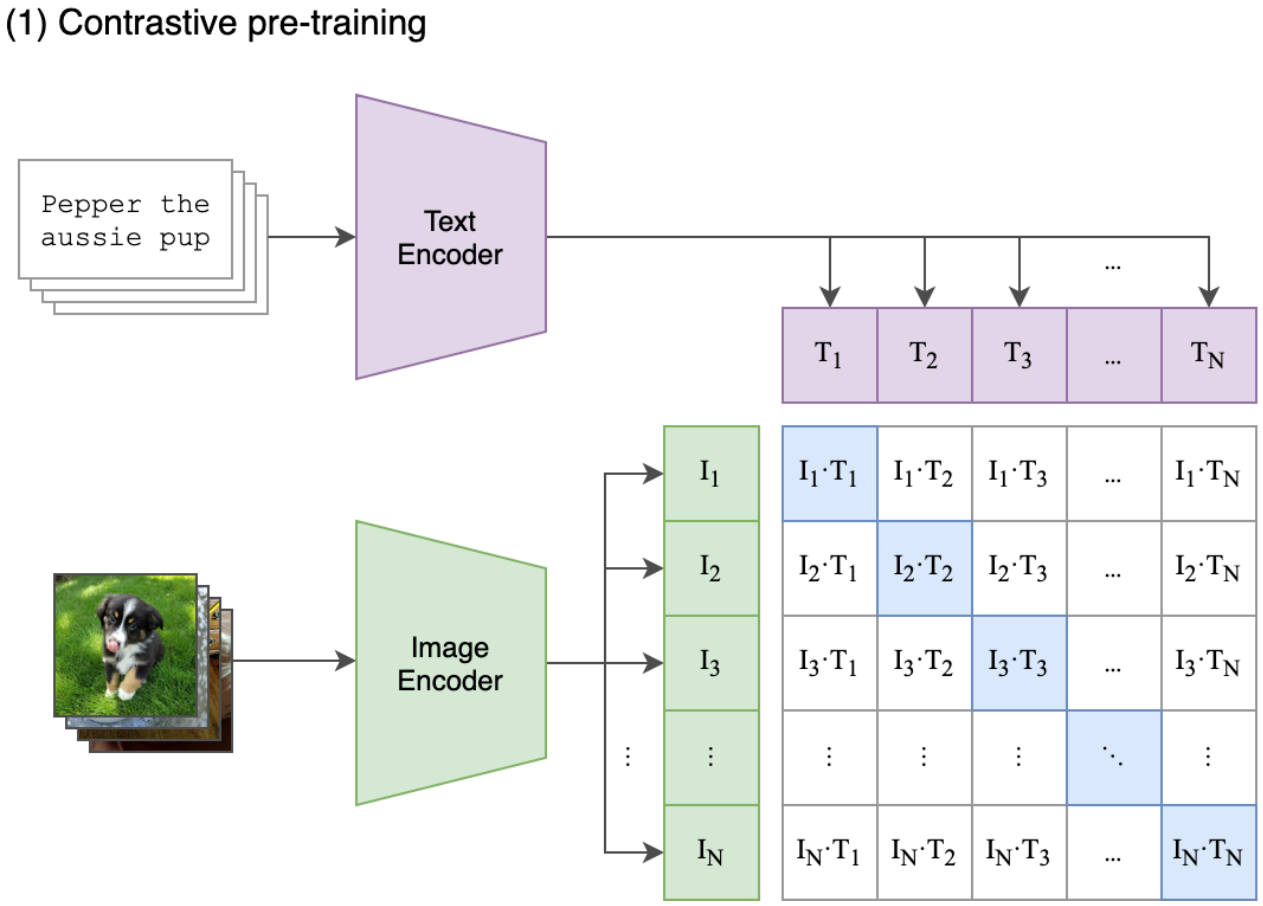
Goal: learning to align images and texts in a shared embedding space without needing explicit labels for specific tasks

DATASET: 400 million image-text pairs collected from Internet.


13:48
Instagram
Summer Dale
your username - This is an Instagram mobile post mockup with text support... more
@graypulse
Liked by savannah and thousands of others
your username - This is an Instagram mobile post mockup with text support... more
Liked by kenzoere and others
eloears summer is for bright colors!
SPRING SALE
25% OFF
CLOSED
itsdougthepug
Doug The Pug
#fashion #menshair #mensstyle #fashiontips #como #styleblogger #streetwear #confidence #india #goanwear #goanstyle #goanboy #fashionblogger #white
yash_mehta__ Where do you buy?

CLIP and zero-shot Learning

CLIP: Learning Transferable Visual Models From Natural Language Supervision (Paper)



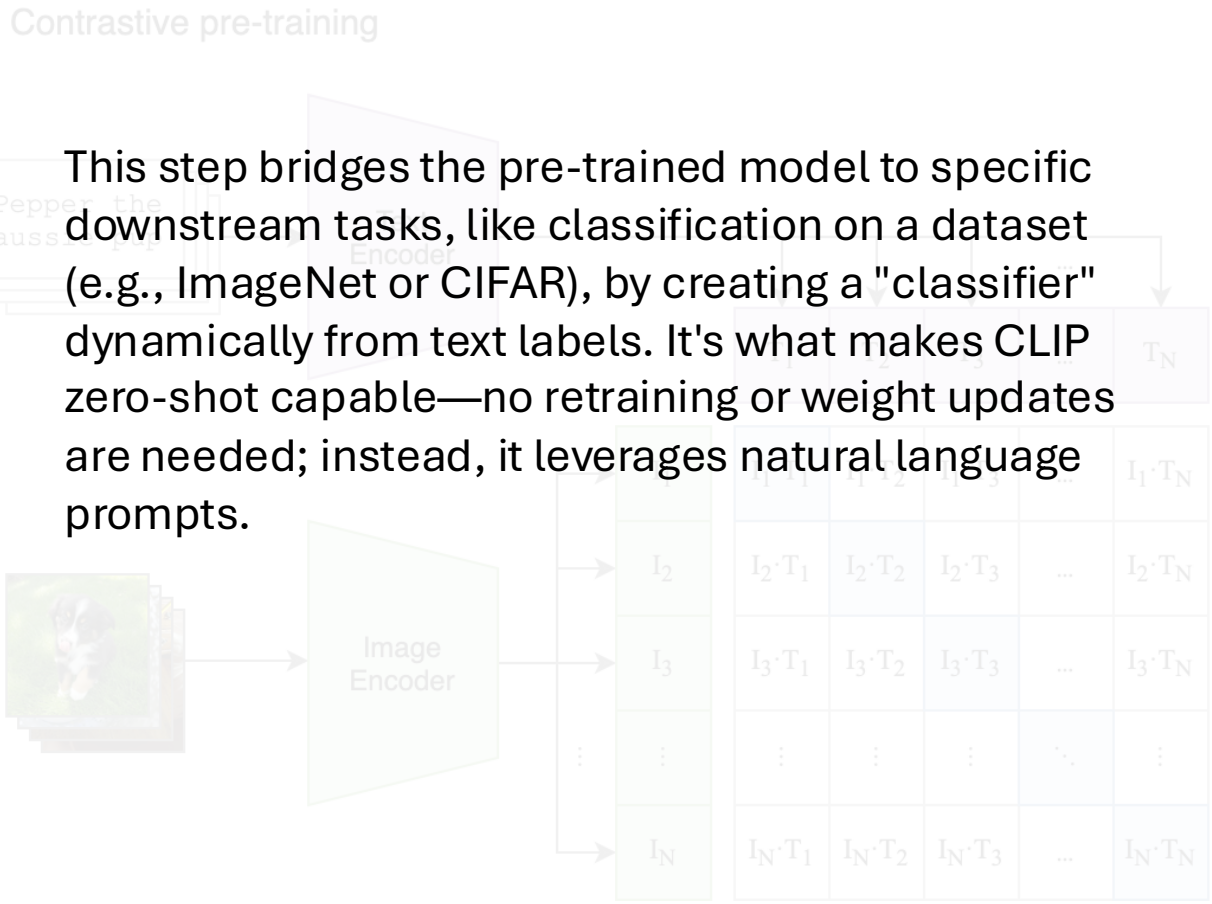
Foundational stage of CLIP's training process

Goal: learning to align images and texts in a shared embedding space without needing explicit labels for specific tasks

CLIP and zero-shot Learning

CLIP: Learning Transferable Visual Models From Natural Language Supervision (Paper)

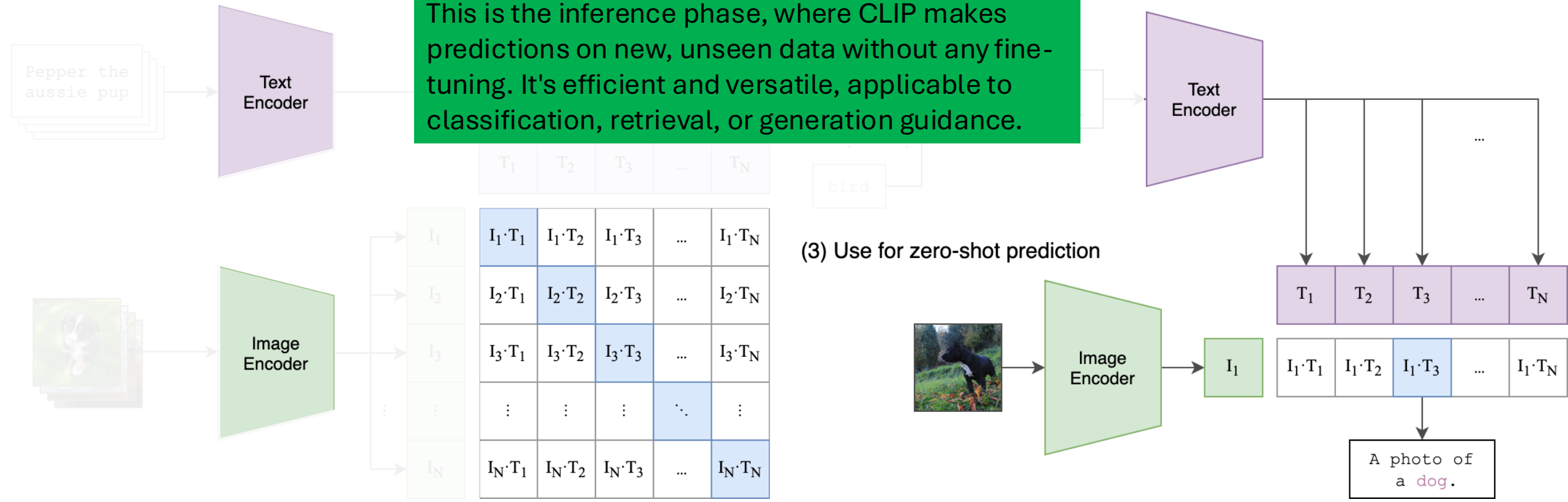
This step bridges the pre-trained model to specific downstream tasks, like classification on a dataset (e.g., ImageNet or CIFAR), by creating a "classifier" dynamically from text labels. It's what makes CLIP zero-shot capable—no retraining or weight updates are needed; instead, it leverages natural language prompts.



CLIP and zero-shot Learning

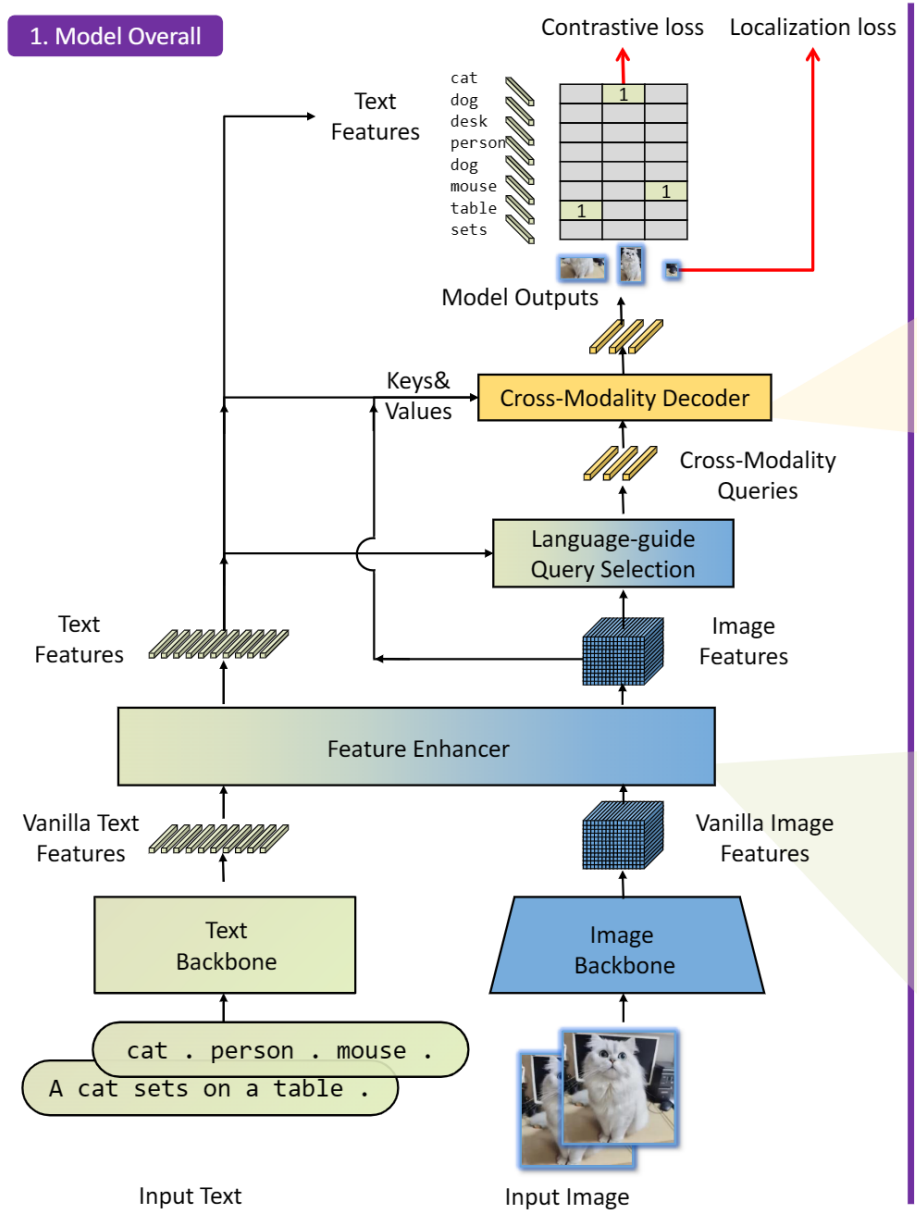
CLIP: Learning Transferable Visual Models From Natural Language Supervision (Paper)

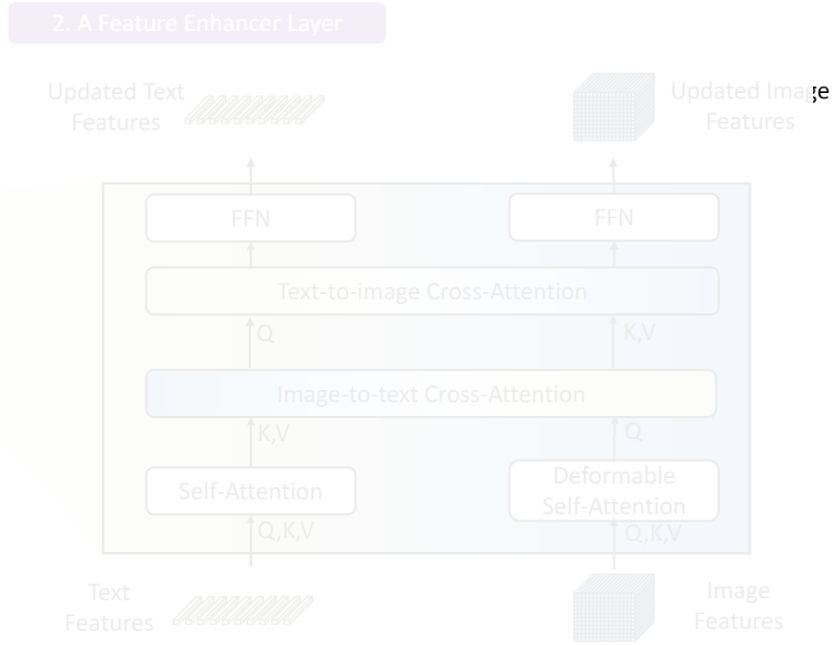
This is the inference phase, where CLIP makes predictions on new, unseen data without any fine-tuning. It's efficient and versatile, applicable to classification, retrieval, or generation guidance.



CLIP and zero-shot Learning

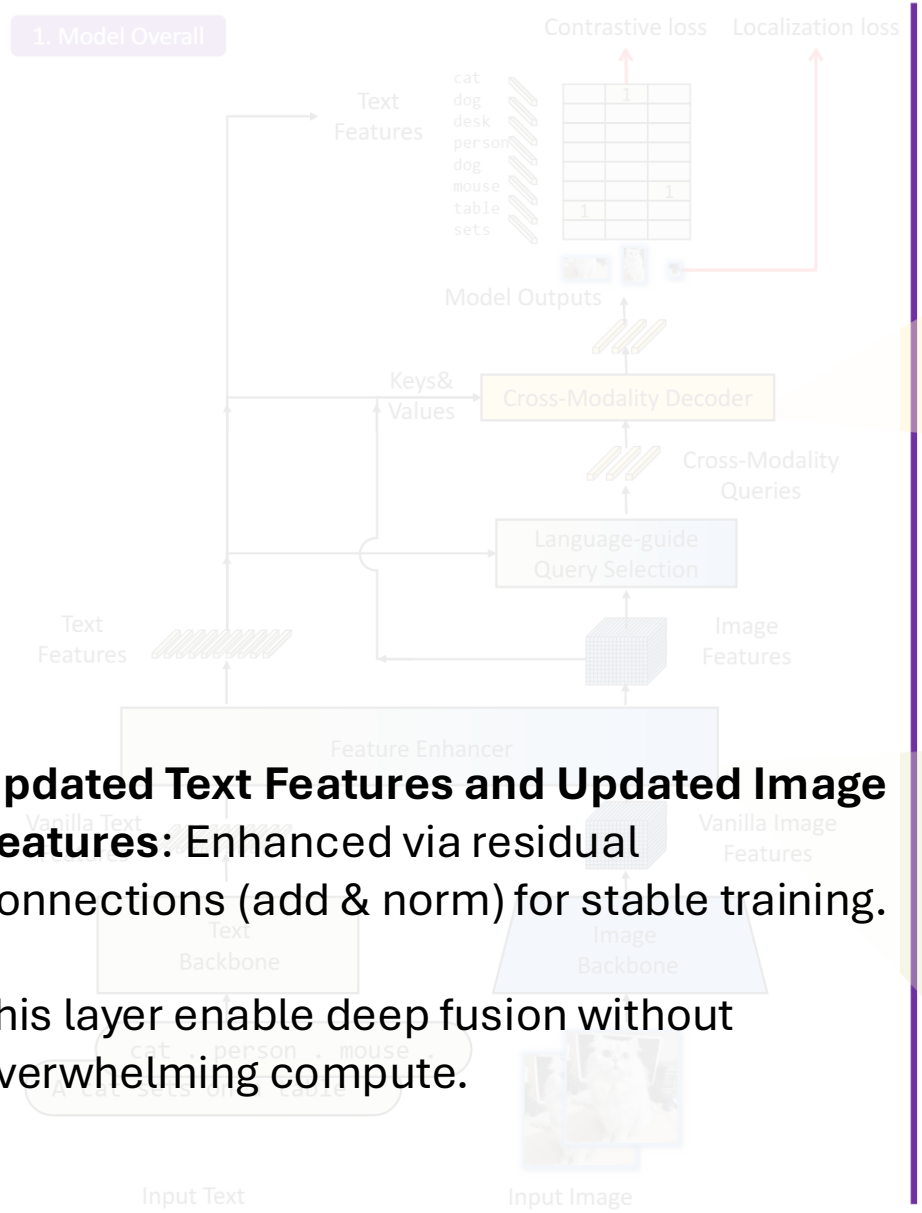
GroundingDINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection ([Paper])





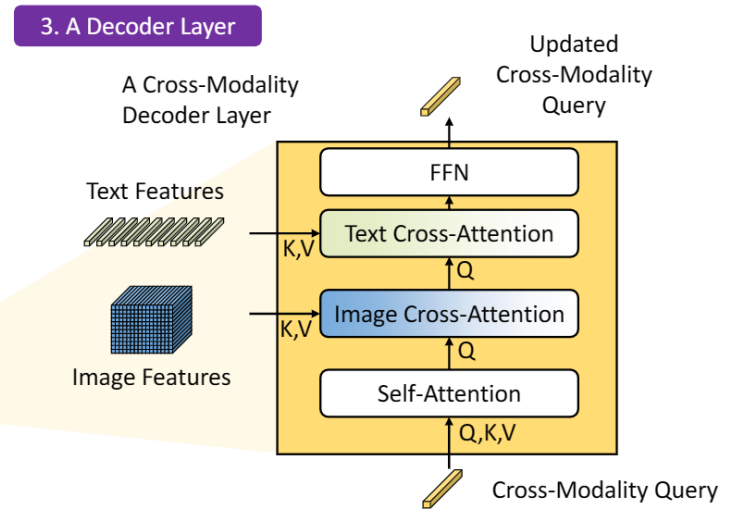
CLIP and zero-shot Learning

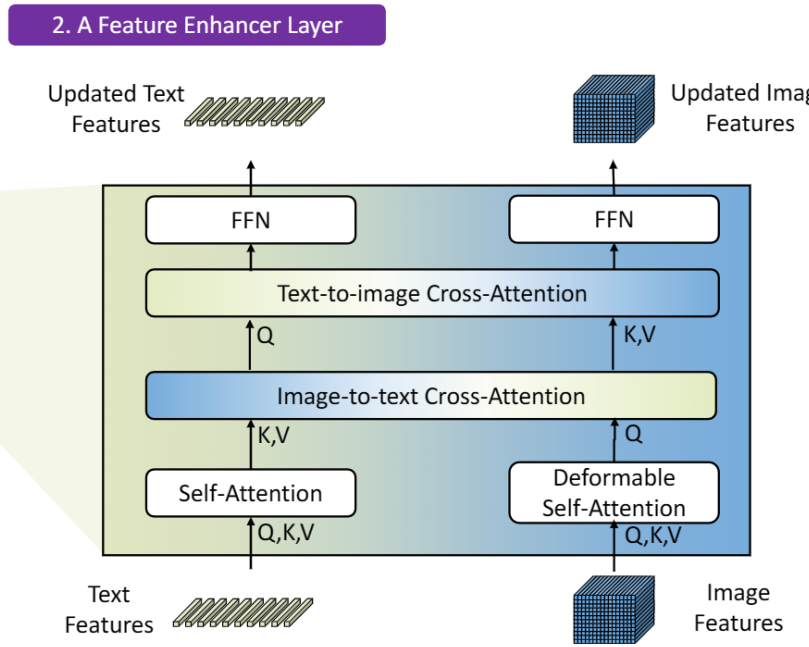
GroundingDINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection (Paper)



Updated Text Features and Updated Image Features: Enhanced via residual connections (add & norm) for stable training.

This layer enable deep fusion without overwhelming compute.





Hands-on:

(1) Pre-trained classification model

HSLU Hochschule Luzern

Hands-on:

(2) Visualization of features map

HSLU Hochschule Luzern

Hands-on:

(3) Finetuning ViT on classification task

HSLU Hochschule Luzern

Hugging Face

Hands-on:

(1) CLIP for zero-shot classification

OpenAI

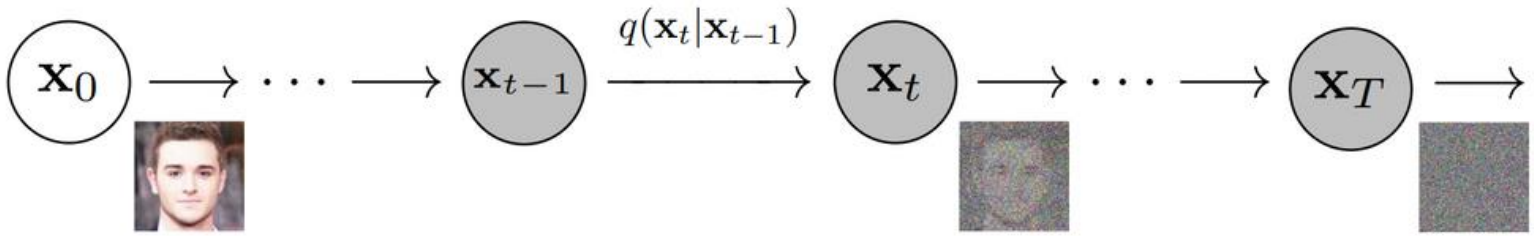
HSLU Hochschule Luzern

Diffusion Models

HSLU Hochschule Luzern

Diffusion Models

Diffusion Models work by destroying training data through the successive addition of Gaussian noise, and then learning to recover the data by reversing this noising process.



Forward diffusion process

$$q(x_1, \ldots, x_T|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1})$$

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

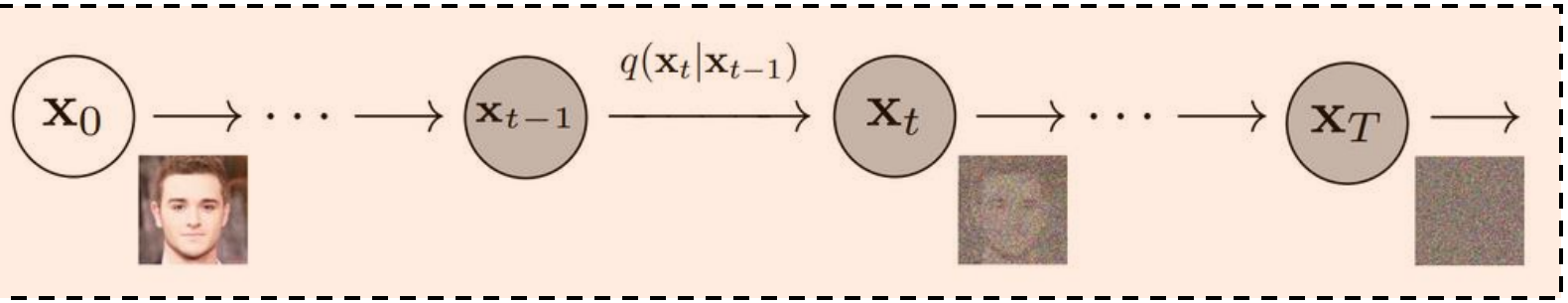
distribution of the noised images | Output | Mean μ_t | Variance Σ_t

- ***t***: This represents the **time step**. It goes from a clear image at t=0 to a completely noisy image at t=T.
- ***x_0***: This is your **original, clean data sample**. It's the starting point of the whole process.
- ***β_t***: This is the **variance schedule**. It's a list of small numbers that control how much noise is added at each step. (1) It's a fixed sequence (not learned). (2) it typically starts with a very small number ($\beta 0 \approx 0$) so the first few steps add very little noise. (3) It gradually increases to a larger number ($\beta_t \approx 1$), so the later steps add a lot of noise. (4) The values of βt are always between 0 and 1.
- **I**: This is the **identity matrix**. It ensures that the noise is added independently to each pixel of the image, without any correlation between pixels.

Diffusion Models

Diffusion Models work by destroying training data through the successive addition of Gaussian noise, and then learning to recover the data by reversing this noising process.

Markov chain



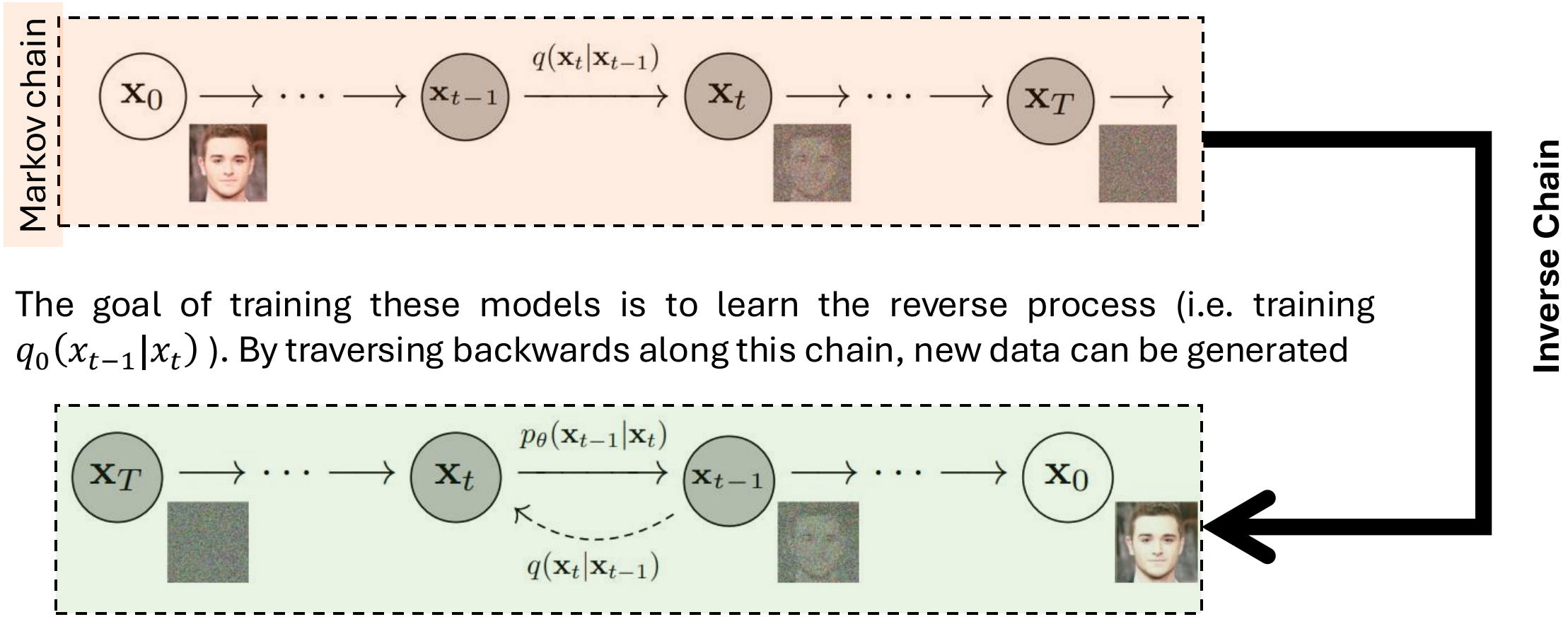
Ultimately, the image is asymptotically transformed to pure Gaussian noise.

The goal of training these models is to learn the reverse process (i.e. training $q_0(x_{t-1}|x_t)$). By traversing backwards along this chain, new data can be generated

Diffusion Models:A Comprehensive Survey of Methods and Applications. LING YANG *et. al. Feb. 2024* arXiv:2209.00796

Diffusion Models

Diffusion Models work by destroying training data through the successive addition of Gaussian noise, and then learning to recover the data by reversing this noising process.



The goal of training these models is to learn the reverse process (i.e. training $q_0(x_{t-1}|x_t)$). By traversing backwards along this chain, new data can be generated

Diffusion Models:A Comprehensive Survey of Methods and Applications. LING YANG *et. al. Feb. 2024* arXiv:2209.00796

Diffusion Models

What model can learn to predict inversing noise ?

Authors selected a ***U-Net*** like architecture

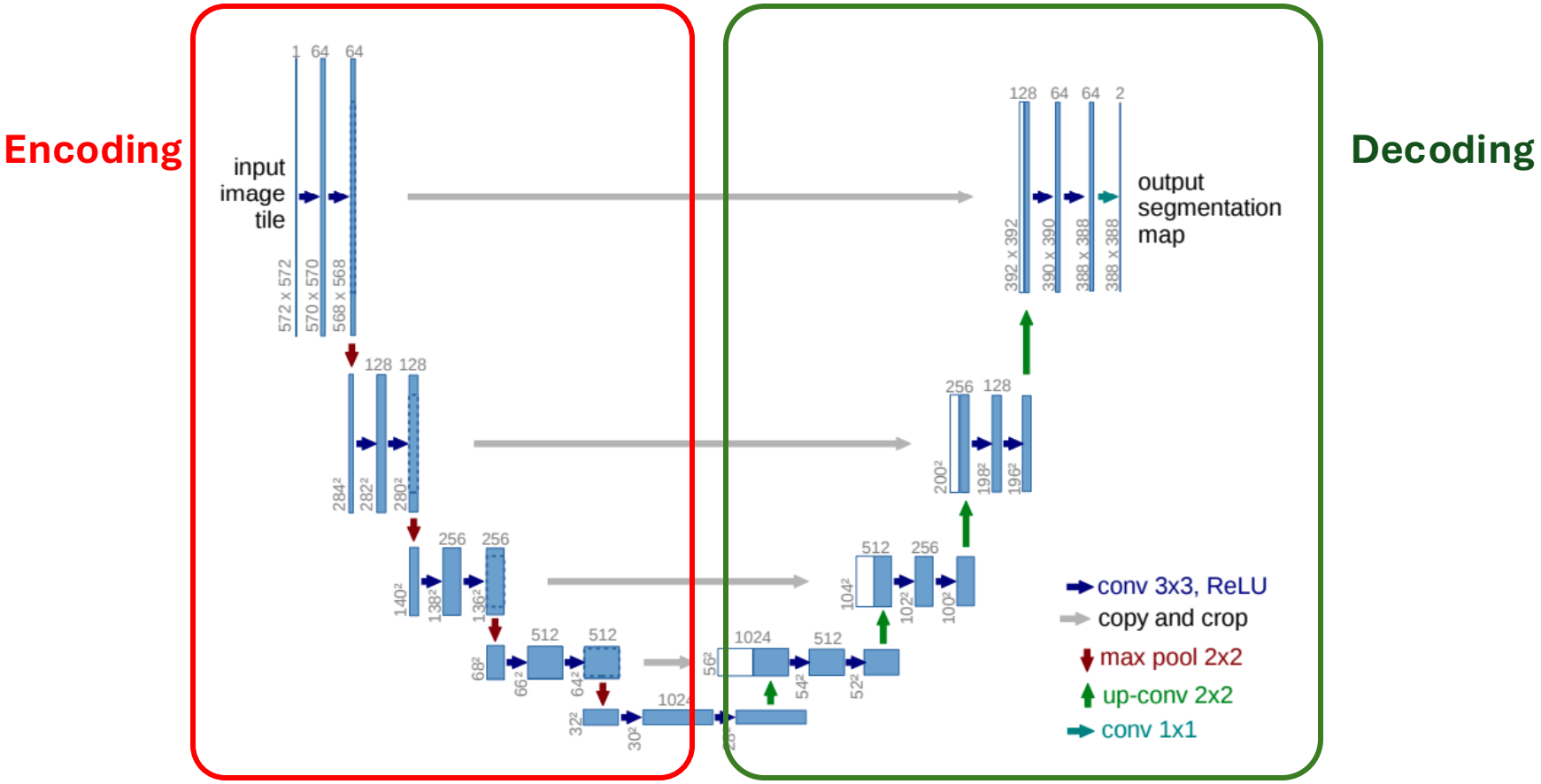


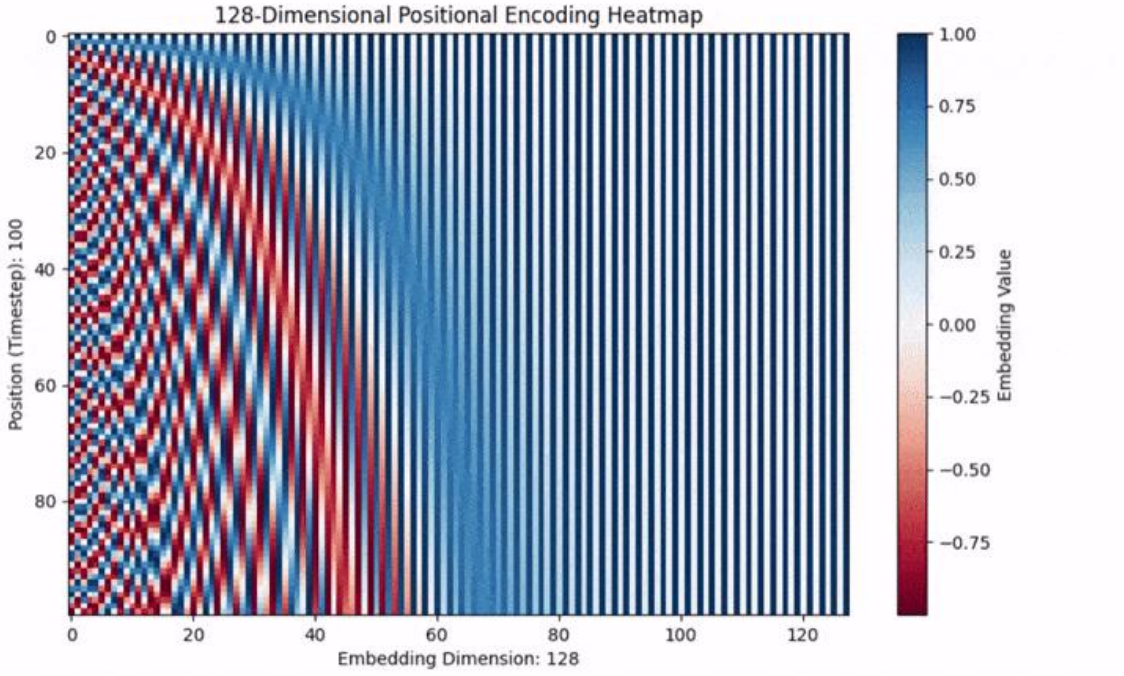
Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Diffusion Models

U-Net components

1. Sinusoidal time embedding:

The Time Embedding encodes this integer t into a continuous, fixed-dimensional vector (dim=128) using sinusoidal positional encodings. This is inspired by transformer models [1]Vaswani et al., 2017, where positions in sequences need unique, learnable representations. Without it, the model couldn't distinguish between different noise levels, leading to poor denoising.



This creates a unique, smooth representation for each $t \in [0, T-1]$,helping the model condition on the noise level

[1]Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, & Illia Polosukhin. (2023). Attention Is All You Need.

Diffusion Models

U-Net components

1. Sinusoidal time embedding:

The Time Embedding encodes this integer *t* into a continuous, fixed-dimensional vector (dim=128) using sinusoidal positional encodings. This is inspired by transformer models [1]Vaswani et al., 2017, where positions in sequences need unique, learnable representations. Without it, the model couldn't distinguish between different noise levels, leading to poor denoising.

```python
class TimeEmbedding(nn.Module):
    """Sinusoidal time embedding."""
    def __init__(self, dim):
        super().__init__()
        self.dim = dim

    def forward(self, time):
        device = time.device
        half_dim = self.dim // 2
        embeddings = math.log(10000) / (half_dim - 1)
        embeddings = torch.exp(torch.arange(half_dim, device=device) * -embeddings)
        embeddings = time[:, None] * embeddings[None, :]
        embeddings = torch.cat([embeddings.sin(), embeddings.cos()], dim=-1)
        return embeddings
```

[1]Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, & Illia Polosukhin. (2023). Attention Is All You Need.

Diffusion Models

U-Net components

2. Simplified Residual Network:

building block for the U-Net, designed as a residual module similar to those in ResNet architectures ([2]Kaiming He et al.).

In diffusion models, the network processes noisy images at various resolutions, and these blocks allow stacking multiple layers without suffering from vanishing gradients or training instability.

[2]Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. (2015). Deep Residual Learning for Image Recognition.

Diffusion Models

U-Net components

2. Simplified Residual Network:

building block for the U-Net, designed as a residual module similar to those in ResNet architectures ([2]Kaiming He et al.).

In diffusion models, the network processes noisy images at various resolutions, and these blocks allow stacking multiple layers without suffering from vanishing gradients or training instability.

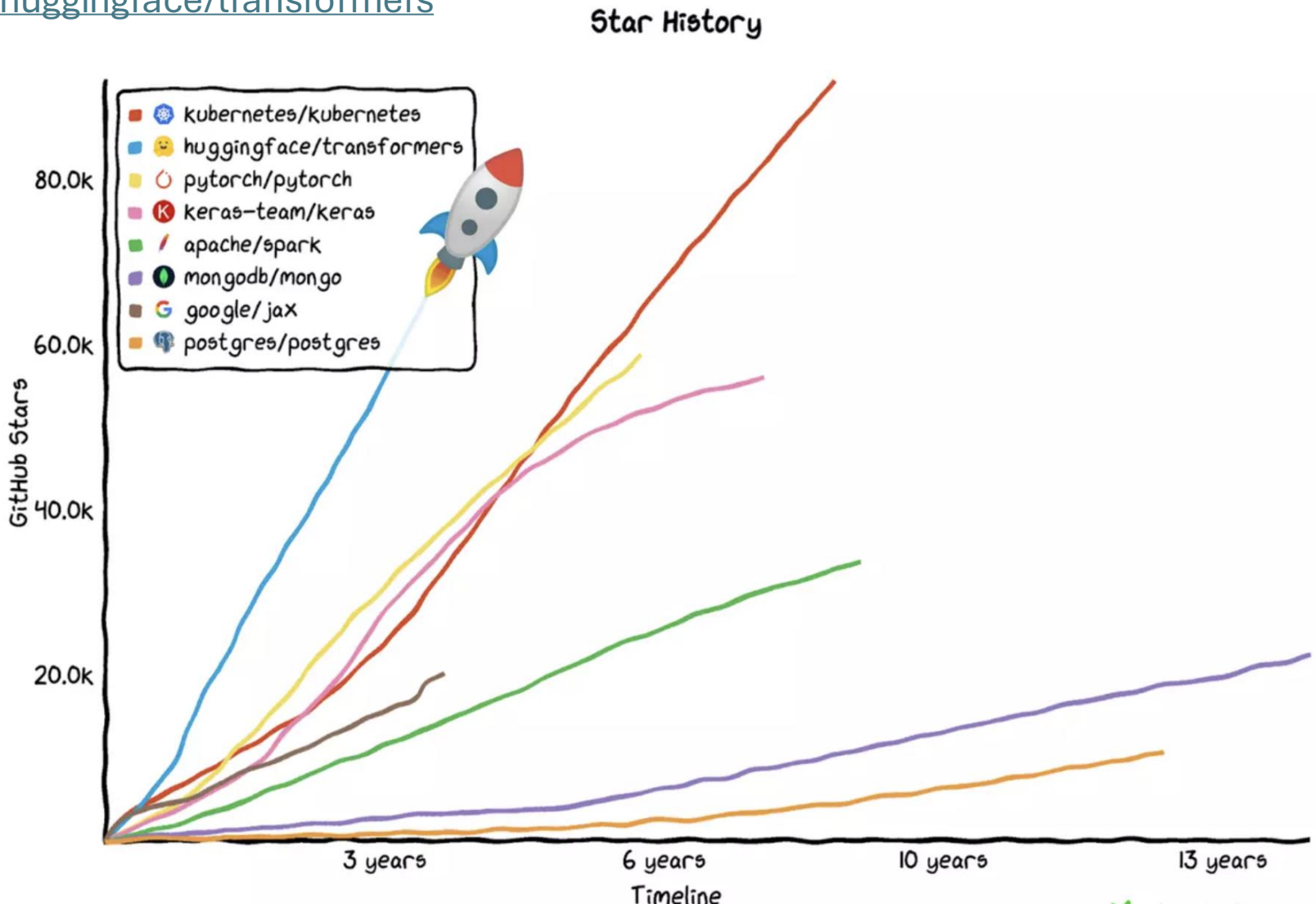
[2]Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. (2015). Deep Residual Learning for Image Recognition.

Hugging Face Transformers library

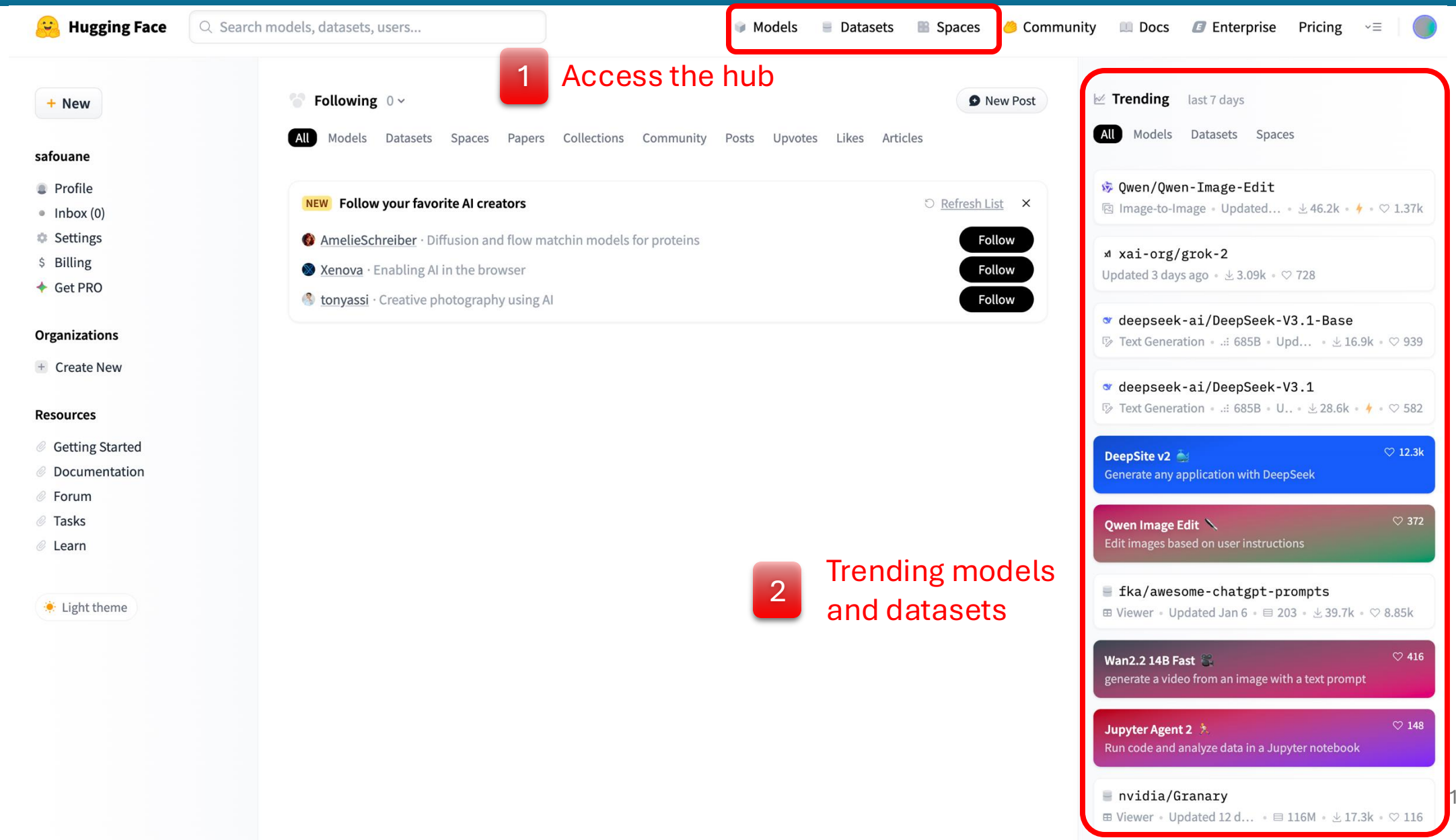
Transformers library

Transformers is one of the fastest growing libraries for open source projects

https://github.com/huggingface/transformers



Transformers library



HSLU Hochschule Luzern

Limitations of ViTs

Limitations of ViTs

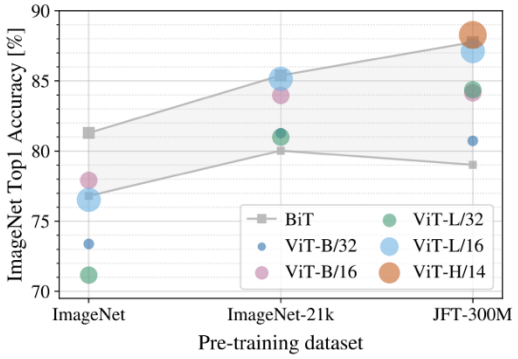


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

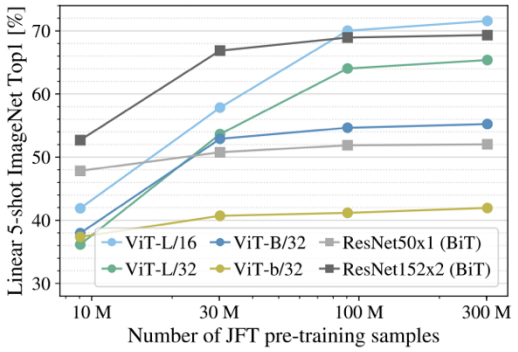


Figure 4: Linear few-shot evaluation on ImageNet versus pre-training size. ResNets perform better with smaller pre-training datasets but plateau sooner than ViT, which performs better with larger pre-training. ViT-b is ViT-B with all hidden dimensions halved.

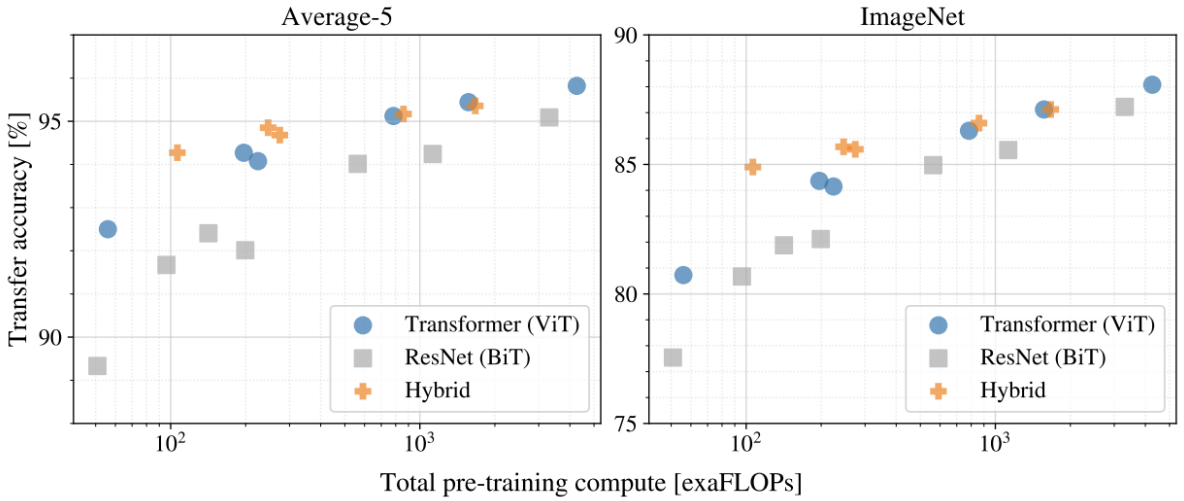


Figure 5: Performance versus pre-training compute for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

Limitations of ViTs

- **High Data Requirements**: ViTs are highly data-hungry and usually need massive datasets for effective training.

- **Computational and Memory Intensity**: The self-attention mechanism in ViTs is computationally expensive and memory-intensive, particularly for high-resolution images, leading to higher processing demands and potential scalability issues.

- **Challenges with Local Textures and High-Frequency Details**: ViTs often struggle to capture high-frequency components or local textures in images, where CNNs excel due to their convolutional operations.

- **Prone to Overfitting on Small Datasets**: Without sufficient data, ViTs are susceptible to overfitting, limiting their applicability in scenarios with limited training samples.

HSLU Hochschule Luzern

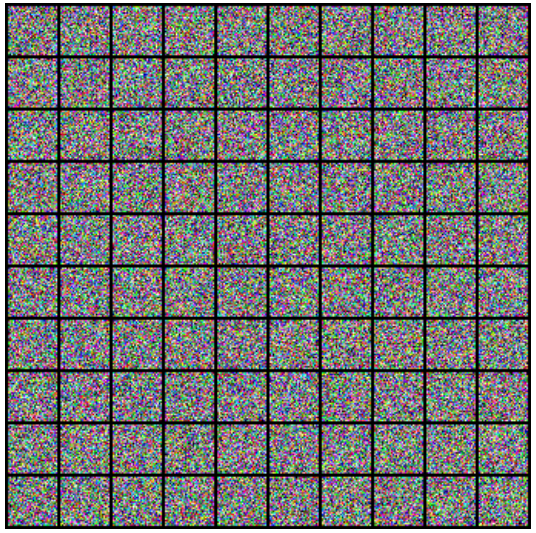
Background & Motivations

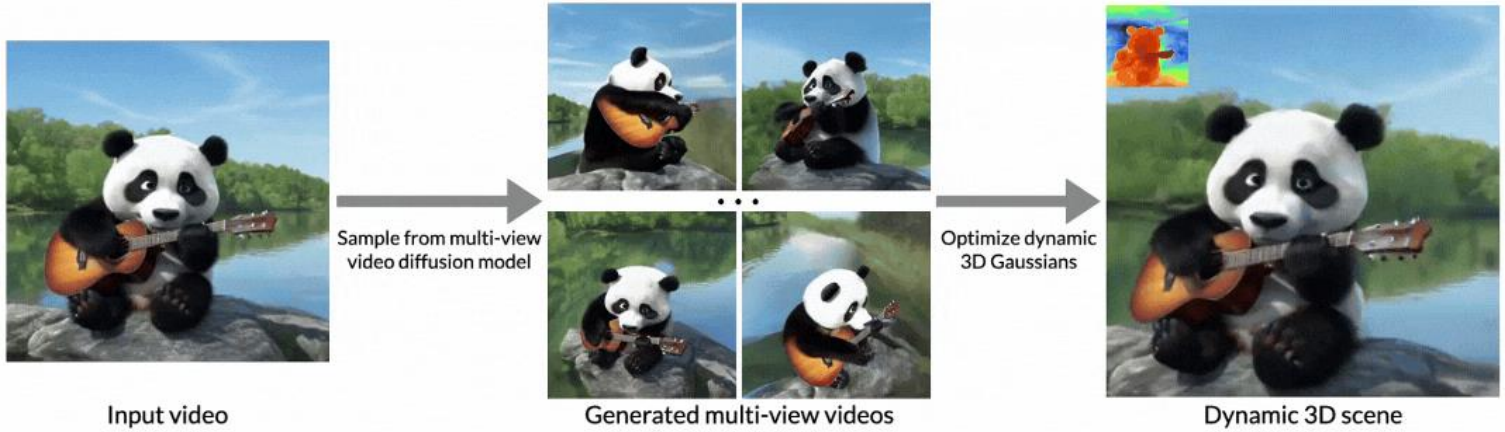
Background & Motivations

Introduction to Generative AI Fundamentals: Previous generative AI models rely on GAN models. Attention-Generative AI as models that create new data (e.g., images, text) from learned patterns in training data, contrasting it with discriminative models.

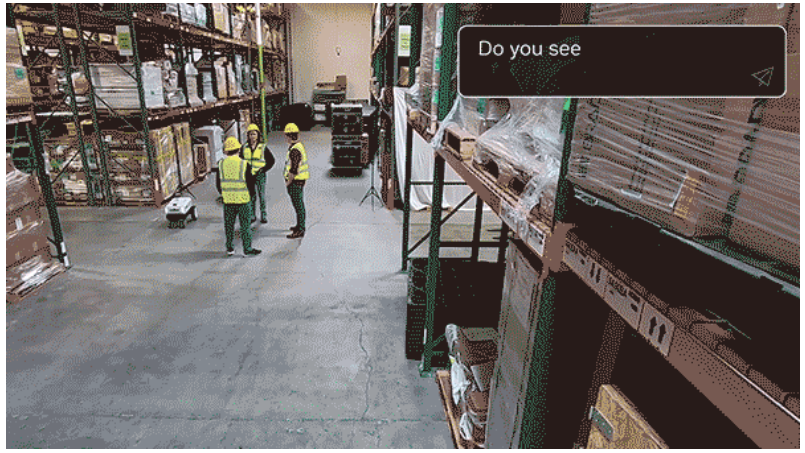
Motivations: solving data scarcity (e.g., synthetic data for training), creativity (art, design), personalization (e.g., custom content), and efficiency (automating tasks like content creation). Highlight real-world drivers like computational power growth, large datasets (e.g., LAION-5B), and ethical considerations (e.g., bias in generated outputs).

Synthetic image creation





Captioning/VQA



HSLU Hochschule Luzern

Applications of Vision Transformers in Gen AI

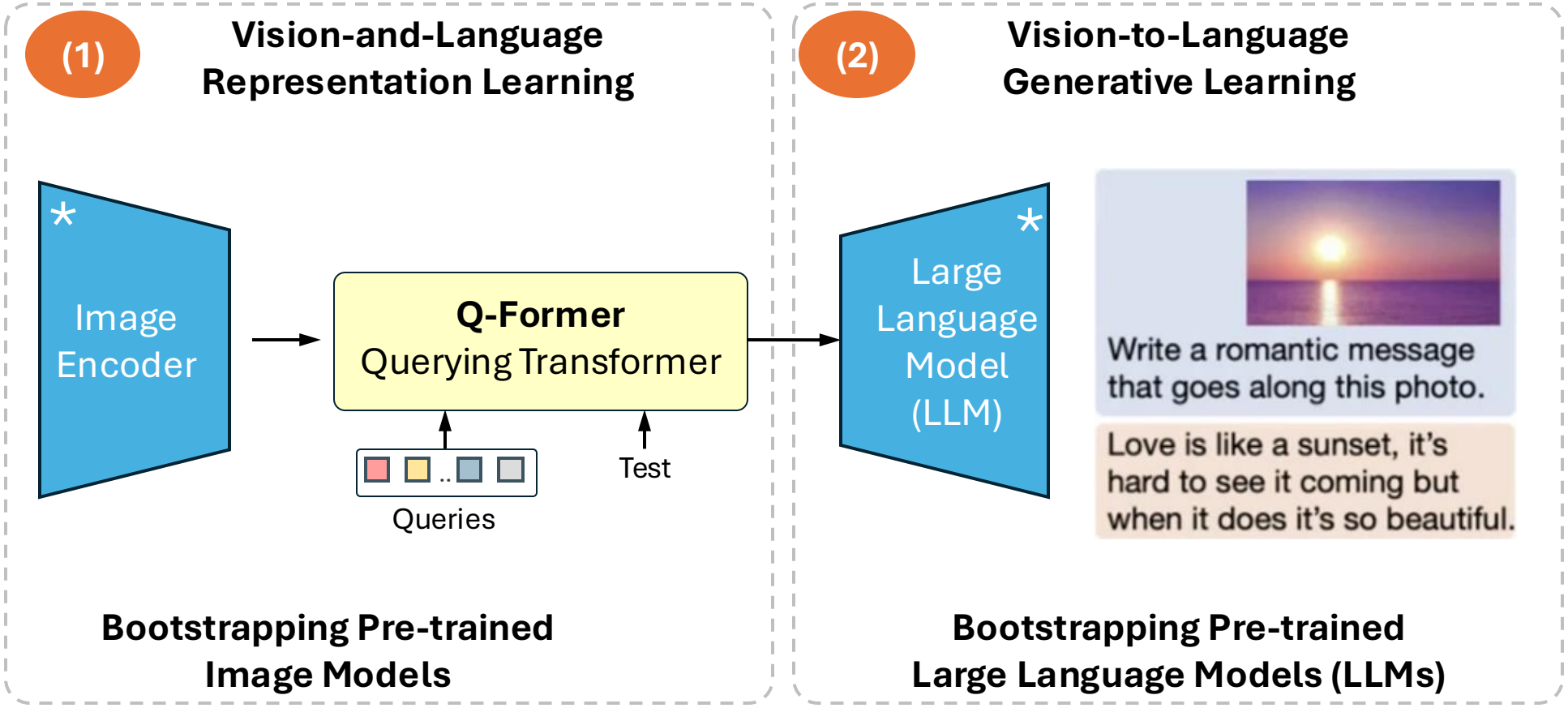
HSLU Hochschule Luzern

BLIP and multimodal generation

BLIP and Multimodal generation

BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models (Paper)

Pre-train a light weight Q-Former in two stages:



BLIP and Multimodal generation

	Concept	Paper
BLIP	A vision-language pretraining framework that uses a bootstrapped approach to generate and filter captions from noisy web data, enabling unified vision-language understanding and generation tasks like image captioning. It combines a vision encoder and text decoder for efficient captioning.	BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation
IDEFICS	A multimodal model designed for vision and language tasks, including image captioning, built by fine-tuning open-access models like LLaVA and CLIP-ViT. It processes images and text jointly, leveraging a large-scale pretraining dataset for robust performance.	IDEFICS: An Open-Source Visual Language Model
GIT	A generative image-to-text transformer that excels in image captioning by directly generating text from image inputs. It uses a single vision transformer and a text decoder, optimized for lightweight and efficient captioning with performance comparable to BLIP-2.	GIT: A Generative Image-to-text Transformer for Vision and Language
InstructBLIP	An enhanced version of BLIP-2, fine-tuned with instruction-tuning datasets to improve zero-shot and few-shot performance in image captioning and other vision-language tasks. It leverages BLIP-2's architecture with additional task-specific optimizations.	InstructBLIP: Towards General-purpose Vision-Language Models with Instruction Tuning
COCA	A contrastive captioning model that combines contrastive learning with generative captioning. It uses image-text alignment to produce high-quality captions, offering performance close to BLIP-2 with lower computational requirements.	CoCa: Contrastive Captioners are Image-Text Foundation Models

Hugging Face

Hands-on:

(1) BLIP image captioning

HSLU Hochschule Luzern

Hugging Face

Hands-on:

(2) Stable diffusion

OpenAI

Computer Vision – Fundamentals

Introduction To Generative AI In Vision

Program directors:

Dr. Prof.
Umberto Michelucci

Dr.
Aygul Zagidullina

Dr.
Safouane El Ghazouali

HSLU Lucerne University of Applied Sciences and Arts