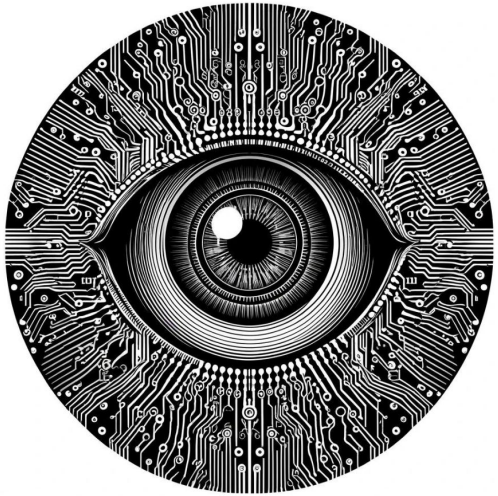
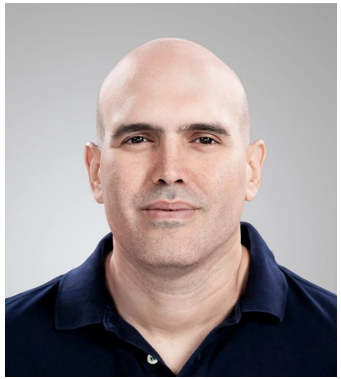
Workshop 12 - Image Generation with Diffusion Models



Antonio Rueda-Toicen

About me

- AI Engineer & DevRel for Voxel51, Researcher at the HPI
- Organizer of the Berlin Computer Vision Group
- Instructor at Nvidia's Deep Learning Institute and Berlin's Data Science Retreat
- Made a MOOC on Practical Computer Vision for OpenHPI



LinkedIn

How to use our Discord channel during the workshop

- Our channel is **#practical-computer-vision-workshops.** Please ask all questions there instead of the Zoom chat. Through Discord we can have better and more detailed discussions.

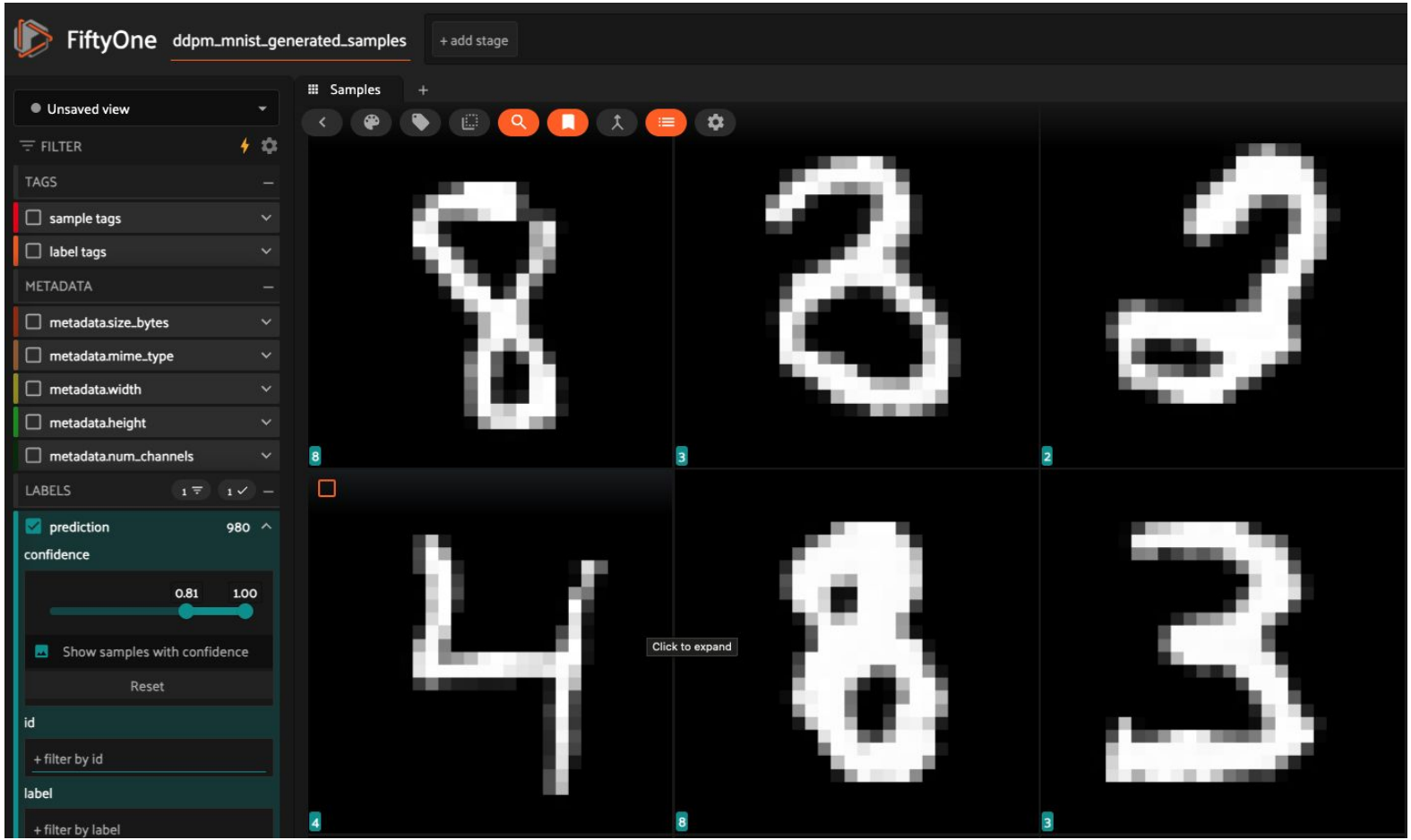
- **Step 1 - Use the Discord invite on the Voxel51 website** https://discord.com/invite/fiftyone-community
- **Step 2 - Access channel** (use the direct link below or search) http://bit.ly/3YmvPXG

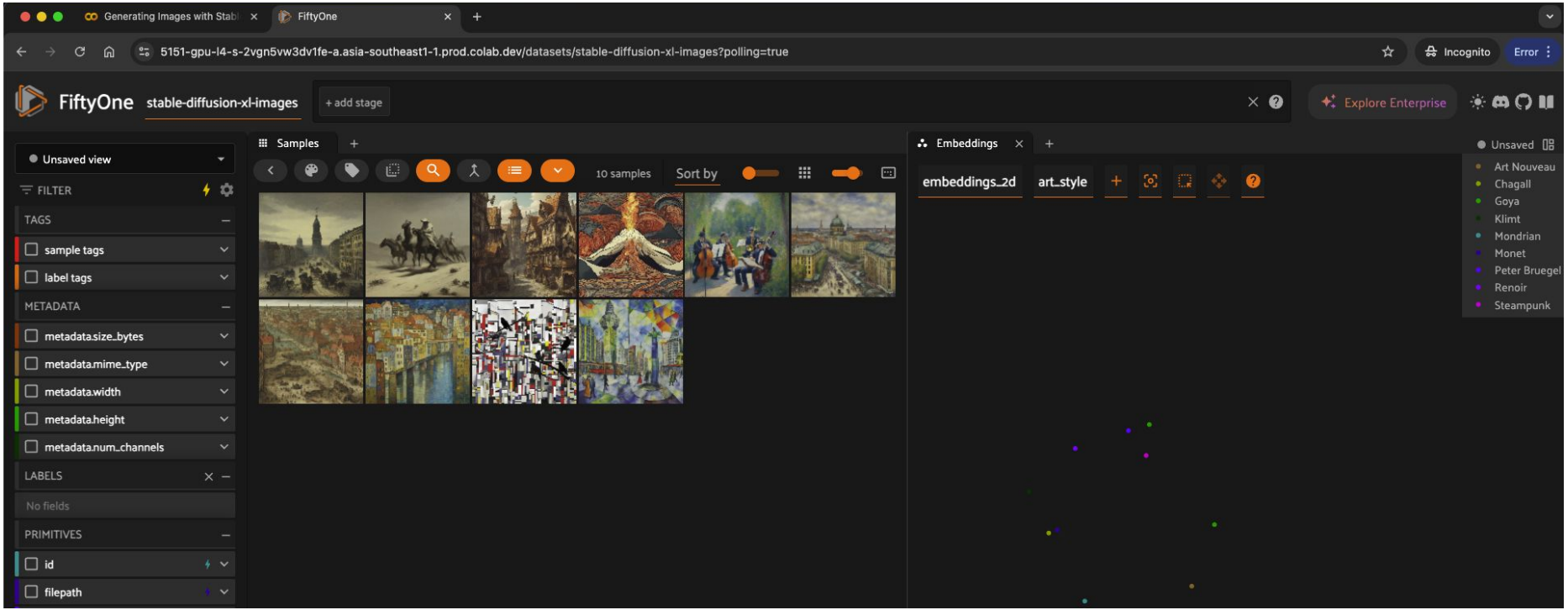
Agenda

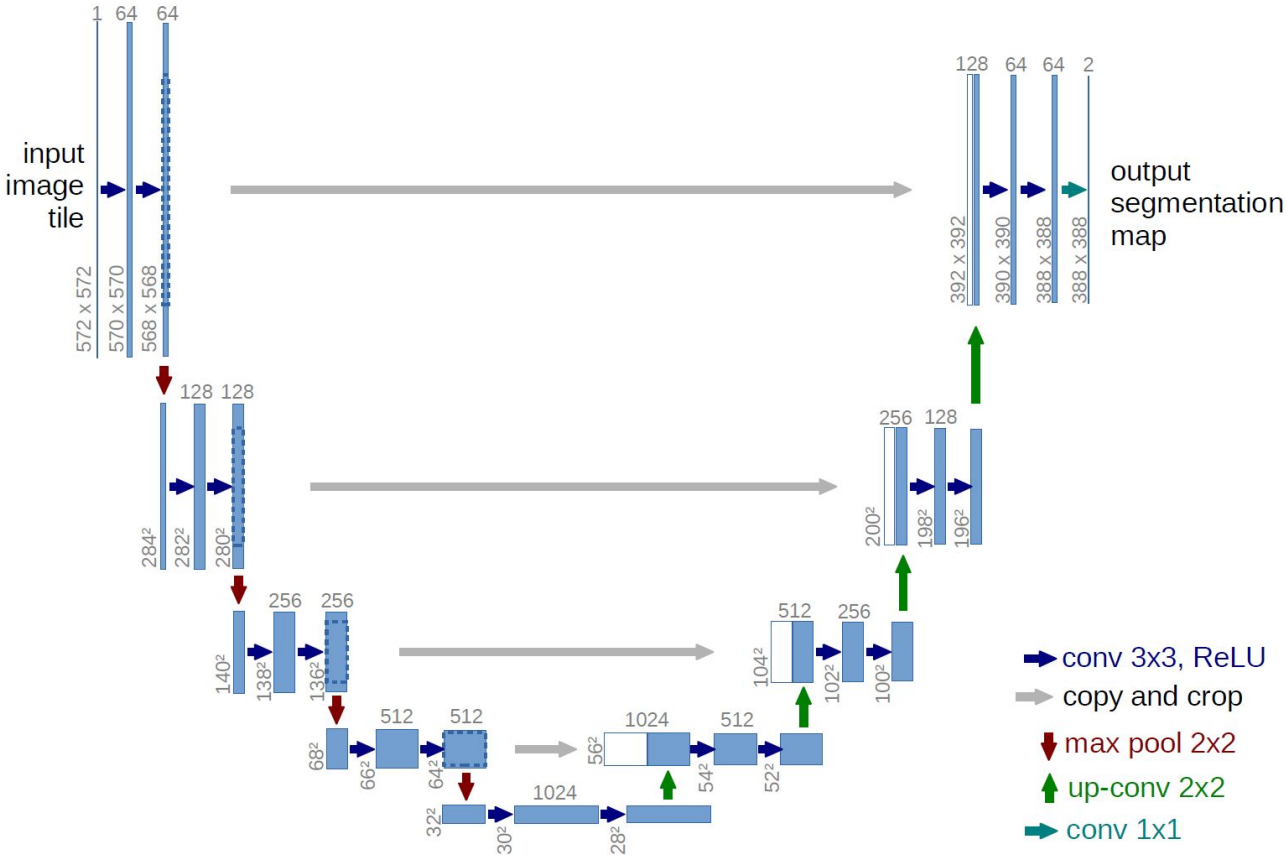
- Upsampling and Channel Mixing with Convolutions (review)
- Contrastive Language-Image Pretraining (CLIP) (review)
- Image Generation with Diffusion Models

Notebooks

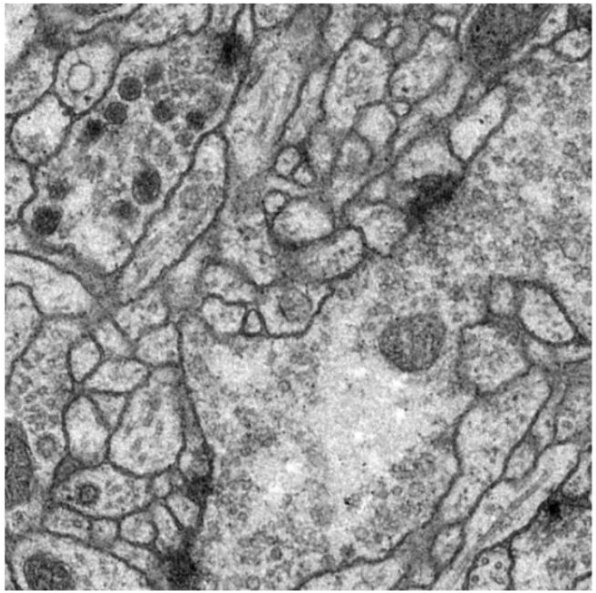
- U-net training (review)
- Denoising Diffusion Probabilistic Model with U-net
- Artwork generation with Stable Diffusion XL


FiftyOne
ddpm_mnist_generated_samples
+ add stage
Samples
Unsaved view
FILTER
TAGS
sample tags
label tags
METADATA
metadata.size_bytes
metadata.mime_type
metadata.width
metadata.height
metadata.num_channels
LABELS
prediction
980
confidence
0.81
1.00
Show samples with confidence
Reset
id
+ filter by id
label
+ filter by label
Click to expand

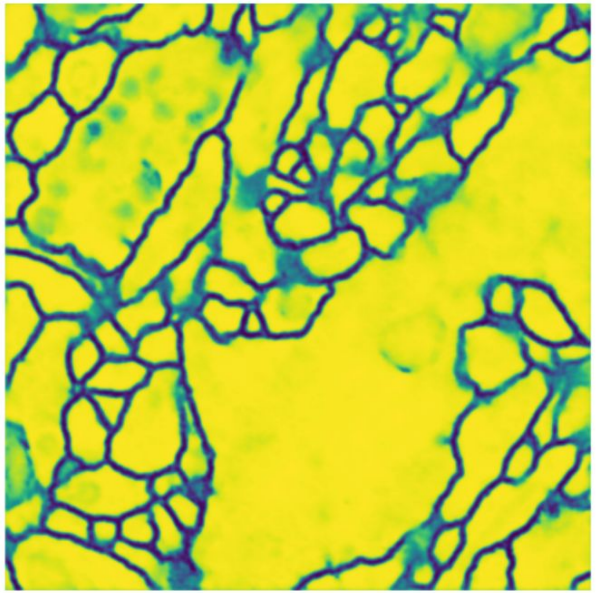

Generating Images with Stabl
FiftyOne
5151-gpu-l4-s-2vgn5vw3dv1fe-a.asia-southeast1-1.prod.colab.dev/datasets/stable-diffusion-xl-images?polling=true
Incognito
Error
FiftyOne
stable-diffusion-xl-images
+ add stage
Explore Enterprise
Unsaved view
FILTER
TAGS
sample tags
label tags
METADATA
metadata.size_bytes
metadata.mime_type
metadata.width
metadata.height
metadata.num_channels
LABELS
No fields
PRIMITIVES
id
filepath
Samples
10 samples
Sort by
Embeddings
embeddings_2d
art_style
Unsaved
Art Nouveau
Chagall
Goya
Klimt
Mondrian
Monet
Peter Bruegel
Renoir
Steampunk


input image tile
1
64
64
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512
256
104²
102²
100²
256
128
200²
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Test Image (Input)

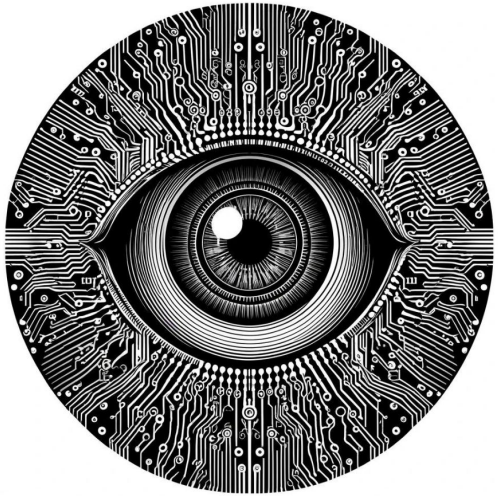


Prediction Probabilities



Prediction (F1 Threshold=0.52)

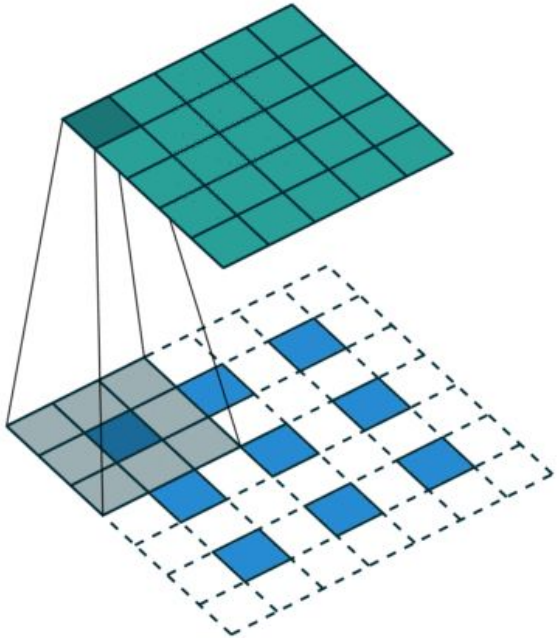
Upsampling and Channel Mixing with Convolutions

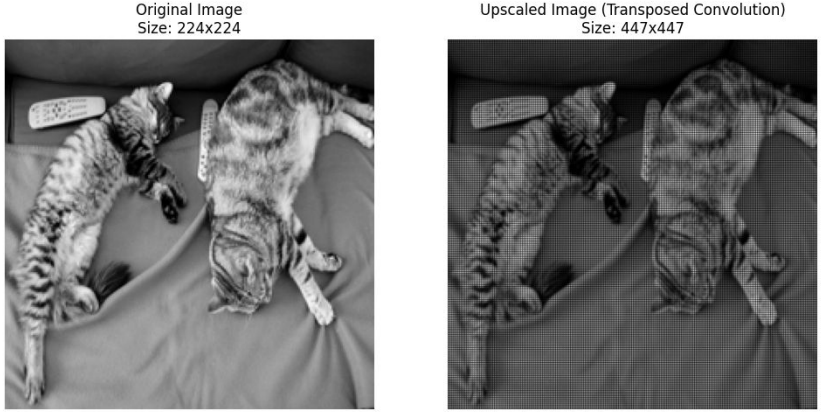


Learning goals

- Master upsampling and channel mixing with convolutions
- Use bilinear interpolation to resize images
- Understand hypercolumns and feature combination for image generation

Upsampling filters (aka transposed convolution / upconvolution / atrous convolution / deconvolution)



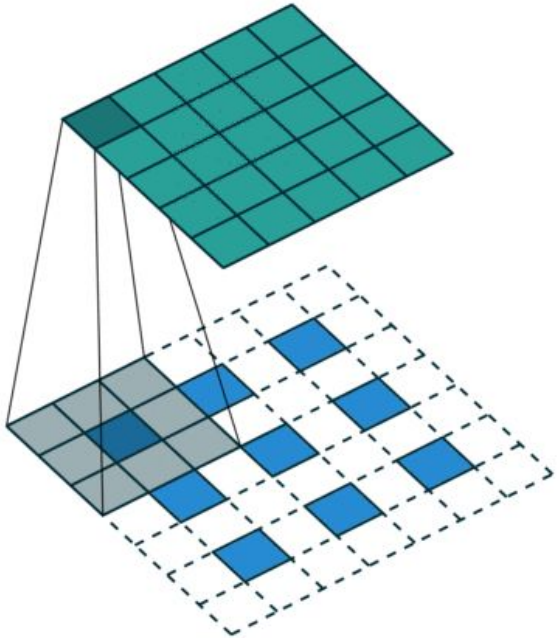


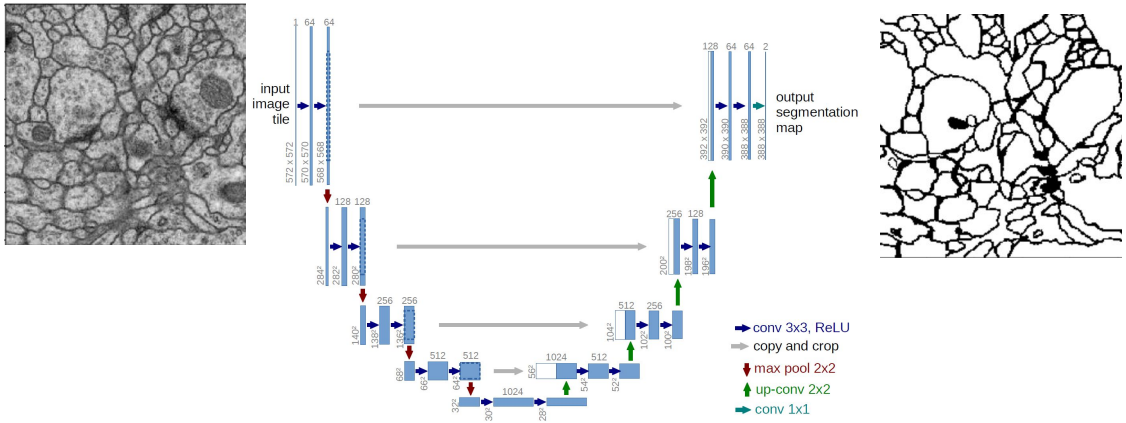
Upconvolution of 3x3 and stride = 1 padded with zeros

Effect: the output is a 5x5 image with ‘synthetic’ pixels

Activity: Conv2D in Pytorch (Colab notebook)

Usage of ‘up-convolution’: decoder paths in image generation models





Upconvolution of 3x3 and stride = 1 padded with zeros

Effect: the output is a 5x5 image with ‘synthetic’ pixels

Activity: Conv2D in Pytorch (Colab notebook)

The checkerboard artifact from “deconvolution”

Deconv in last two layers.
Other layers use resize-convolution.
Artifacts of frequency 2 and 4.

Deconv only in last layer.
Other layers use resize-convolution.
Artifacts of frequency 2.

All layers use resize-convolution.
No artifacts.

Image from [Deconvolution and Checkerboard artifacts]

Original Image
Size: 224x224

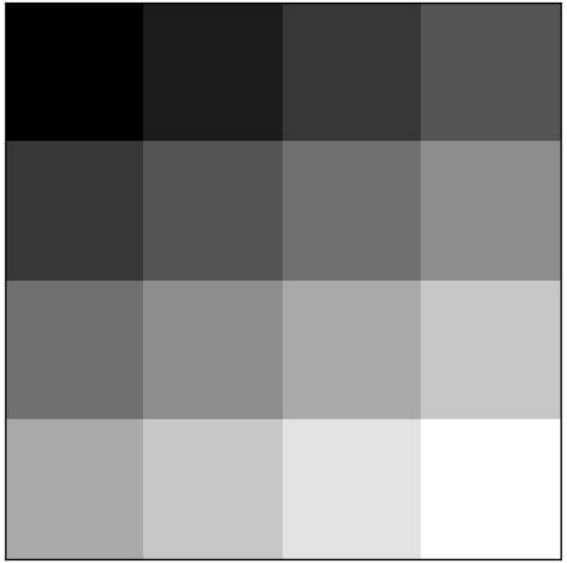
Upscaled Image (Transposed Convolution)
Size: 447x447

Problem: “checkerboard” artifacts

Bilinear interpolation

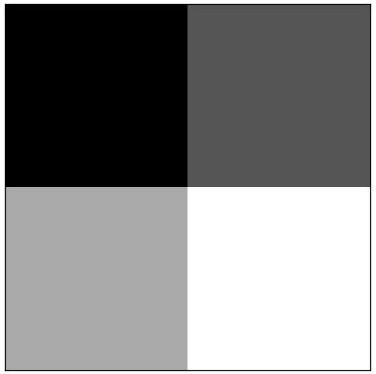
$$w_x = \frac{x - x_1}{x_2 - x_1}$$

output



$$w_y = \frac{y - y_1}{y_2 - y_1}$$

input



$(x_1, y_1) = (0, 0)$ coordinates of Q_{11}

$(x_2, y_2) = (1, 1)$ coordinates of Q_{22}

$(x, y) = (0.6, 0.7)$ target point

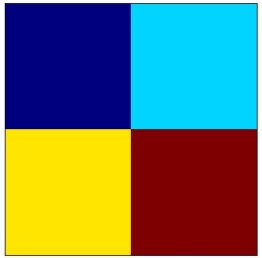
$$Q = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

$$f(x, y) = \begin{bmatrix} 1 - w_x & w_x \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} 1 - w_y \\ w_y \end{bmatrix}$$

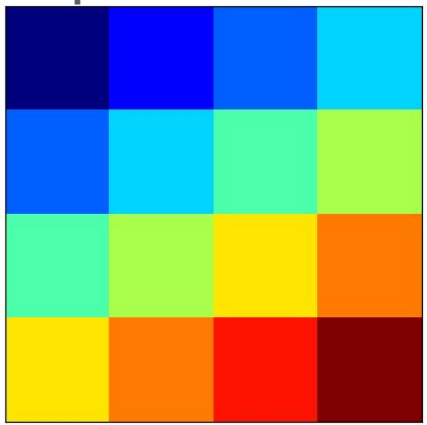
Bilinear interpolation in PyTorch

```
import torch
import torch.nn.functional as F

# Create a 1x1x2x2 tensor (batch_size x channels x height x width)
# Notice that the batch size and channel dimensions are created by wrapping
# the height and width tensor with two pairs of extra square brackets
input = torch.tensor([[[[10, 20],
                        [30, 40]]]],
                     dtype=torch.float32)

# Upscale to 4x4
output = F.interpolate(input, size=(4, 4), mode='bilinear',
align_corners=True)

import matplotlib.pyplot as plt
# The colormap is just for illustration of corner alignment
plt.imshow(input.squeeze(), cmap="jet")
plt.imshow(output.squeeze(), cmap="jet")
```

input (with colormap)
shape: 2x2



output (with colormap)
shape: 4x4



1x1 convolutions mix image channels

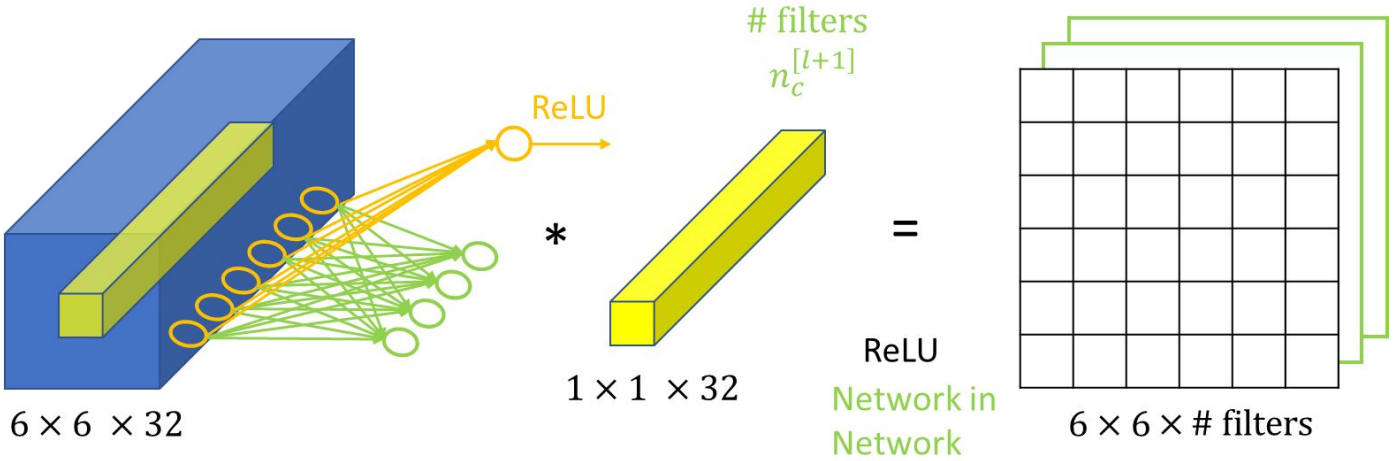


Image from "What does a 1x1 convolution do?" by Andrew Ng

Upsampling and channel mixing with 1x1 convs

Original Image
Size: 224x224

Sample Convolution Outputs (4 of 64)
Size: 224x224

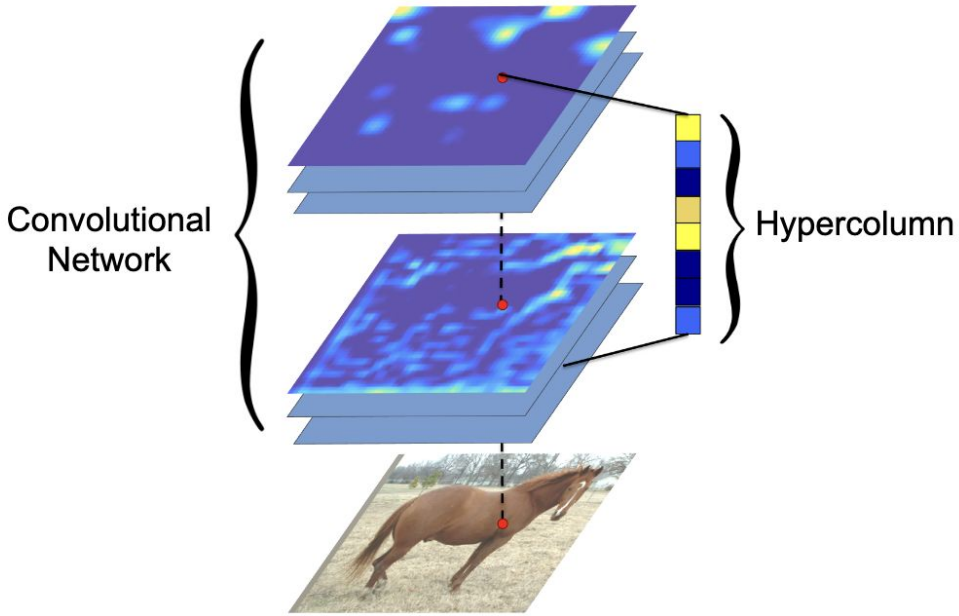
Sample Upsampled Outputs
Size: 448x448

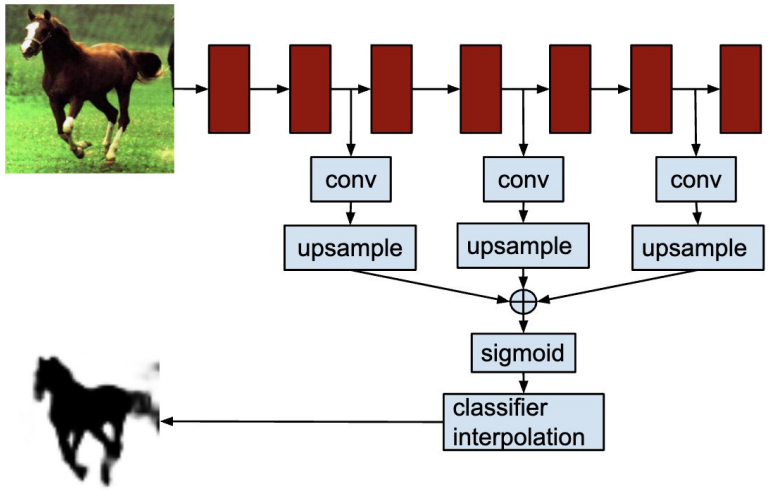
Final Output (After 1x1 Conv)
Size: 448x448

```
layers = nn.Sequential(
    nn.Conv2d(
        in_channels=1,
        out_channels=64,
        kernel_size=3,
        padding=1,
        bias=False
    ),
    nn.Upsample(
        scale_factor=2,
        mode='bilinear',
        align_corners=True
    ),
    nn.Conv2d(
        in_channels=64,
        out_channels=1,
        kernel_size=1,
        bias=False
    )
)
```

Note: no checkerboard artifacts

Combining “hypercolumns” is a common use of 1x1 convolutions





Images from “Hypercolumns for Object Segmentation and Fine-grained Localization”

Summary

1x1 convolutions mix channel information

- Mixing channels allows us to create new images by combining learned features

Transposed convolutions enable upsampling (with checkerboard artifacts)

- Bilinear interpolation and channel mixing is a better alternative

Further reading and references

A guide to convolution arithmetic for deep learning

- https://arxiv.org/abs/1603.07285

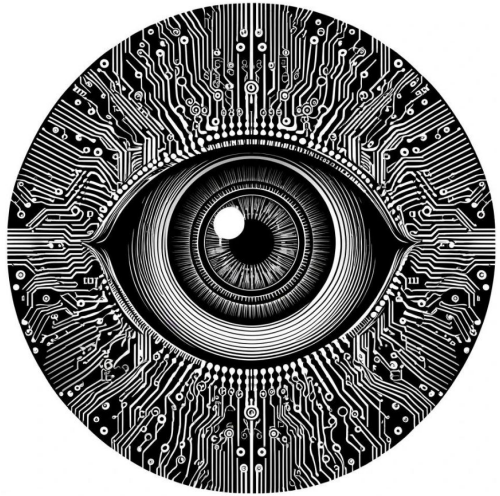
Network in network (1x1 convolutions)

- https://arxiv.org/abs/1312.4400

Hypercolumns for object segmentation and fine-grained localization

- https://openaccess.thecvf.com/content_cvpr_2015/papers/Hariharan_Hypercolumns_for_Object_2015_CVPR_paper.pdf

Contrastive Language-Image Pretraining (CLIP)



Learning goals

- Understand how CLIP models are trained
- Classify images with CLIP using a zero-shot approach
- Create image and text embeddings with a pretrained CLIP model
- Discuss CLIP’s uses and limitations

CLIP: ‘Contrastive Language Image Pretraining’

Learning Transferable Visual Models From Natural Language Supervision

Alec Radford [*1] **Jong Wook Kim** [*1] **Chris Hallacy** [1] **Aditya Ramesh** [1] **Gabriel Goh** [1] **Sandhini Agarwal** [1]
Girish Sastry [1] **Amanda Askell** [1] **Pamela Mishkin** [1] **Jack Clark** [1] **Gretchen Krueger** [1] **Ilya Sutskever** [1]

- Connects text and images in a shared embedding space
- Created by OpenAI in 2021
- Trained on **400M image-textual description pairs** scraped from the internet
- Predicts the most relevant text snippet given an image, enabling “zero-shot classification”

Aligning text and image embeddings

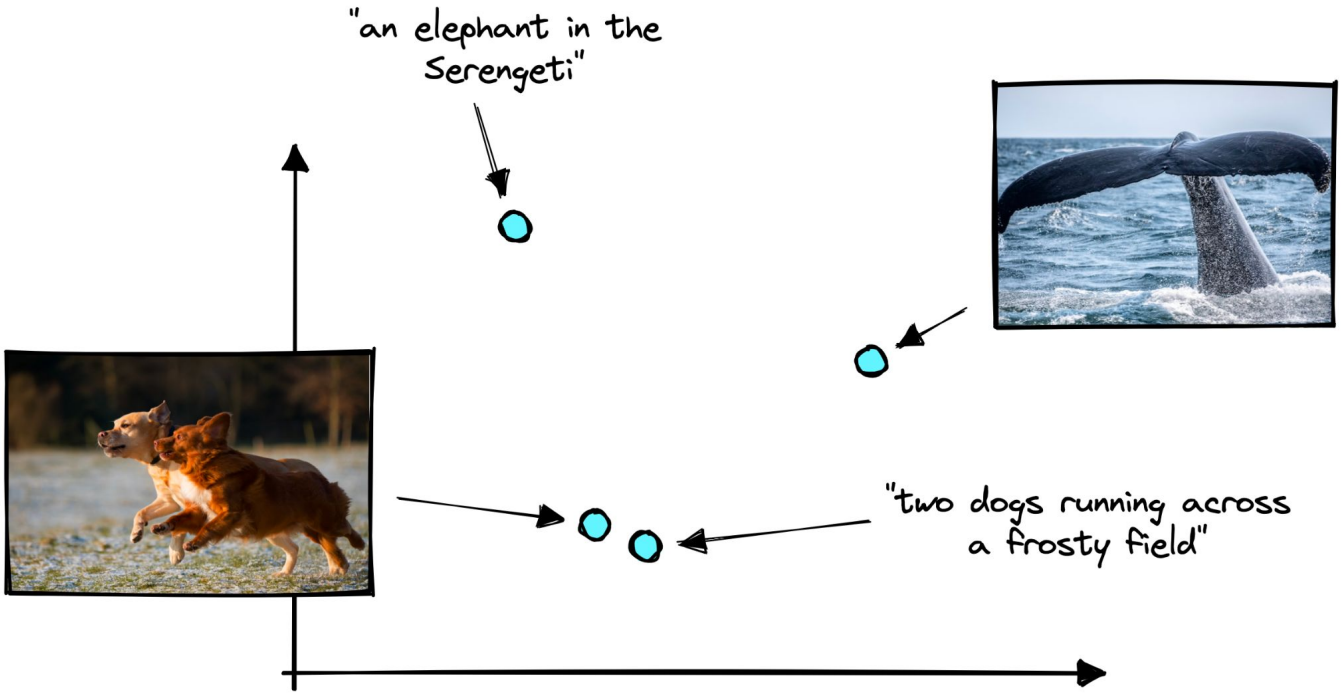
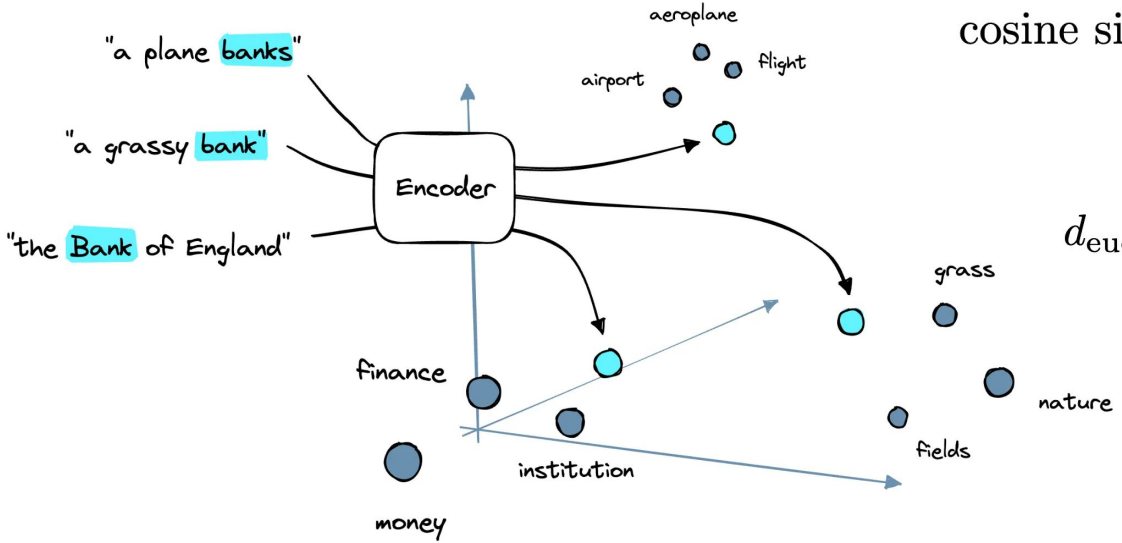


Image from https://www.pinecone.io/learn/series/image-search/clip/

Text encoders

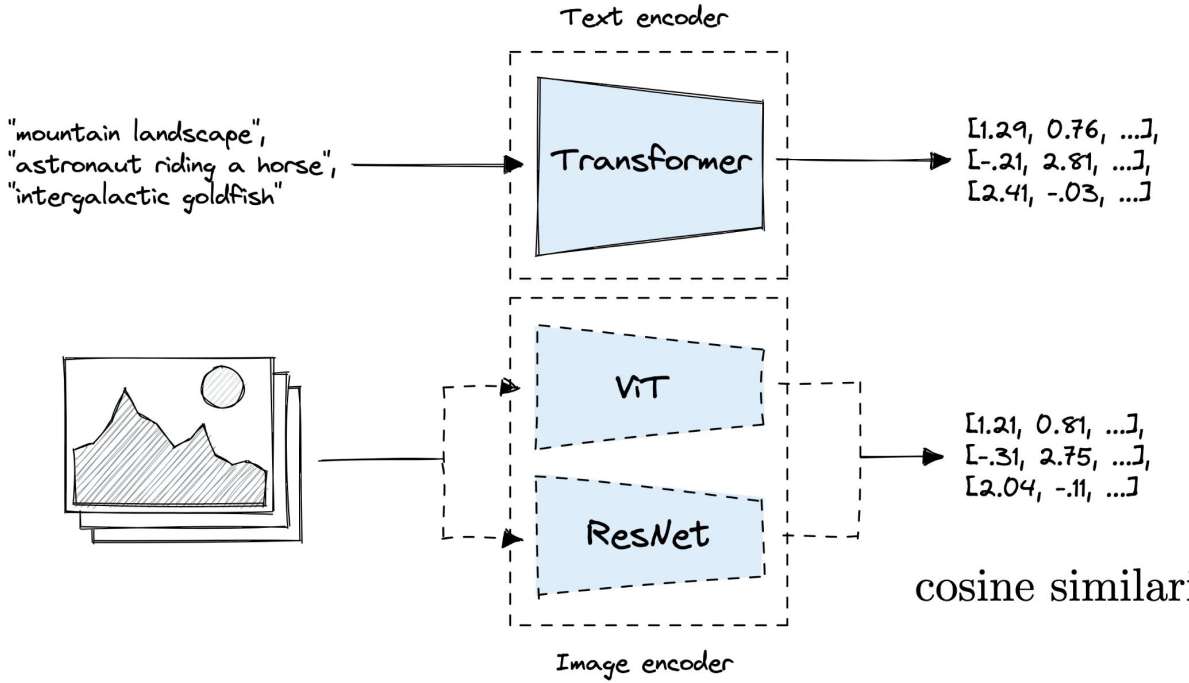


$$\text{cosine similarity}(x_1, x_2) = \frac{x_1 \cdot x_2}{\|x_1\| \, \|x_2\|}$$

$$d_{\text{euclid}}(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

Image from https://www.pinecone.io/learn/series/image-search/vision-transformers/

CLIP's architecture



Both encoders map inputs to 512 dimensional embeddings

$$\text{cosine similarity}(x_1, x_2) = \frac{x_1 \cdot x_2}{\|x_1\| \ \|x_2\|}$$

Image from https://www.pinecone.io/learn/series/image-search/clip/

Maximizing cosine similarity of matching text and image embeddings

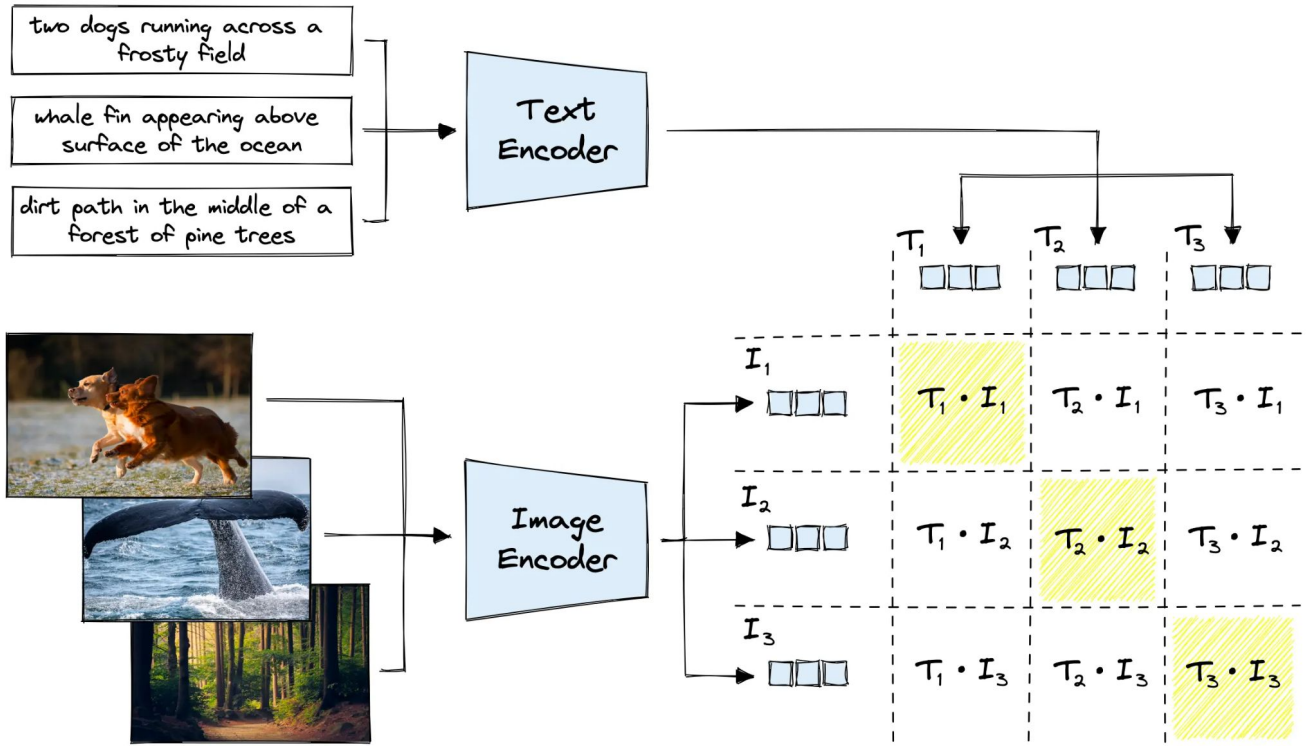
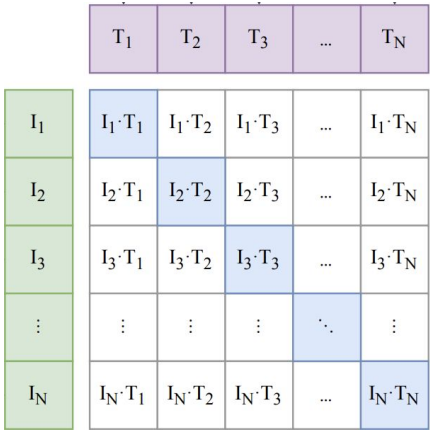


Image from https://www.pinecone.io/learn/series/image-search/clip/

Training algorithm

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

Figure 3. Numpy-like pseudocode for the core of an implementation of CLIP.

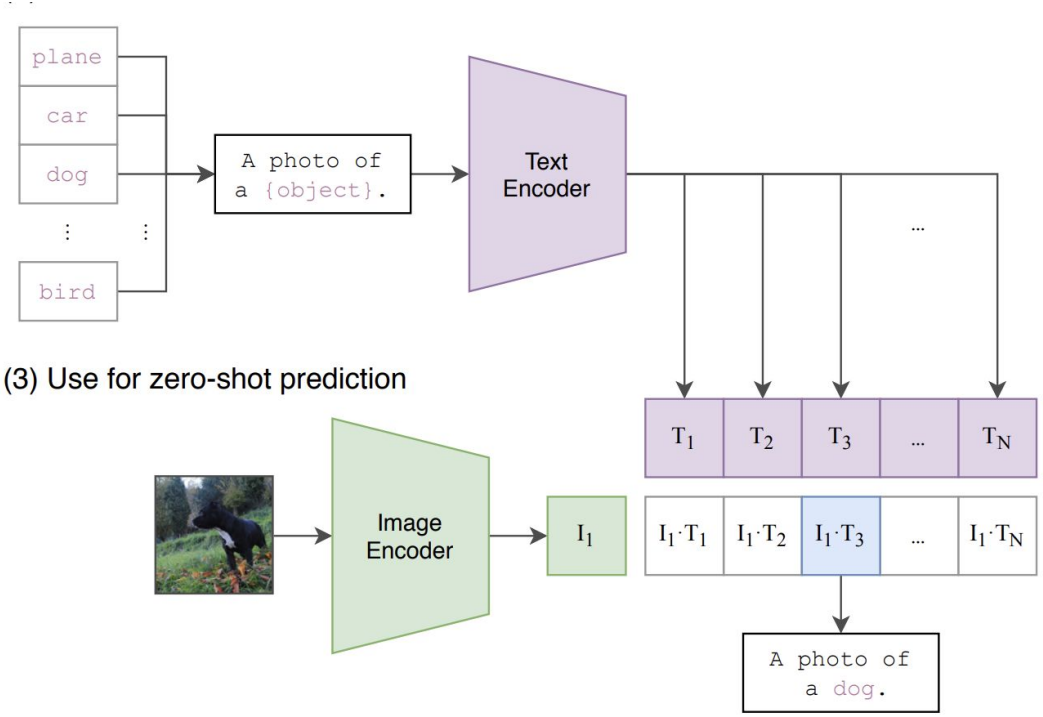


$$H(p,q) = -\sum_i p(i) \log q(i)$$

$$H(q,p) = -\sum_i q(i) \log p(i)$$

$$\text{symmetric CE loss} = \frac{H(p,q) + H(q,p)}{2}$$

Zero-shot classification with CLIP



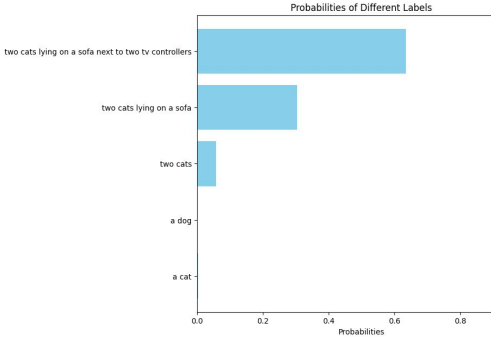
The model is able to create good classifiers without extra training

$$\text{softmax}(x_i, T) = \frac{e^{x_i/T}}{\sum_{j=1}^{n} e^{x_j/T}}$$

Image from Learning Transferable Visual Models From Natural Language Supervision

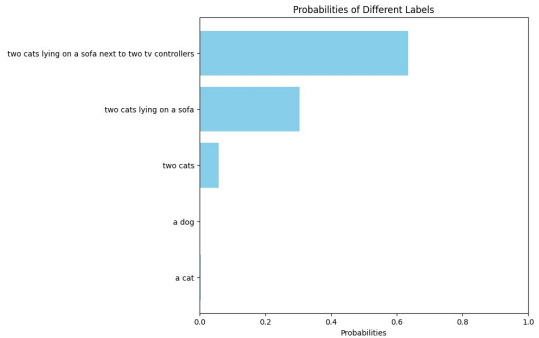
Producing embeddings with CLIP (½)

```
import torch
from transformers import CLIPProcessor, CLIPModel
from PIL import Image

# Load model and inputs
model =
CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
processor =
CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
image = Image.open("cats.jpg")
cat_text = ['a cat', 'a dog', 'two cats',
            'two cats lying on a sofa',
             """two cats lying on a sofa
             next to two tv controllers"""]
```



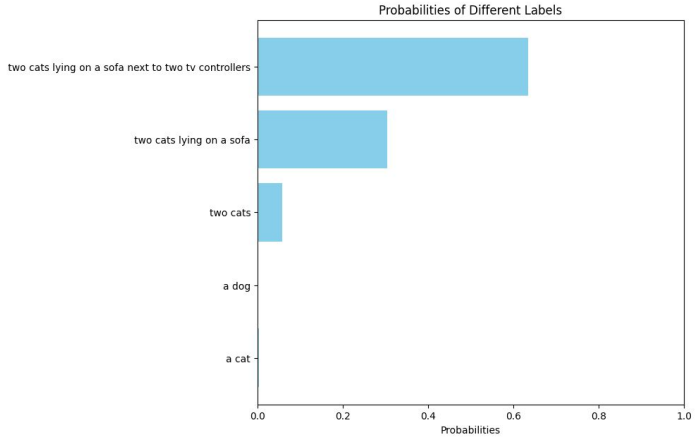
Producing embeddings with CLIP (2/2)

```python
# Get embeddings
inputs = processor(text=cat_text, images=image, return_tensors="pt",
padding=True)
outputs = model(**inputs)

# Calculate similarities
sims = torch.nn.functional.cosine_similarity(
    outputs.image_embeds[:, None],
    outputs.text_embeds[None, :],
    dim=-1
)

# Print results
for text, sim in zip(cat_text, sims[0]):
    print(f"{text}: {sim:.3f}")
```



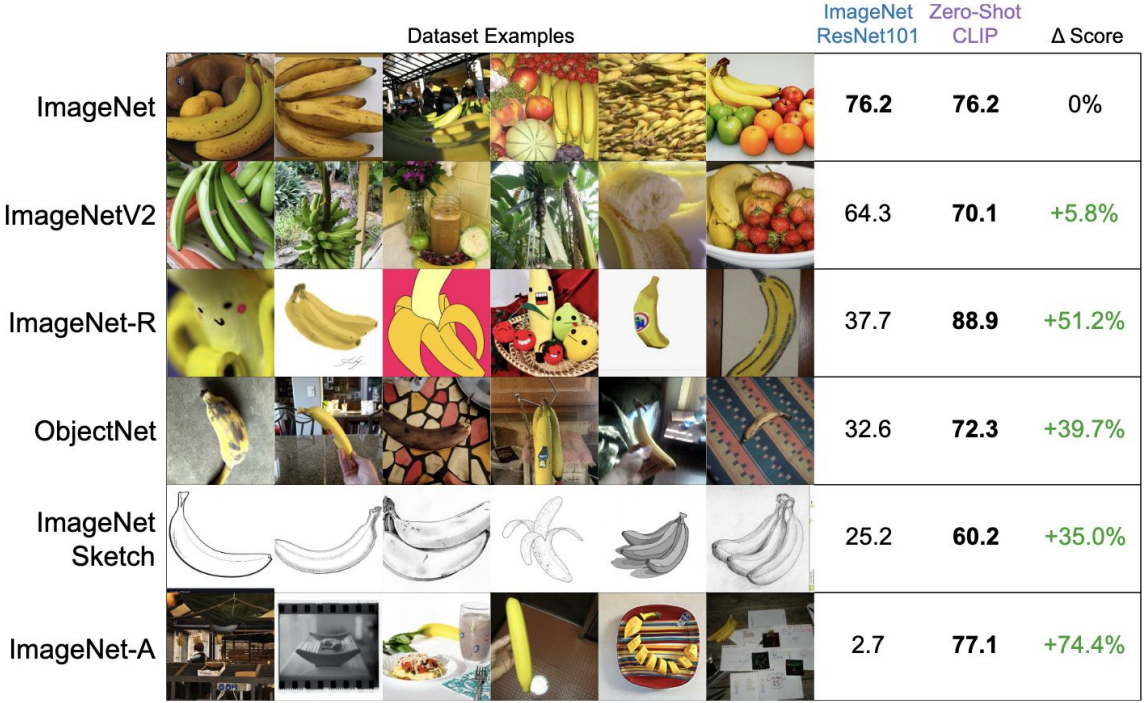
Zero-shot classification with CLIP

```
# Notice what happens with the output probabilities
# when we make the labels more specific
cat_text = ['a cat',
            'a dog',
            'two cats',
           ' two cats lying on a sofa',
            'two cats lying on a sofa next to two tv controllers'
            ]

inputs = processor(text=cat_text,
                   images=[cats_img],
                   return_tensors="pt", padding=True)
cat_outputs = model(**inputs)
cat_logits_per_image = cat_outputs.logits_per_image
cat_probs = (cat_logits_per_image/temperature).softmax(dim=1)
```



$$\text{softmax}(x_i, T) = \frac{e^{x_i/T}}{\sum_{j=1}^{n} e^{x_j/T}}$$

Transferable representations: CLIP against a ResNet101 pretrained on Imagenet



Training Resnet101: about 90 V100 GPU hours (4 days)
Training CLIP ViT-L/14: about ~16,000 V100 GPU hours (666 days)

Note that CLIP was trained on a dataset of **400 million images** scraped from the Internet

The Imagenet1K dataset has only **1.28 million training images**

CLIP is a much more capable "foundation model"

Resnet101

~44.5 million parameters

CLIP ViT-L/14@336px:

~428 million parameters

Limitations against fully supervised models

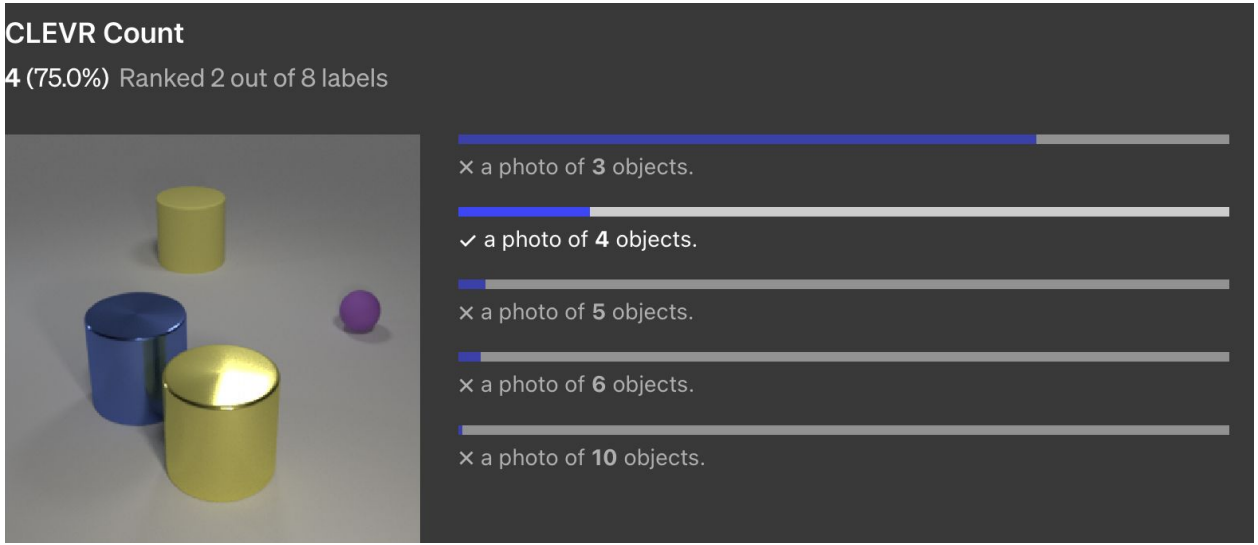
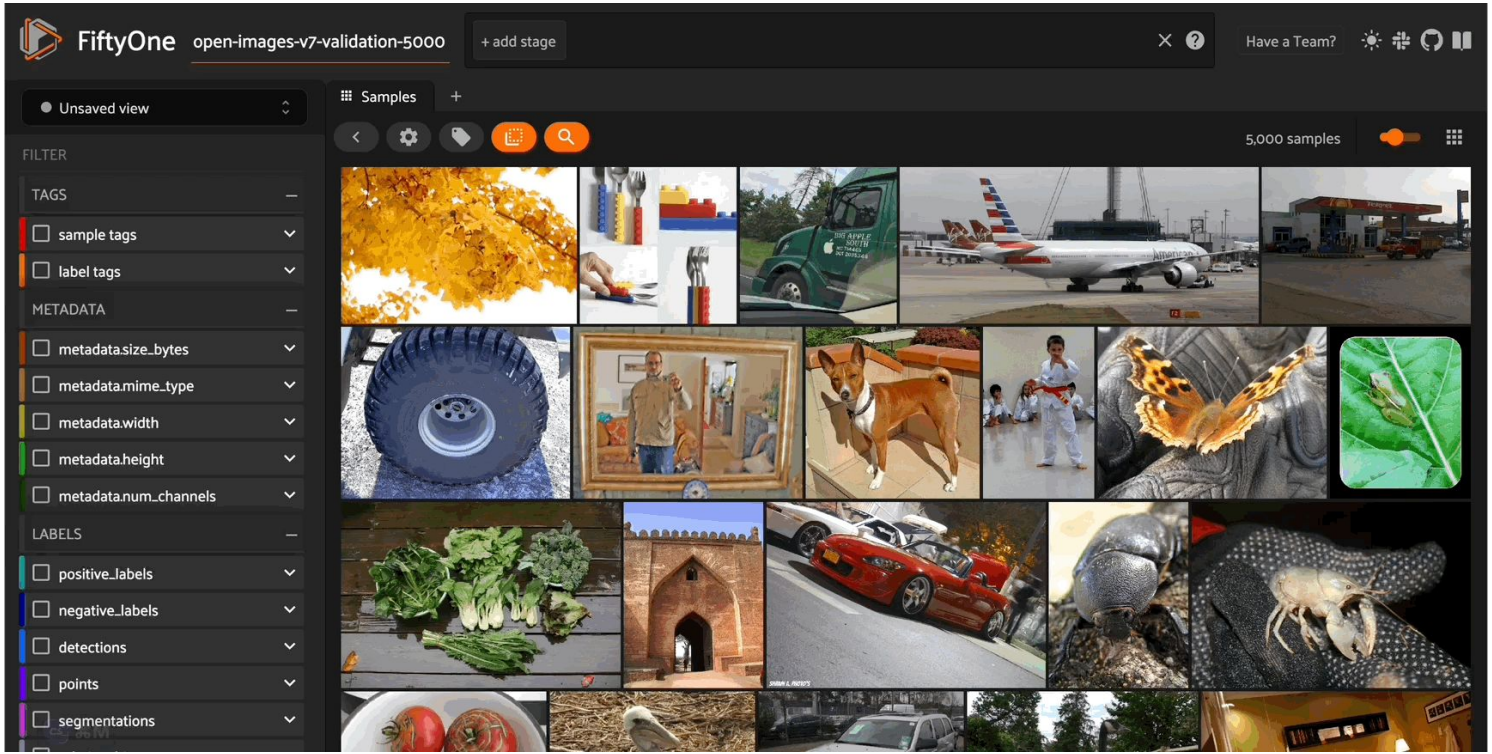


Image from
https://openai.com/index/clip/

The authors comment on the limitations section of the paper
that CLIP often fails to perform as well as a fully supervised model fine tuned for the task.
Notable examples:

- digit recognition (MNIST)
- tumor classification (PatchCamelyon)
- satellite imaging (EuroSAT)
- object counting (CLEVR Count)

Semantic search with CLIP



CLIP guides image generation of diffusion models

Prompt:

"Create an image of an astronaut riding a horse in pencil drawing style"

Image by OpenAI's Dall-E 3

Summary

CLIP learns joint embeddings

- CLIP creates a shared embedding space for images and text
- These multimodal embeddings have applications on semantic search and image generation

CLIP excels at zero-shot classification

- CLIP performs well on unseen tasks without additional training

CLIP has limitations

- CLIP is not always better than models fine-tuned for specific tasks
- Retraining CLIP with a ViT base is expensive both computationally and data-wise

Further reading

Learning Transferable Visual Models From Natural Language Supervision

- https://arxiv.org/abs/2103.00020

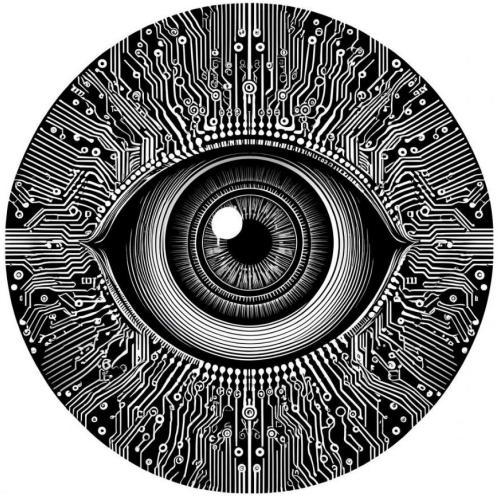
A Google Search Experience for Computer Vision Data

- https://voxel51.com/blog/a-google-search-experience-for-computer-vision-data/

Multi-modal ML with OpenAI's CLIP

- https://www.pinecone.io/learn/series/image-search/clip/

Image Generation with Diffusion Models



Learning goals

- Gain an overview of the denoising diffusion process
- Recognize the use of CLIP models to guide image generation with text prompts

Image generation with diffusion models

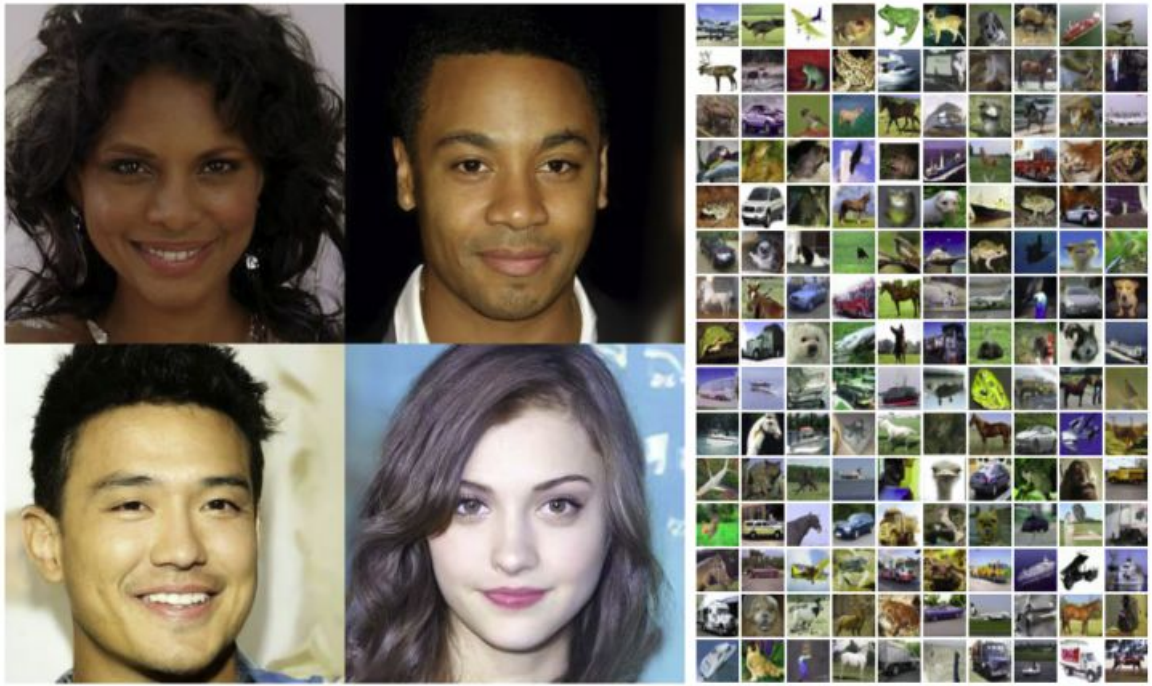
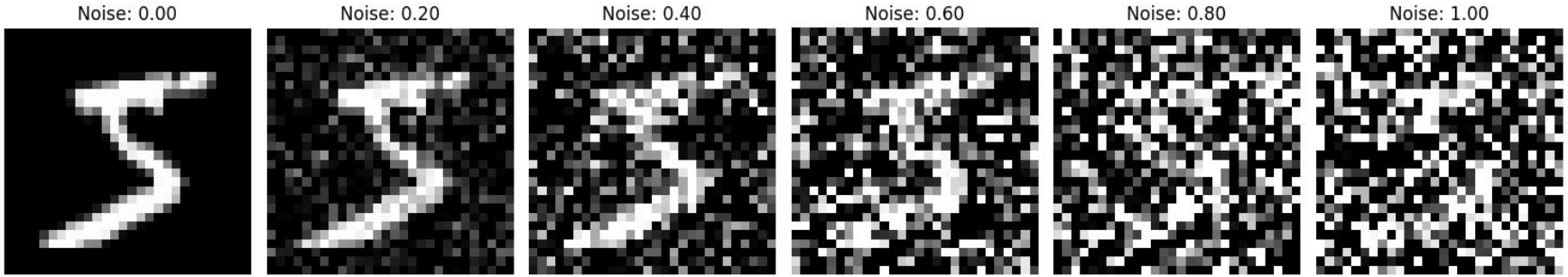


Figure 1: Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

Image from Denoising Diffusion Probabilistic Models

Corrupting an image with Gaussian noise



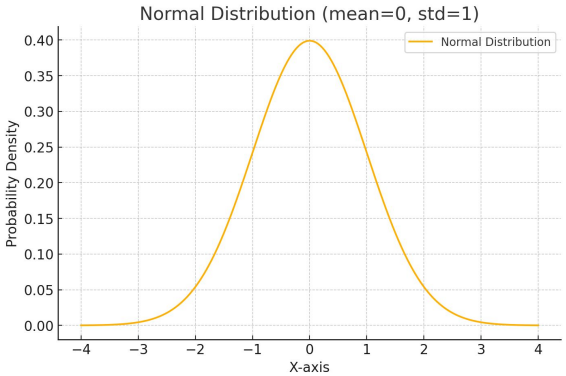
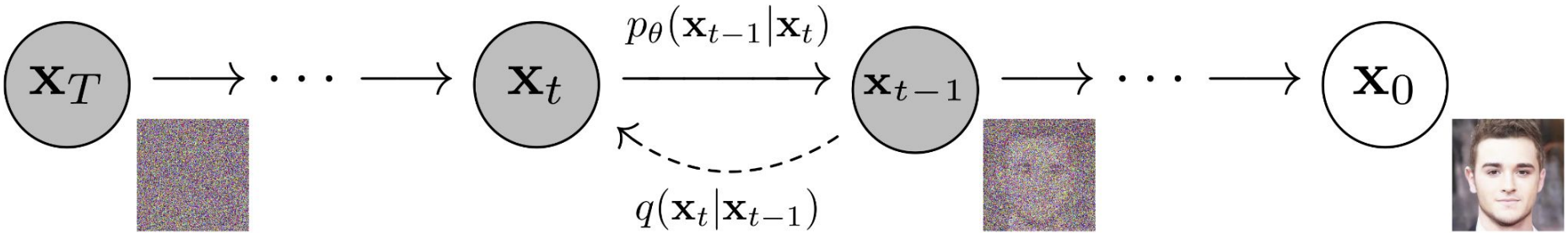
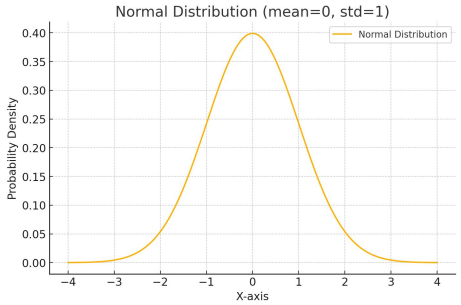


Image from source

Iterative denoising process





Forward diffusion

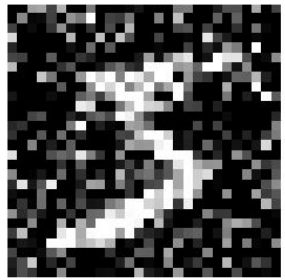
Beta controls how much noise is added on each time step, it is increased gradually. This increase is called the "noise schedule"

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = N(\mathbf{x}_t; \sqrt{1-\beta_t} \cdot \mathbf{x}_{t-1}, \beta_t \cdot \mathbf{I})$$

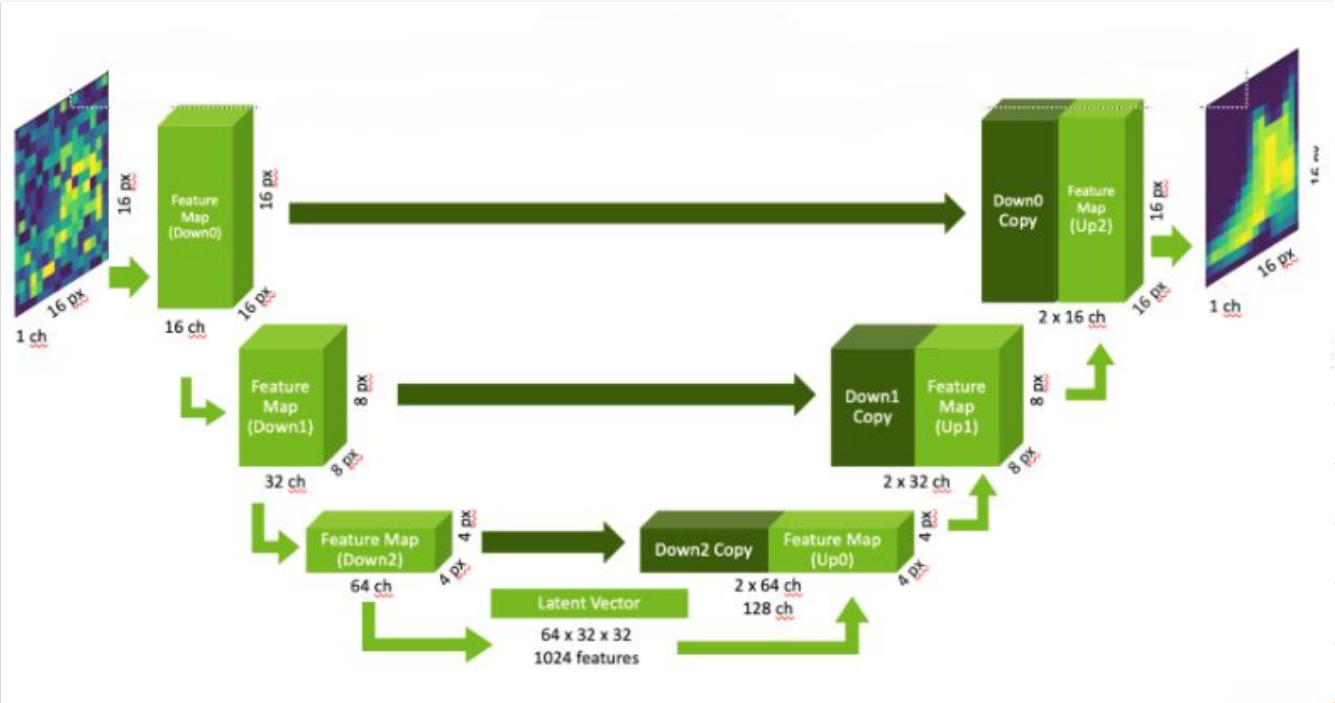
```python
import numpy as np

def step_forward(x_prev, beta_t):
    # Scale down the previous position
    mean = np.sqrt(1 - beta_t) * x_prev

    # Add random noise
    std = np.sqrt(beta_t)
    noise = np.random.normal(0, std, size=x_prev.shape)

    x_t = mean + noise
    return x_t
```

Noise: 0.20

Noise: 0.40



Reverse diffusion with U-net



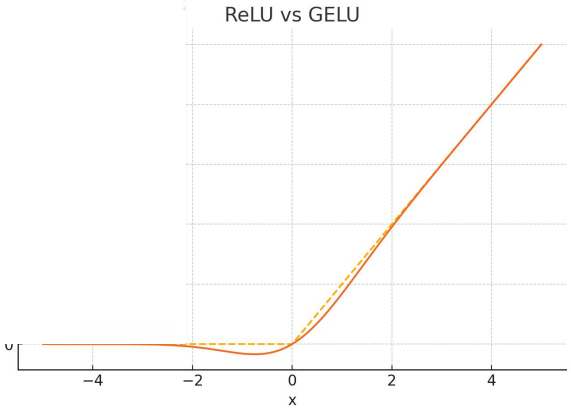
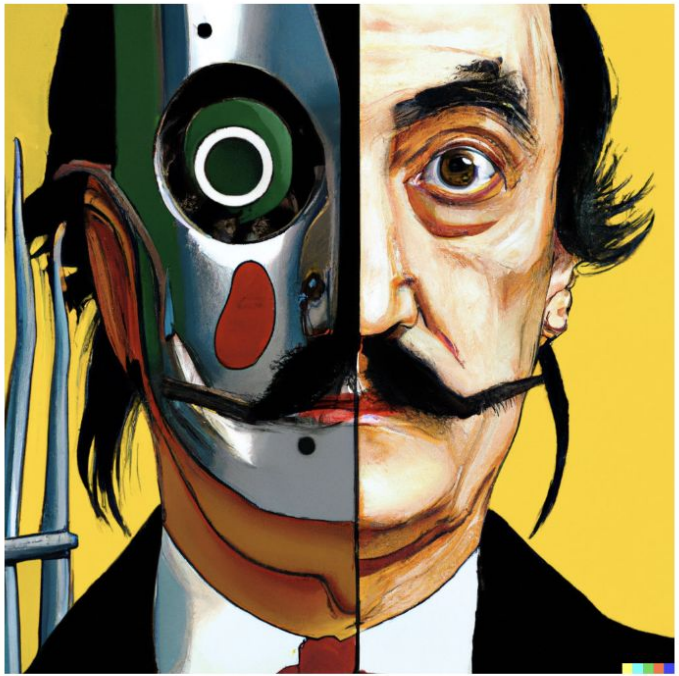


Image source

CLIP to guide text to image generation



vibrant portrait painting of Salvador Dalí with a robotic half face

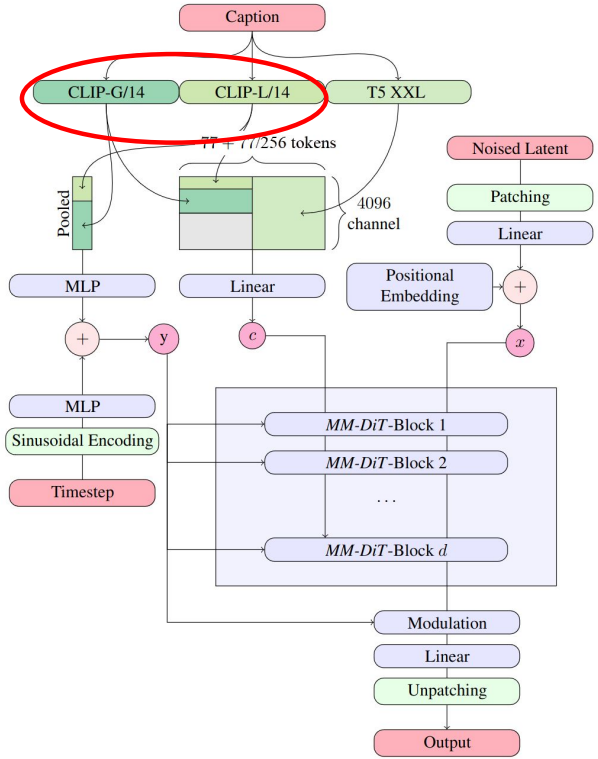


Image from Hierarchical Text-Conditional Image Generation with CLIP Latents

Architecture diagram from Stable Diffusion 3.5

CLIP as input to decoders

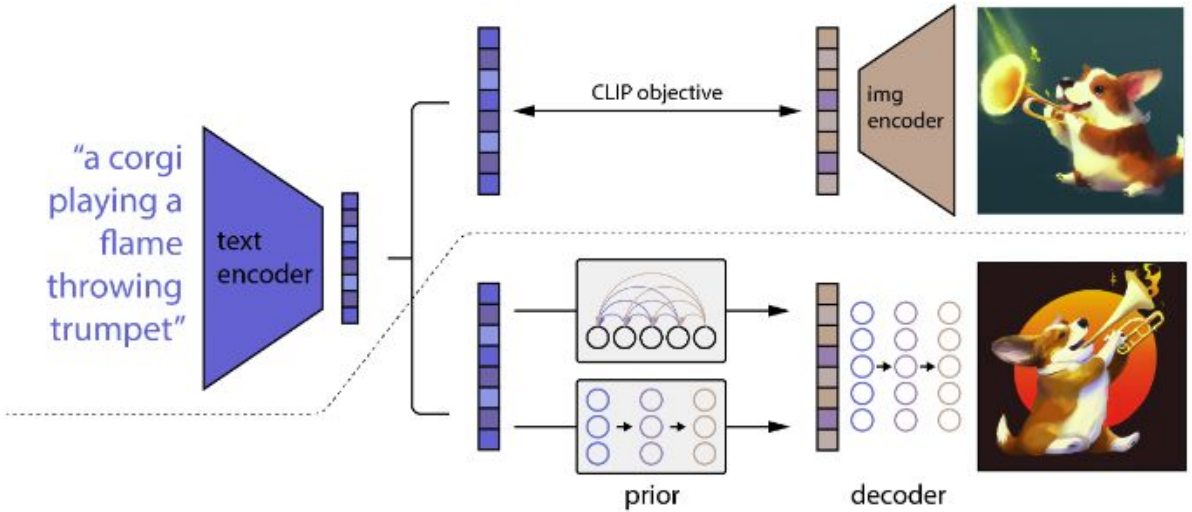


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

Image from Hierarchical Text-Conditional Image Generation with CLIP Latents

Summary

Diffusion models generate high quality images by reversing a noise addition process.

- They iteratively denoise from pure noise to generate images.

The forward diffusion process corrupts images

- Gradually adds Gaussian noise to images following a schedule (beta parameter)

The reverse diffusion process is about learning to predict the noise

- Uses a U-net architecture to estimate what noise was added at each step
- We predict the noise component to subtract it from the corrupted image
- The network is trained to minimize the difference between predicted and actual noise

CLIP enables text-guided image generation

- CLIP text embeddings help us control the reverse diffusion process.

Further reading and references

Denoising Diffusion Probabilistic Models

- https://arxiv.org/abs/2006.11239

Hierarchical Text-Conditional Image Generation with CLIP Latents

- https://arxiv.org/abs/2204.06125

The Annotated Diffusion Model

- https://huggingface.co/blog/annotated-diffusion

The Physics Principle That Inspired Modern AI Art

- https://www.quantamagazine.org/the-physics-principle-that-inspired-modern-ai-art-20230105/

Review questions

- Questions on CLIP
- Questions on Diffusion Models

Resources

- Github Repository
- YouTube playlist
- Discord channel
  #practical-computer-vision-workshops