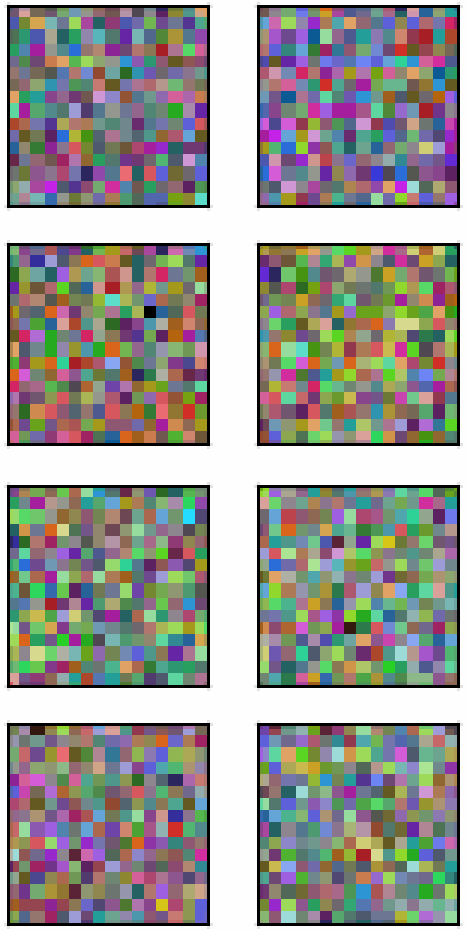
55 AÑOS PUCP
Departamento Académico de Ingeniería
IA PUCP

Entendiendo los modelos de difusión

Winter Camp
en Inteligencia Artificial

Desbloqueando Modelos de Difusión

- Aprender a crear imágenes coherentes de pixel art con modelos de difusión.
- Obtener y comprender el código completo detrás del modelo.
- Descifrar el funcionamiento y los principios de los modelos de difusión.
- Conocer la evolución y la historia detrás de los modelos de difusión.
- Descubrir y aplicar los modelos de difusión en una variedad de campos.



Flujo de la Charla

Parte Teórica

- Evolución del Estado del Arte
- Intuición de Redes Profundas
- Intuición del Proceso de Difusión
- Aplicaciones Diversas de Modelos de Difusión

Coffee Break

Parte Práctica

- Lab 1: Uso Básico de Modelos
- Lab 2: Entrenamiento Básico
- Lab 3: Control sobre el Modelo
- Lab 4: Mejorando Eficiencia
- Recomendaciones Generales

7.

Evolución del Estado del Arte

Todo Comenzó…

Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein
Stanford University

Eric A. Weiss
University of California, Berkeley

Niru Maheswaranathan
Stanford University

Surya Ganguli
Stanford University

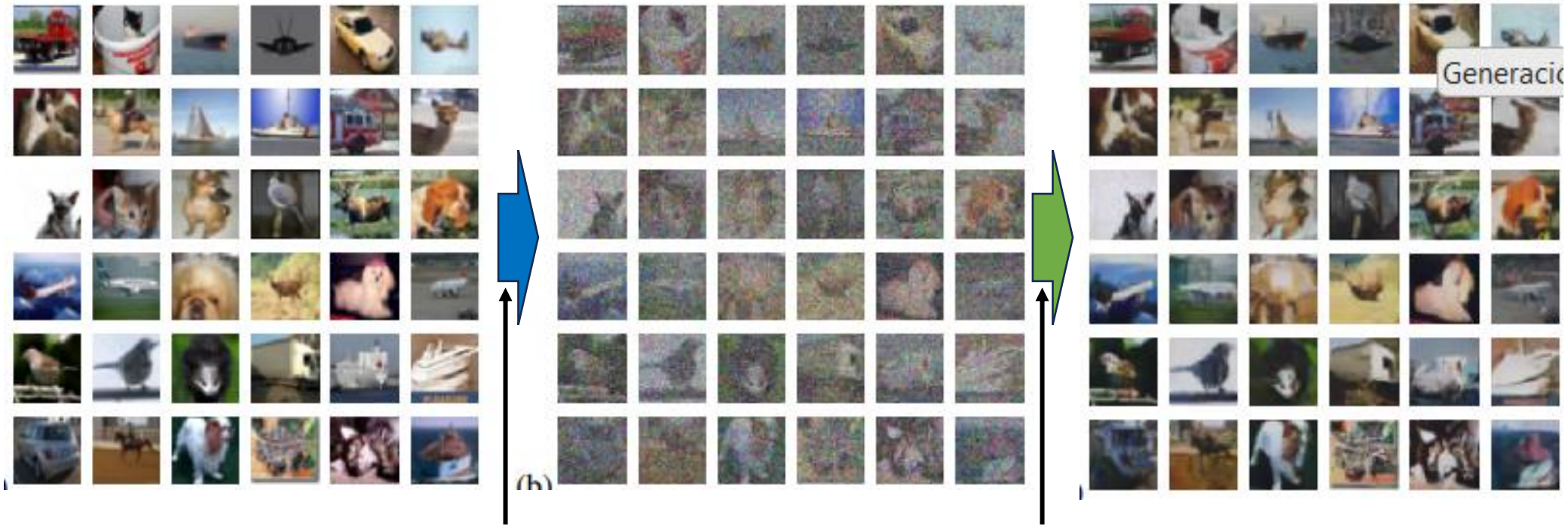
Abstract

A central problem in machine learning involves modeling complex data-sets using highly flexible families of probability distributions in which learning, sampling, inference, and evaluation are still analytically or computationally tractable. Here, we develop an approach that simultaneously achieves both flexibility and tractability.

these models are unable t
datasets. On the other han
molded to fit structure in
can define models in term
$\phi(\mathbf{x})$ yielding the flexible
Z is a normalization cons
normalization constant is g
training, or drawing samples from such flexible models typ-

- **Publicado en 2015:** El concepto inicial de DDPM se presentó en 2015.
- **Basado en Conceptos Físicos:** Utilizaba una metodología inspirada en procesos de difusión física para la generación de datos.
- **No Alcanzó Resultados Prometedores**: El modelo inicial no logró resultados de vanguardia y fue superado en rendimiento por otros modelos generativos contemporáneos.

La idea original



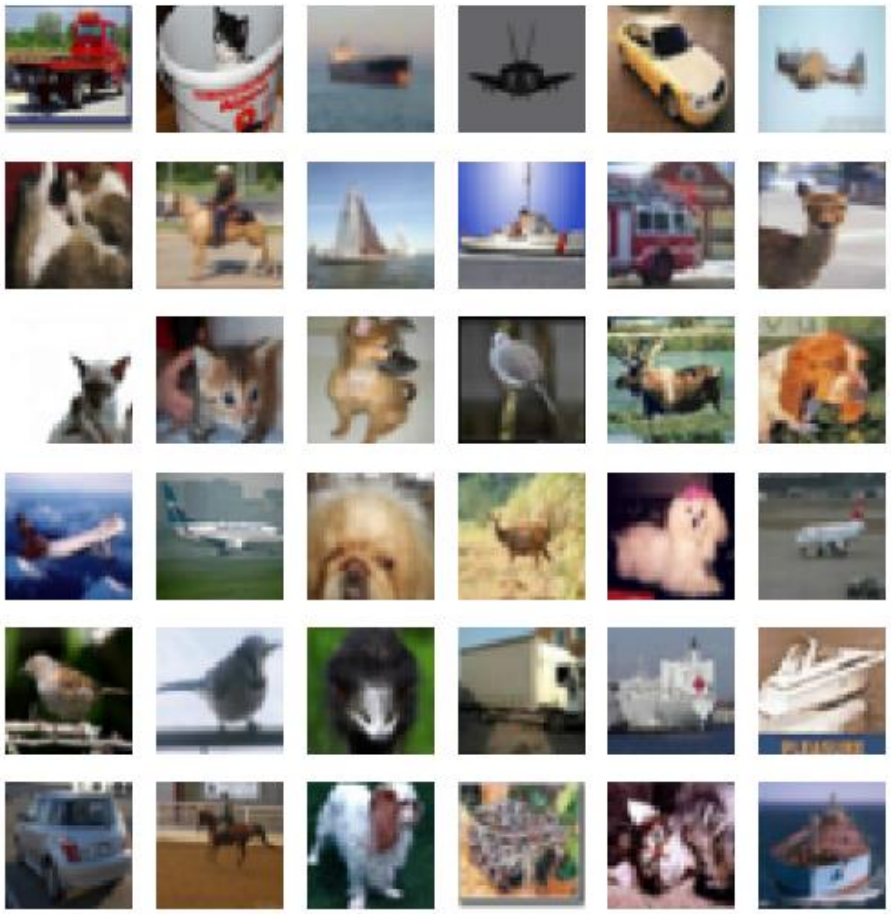
AGREGAR RUIDO
(APRENDER)

QUITAR RUIDO
(APLICAR LO
APRENDIDO)

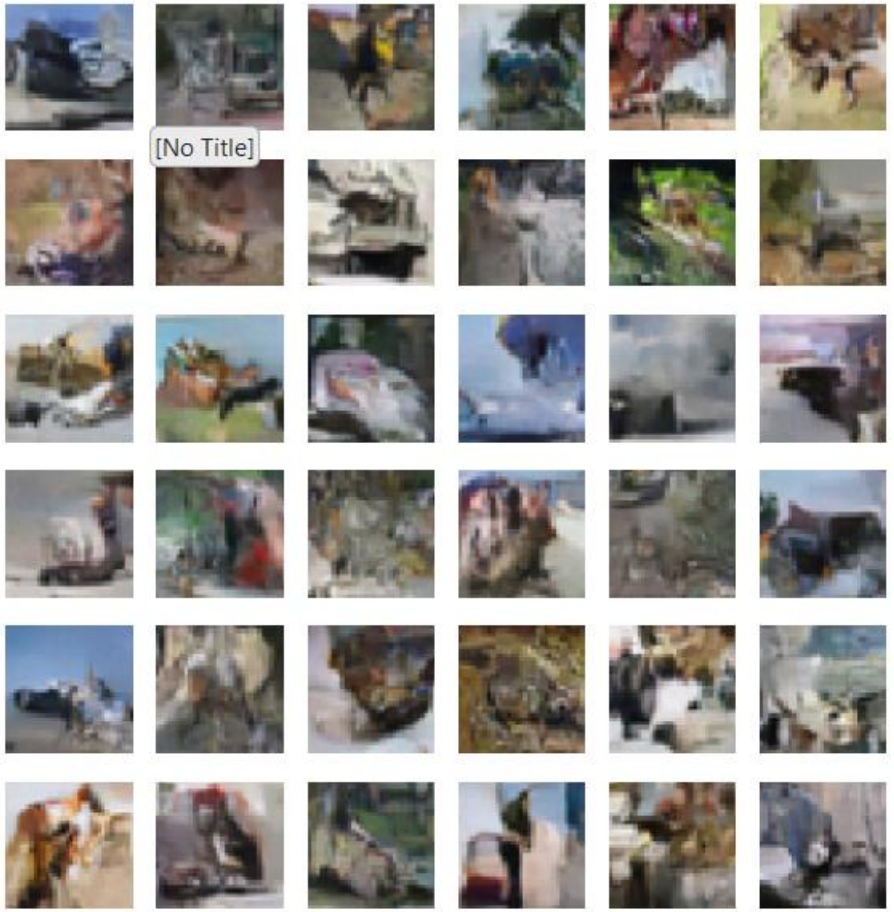
Extendiendo a perder toda señal

Y crear imágenes quitando el ruido

Generación de CIFAR10 en 2015



Dataset



Modelo de difusión

La Influencia de los Transformers

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

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly

Primeros Modelos de DDPM (2015-2016):

- Inicio con modelos simples de DDPM, centrados en demostrar la viabilidad de la difusión de ruido en la generación de datos.

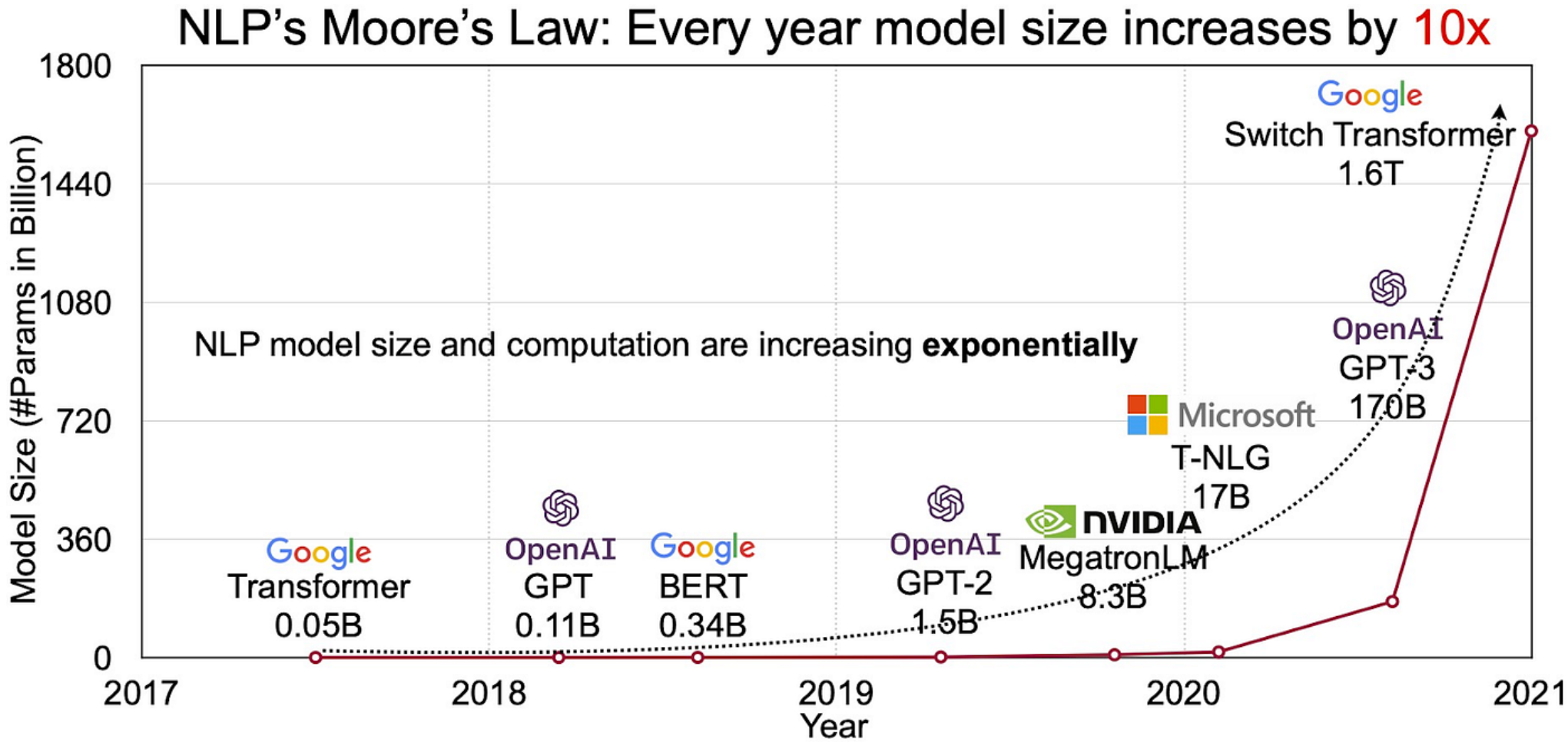
Aparición de los Transformers (2017):

- Lanzamiento de los Transformers, revolucionando el manejo de secuencias de datos y parámetros en aprendizaje profundo.

Inspiración y Cambio de Paradigma (2018-2019):

- El éxito de los Transformers inspira una nueva dirección en el diseño de modelos, incluyendo DDPMs, hacia estructuras más complejas y de mayor escala.
- Este período establece las bases para investigaciones futuras en DDPMs, marcando el inicio de una era de modelos más grandes y computacionalmente intensivos.

Aumento Exponencial de Parámetros



*más de esto después

Aumento de la escala para predecir el “Ruido”

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

Abstract

We present high quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics. Our best results are obtained by training on a weighted variational bound designed according to a novel connection between diffusion probabilistic models and denoising score matching with Langevin dynamics, and our models naturally admit a progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding. On the unconditional CIFAR10 dataset, we obtain an Inception score of 9.46 and a state-of-the-art FID score of 3.17. On 256x256 LSUN, we obtain sample quality similar to ProgressiveGAN. Our implementation is available at https://github.com/hojonathanho/diffusion.

1 Introduction

Deep generative models of all kinds have recently exhibited high quality samples in a wide variety of data modalities. Generative adversarial networks (GANs), autoregressive models, flows, and

- **Publicado en 2020:** "Denoising Diffusion Probabilistic Models" redefine los DDPMs.
- **Innovación:** Introduce un enfoque eficiente para revertir el proceso de difusión, usando una U-Net para invertir el ruido.
- **Mejoras Significativas:** Supera a otros modelos generativos en calidad y detalle, especialmente en la generación de imágenes.
- **Impulso** a la Investigación Futura: Establece un nuevo estándar.

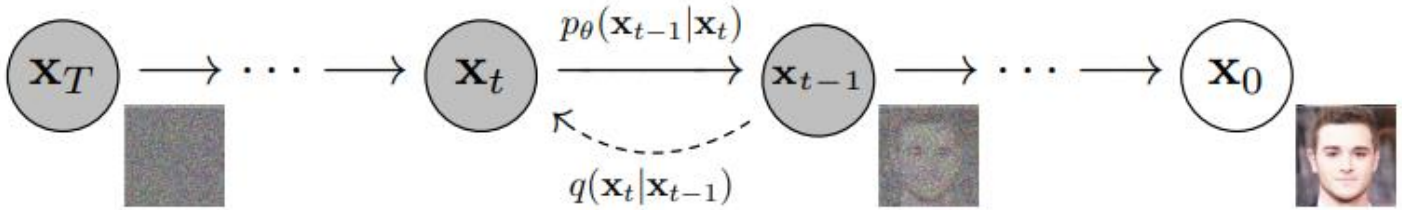
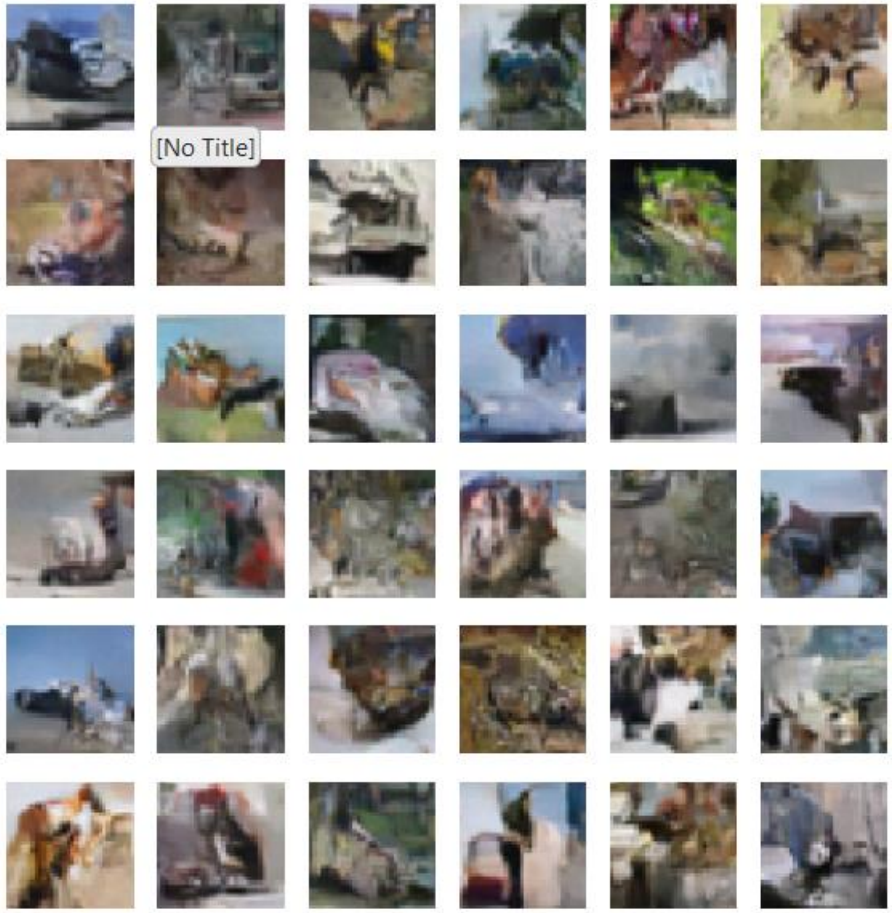
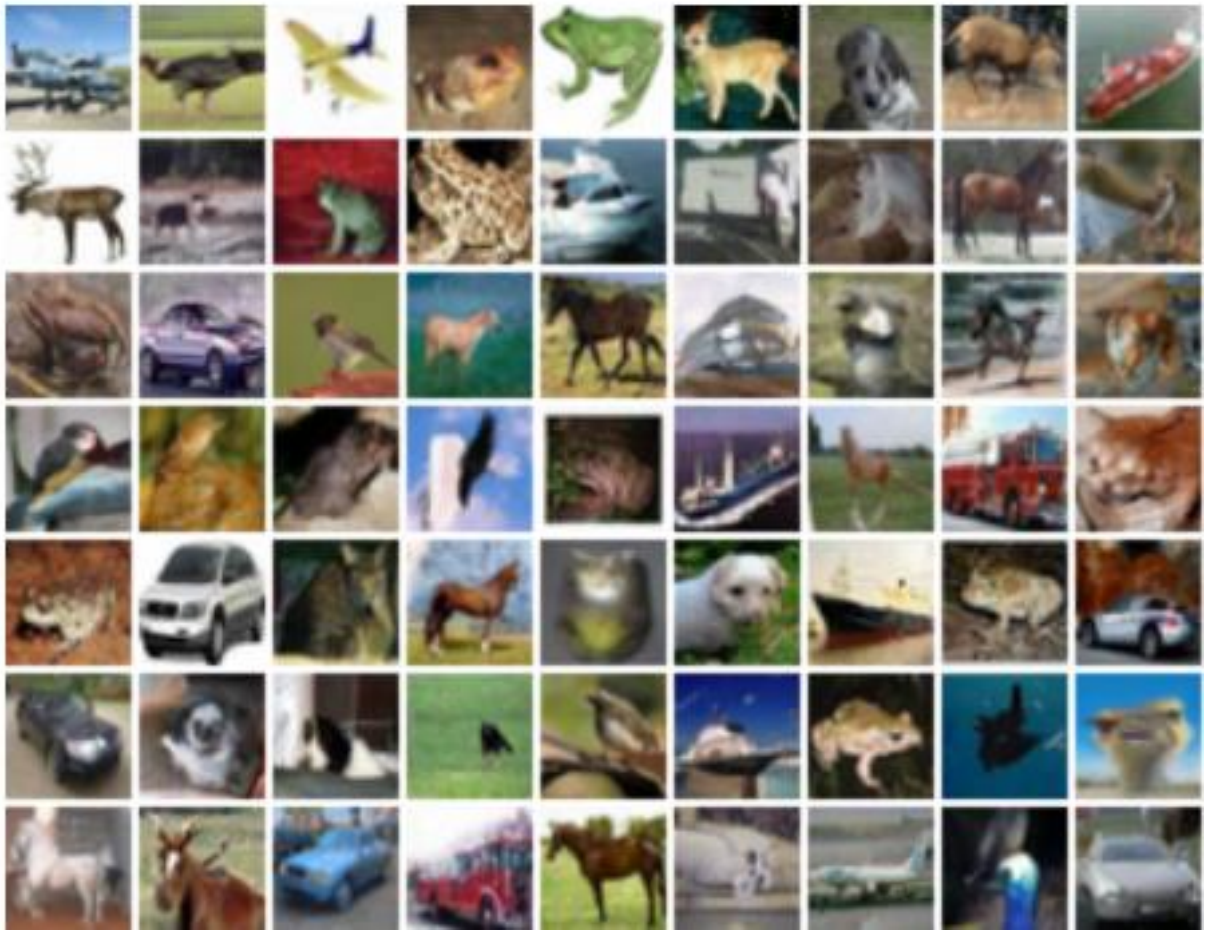


Figure 2: The directed graphical model considered in this work.

Generación de CIFAR10 en 2020



2015



2020

El Nuevo Chico en la Cuadra

Diffusion Models Beat GANs on Image Synthesis

Prafulla Dhariwal*
OpenAI
prafulla@openai.com

Alex Nichol*
OpenAI
alex@openai.com

Abstract

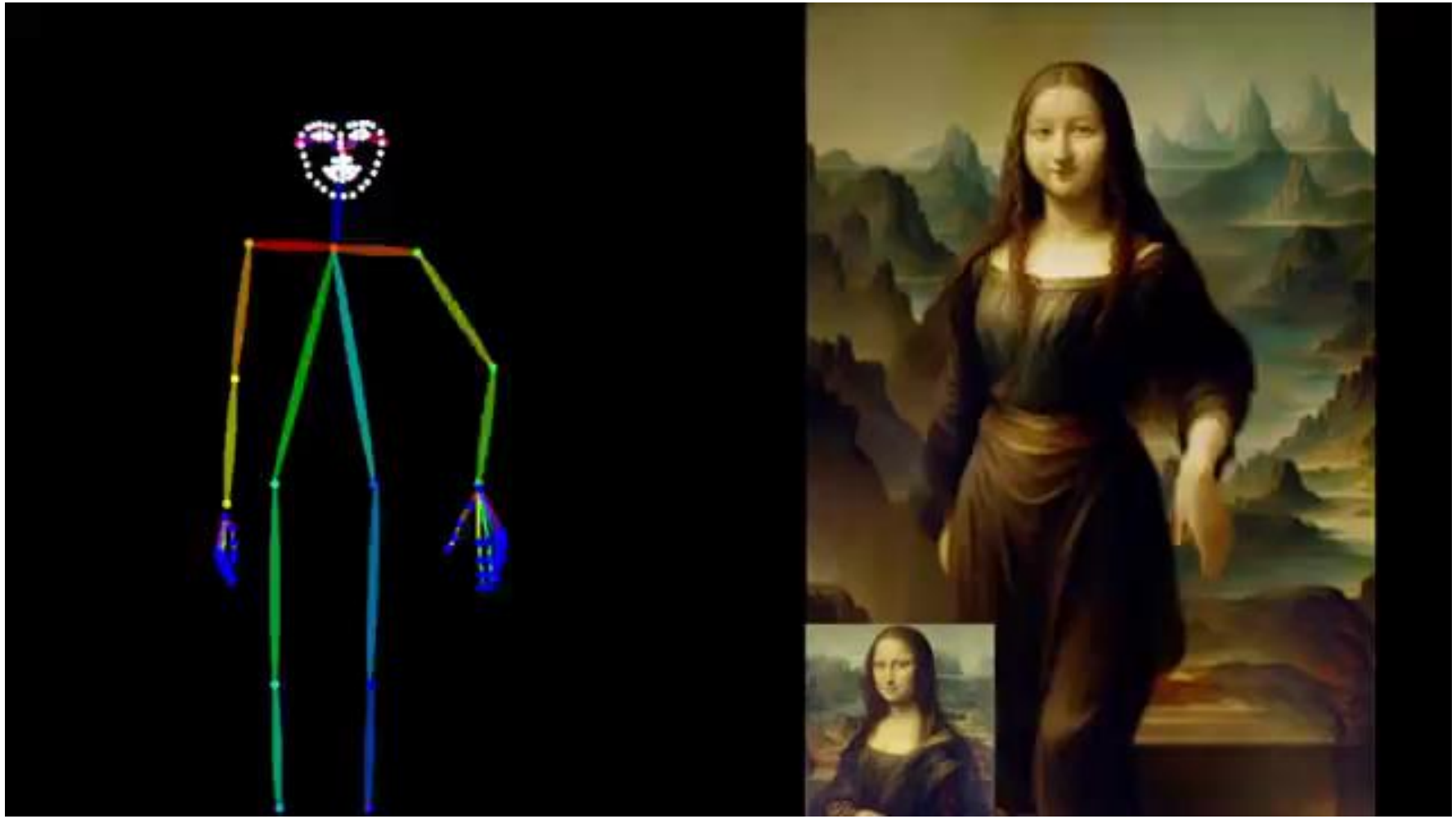
We show that diffusion models can achieve image sample quality superior to the current state-of-the-art generative models. We achieve this on unconditional image synthesis by finding a better architecture through a series of ablations. For conditional image synthesis, we further improve sample quality with classifier guidance: a simple, compute-efficient method for trading off diversity for fidelity using gradients from a classifier. We achieve an FID of 2.97 on ImageNet 128×128, 4.59 on ImageNet 256×256, and 7.72 on ImageNet 512×512, and we match BigGAN-deep even with as few as 25 forward passes per sample, all while maintaining better coverage of the distribution. Finally, we find that classifier guidance combines well with upsampling diffusion models, further improving FID to 3.94 on ImageNet 256×256 and 3.85 on ImageNet 512×512.

1 Introduction

*2021

DALL-E	**Jan 2021**	OpenAI
Midjourney V1	**Feb 2022**	Midjourney, Inc.
Midjourney V2	**Apr 2022**	
Midjourney V3	**Jul 2022**	
Stable Diffusion	**Aug 2022**	Stability AI
DALL-E 2	**Sep 2022**	OpenAI
Midjourney V4	**Nov 2022**	Midjourney, Inc.
Midjourney V5	**Mar 2023**	

Expansión en 2023



Desatando el Poder de los Modelos de Difusión

2.

Intuición de Redes Profundas ¿Cómo funcionan?

Redes Neuronales: El Poder de la Multiplicación de Matrices

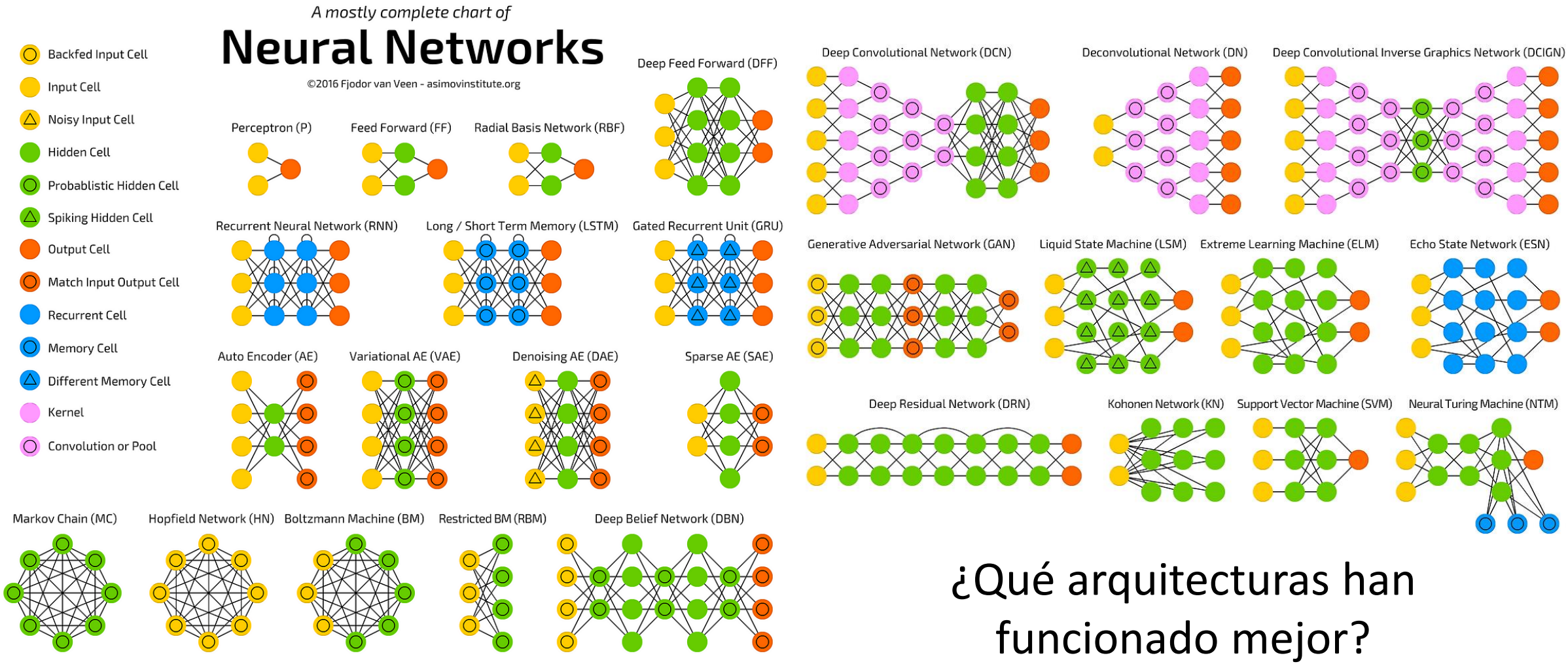
- Una red neuronal puede entenderse como una serie de multiplicaciones de matrices y aplicaciones de funciones no lineales.
- La multiplicación de matrices permite a la red combinar y transformar características de entrada de maneras complejas y no lineales.

Grayscale Image

28 x 28

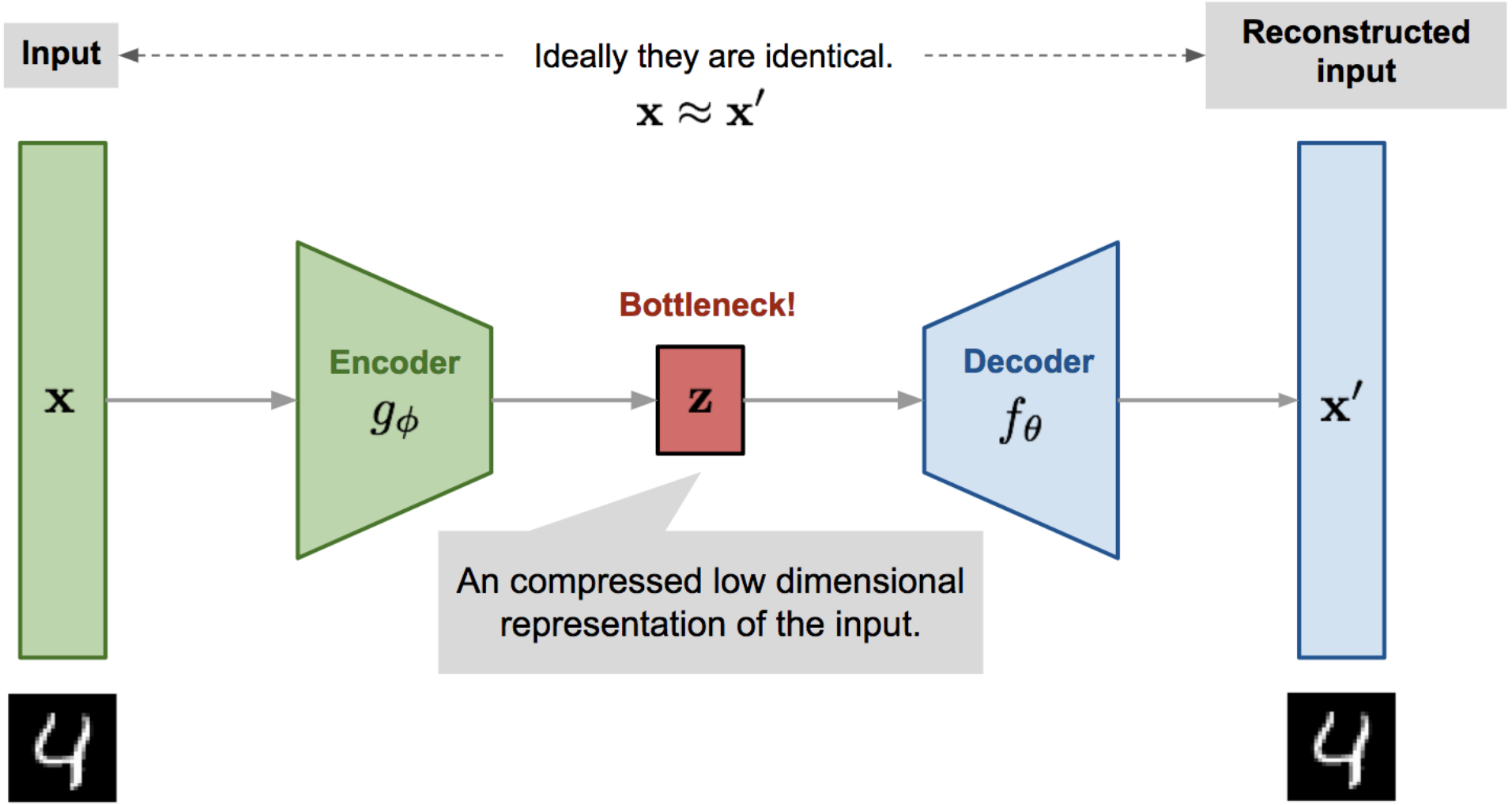
Pixel Intensities

```
[[  0   0   0   0   0   0   0   0   1  12  33  39  30   1   0   0   0   0]
 [  0   0   0   0   0   0   0   6  23   0   0   0   0  19   1   0   0   0]
 [  0   0   0   0   0   0   9   0   0  35 158 183 141   0   0   4   0   0]
 [  0   0   0   0   0  10   0  44 198 248 251 255 255 179   0  20   4   0]
 [  0   0   0   0   0  24   0 124 218 198 126  74 243 235  50   0  13   0]
 [  0   0   0   0   0   0   1  29   0   0   0 163 255 198   0  26   4   0]
 [  0   0   0   0   0   0   5   0   0   0 171 255 214  86   0  20   0   0]
 [  0   0   0   0   0   5   0 140 196 236 255 213   0   0  22   0   0   0]
 [  0   0   0   0  25   0 140 255 245 223 255 201   0  41   0   1   0   0]
 [  0   0   0   0   5   0  56  99  68   0 228 231  31   0  13   0   0   0]
 [  0   0   6  23   9   5   0   0  21   0 219 248  88   0  20   0   0   0]
 [  0   7   0   0   0  35  14  33  34   0 234 223   0  17   8   0   0   0]
 [ 21   0 118 178   0   0   0   0   0 174 255 145   0  24   1   0   0   0]
 [ 14   0  63 242 229 160  67  96 187 255 204   0  16   1   0   0   0   0]
 [  0  21   0 124 234 255 252 251 248 185   0  13   8   0   0   0   0   0]
 [  0   0  17   0   0 138 180 170  65   0  18   8   0   0   0   0   0   0]
 [  0   0   0  21  13   0   0   0   0  29   4   0   0   0   0   0   0   0]
 [  0   0   0   0   5  30  38  36  15   0   0   0   0   0   0   0   0   0]]
```

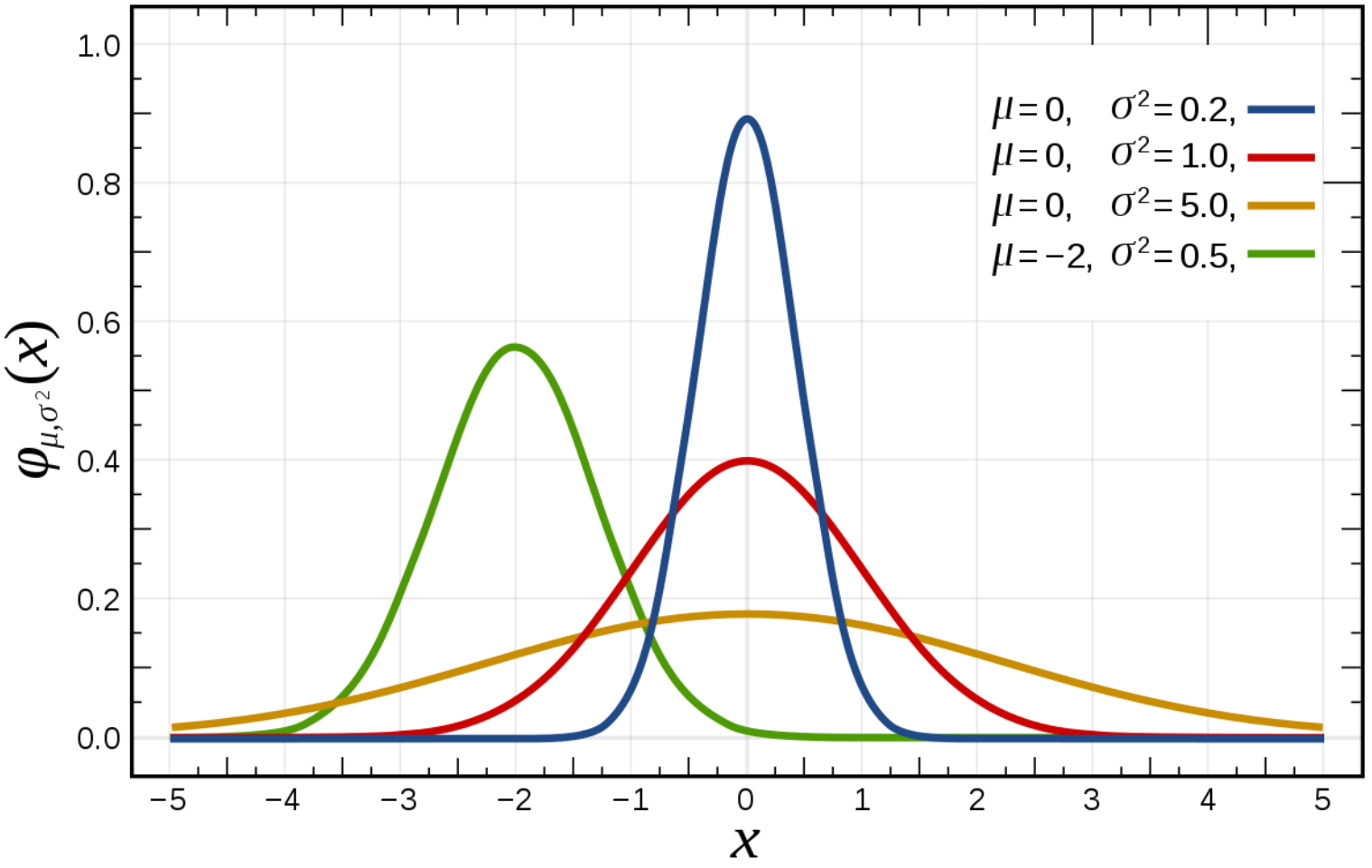


¿Qué arquitecturas han funcionado mejor?

Reducir la dimensionalidad: Autoencoders



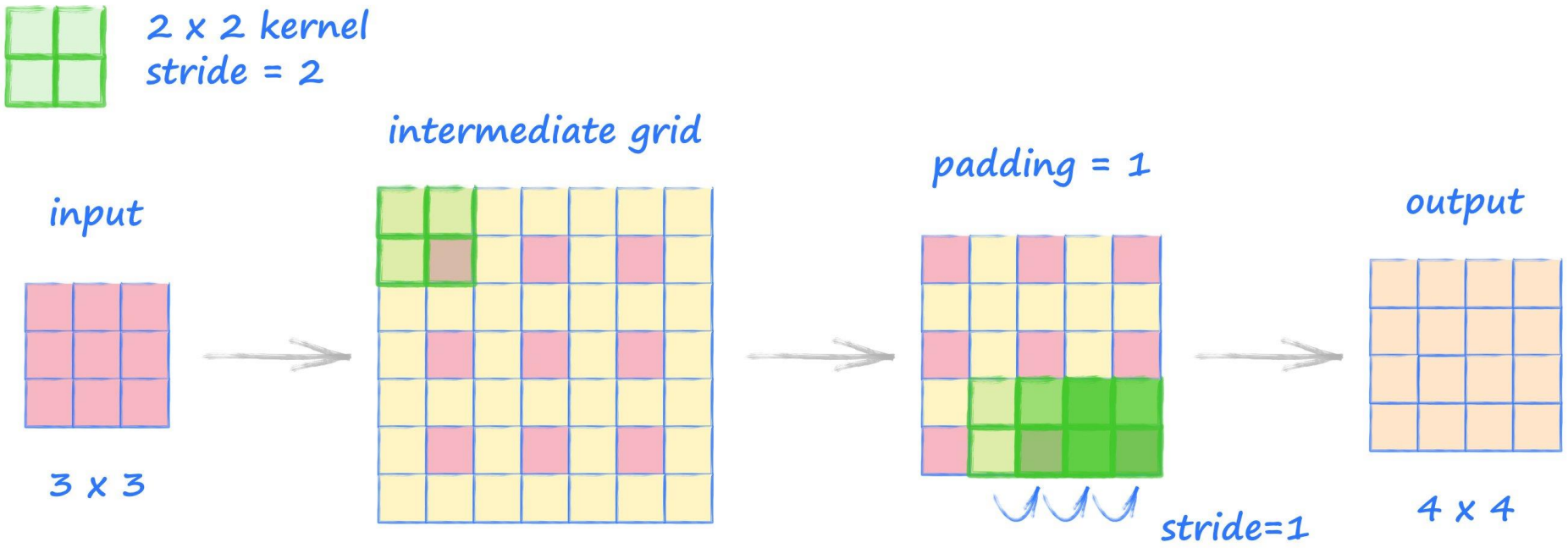
En estadística es similar al concepto de distribución probabilística



Funcionan bien para resolver problemas simples que no están influenciados por la escala

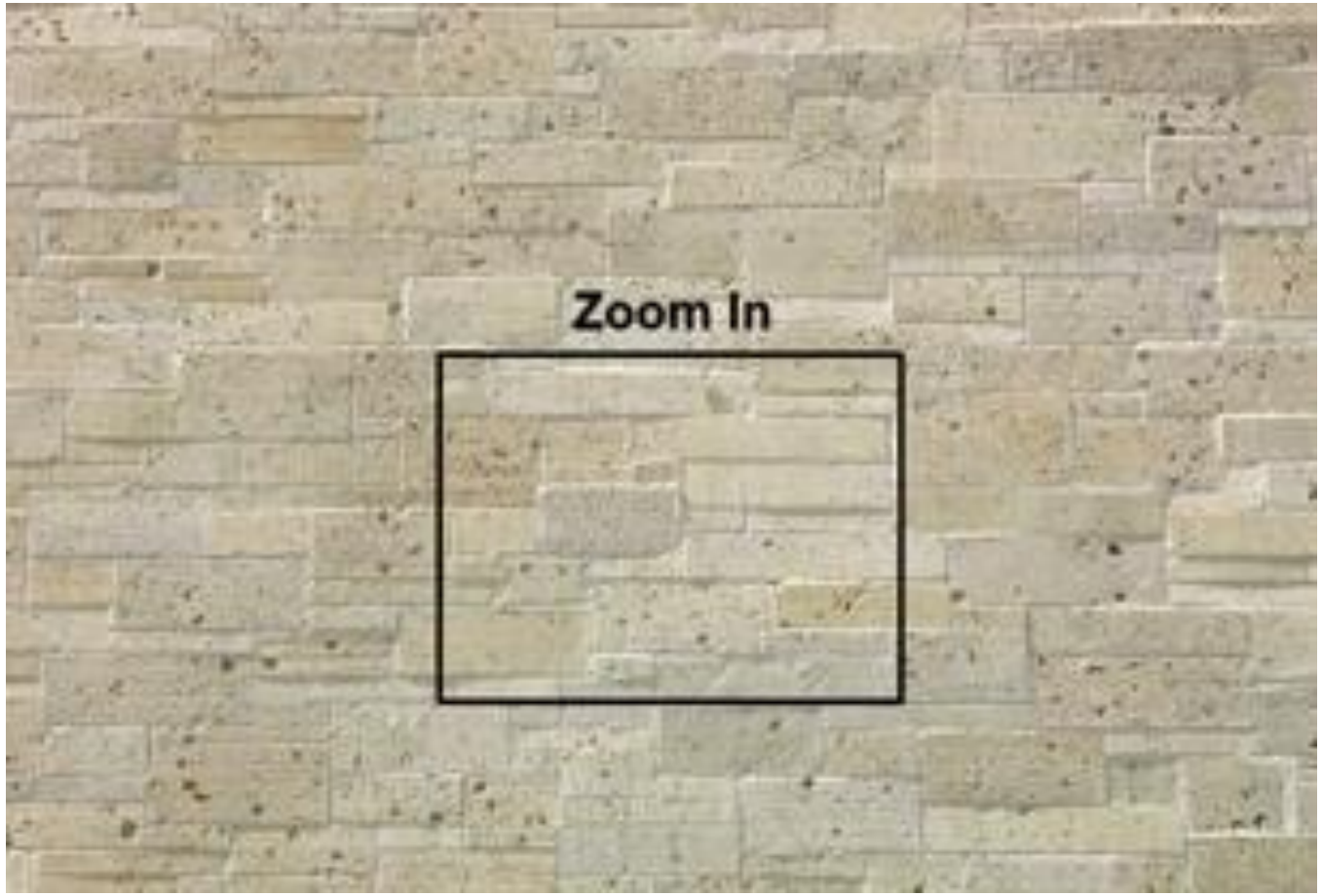
Redes Convolucionales: El Poder de la Multiplicación de Matrices

A distintas escalas se resuelven distintos problemas, estas redes se enfocan en relacionar o acotar el problema a distintas escalas.

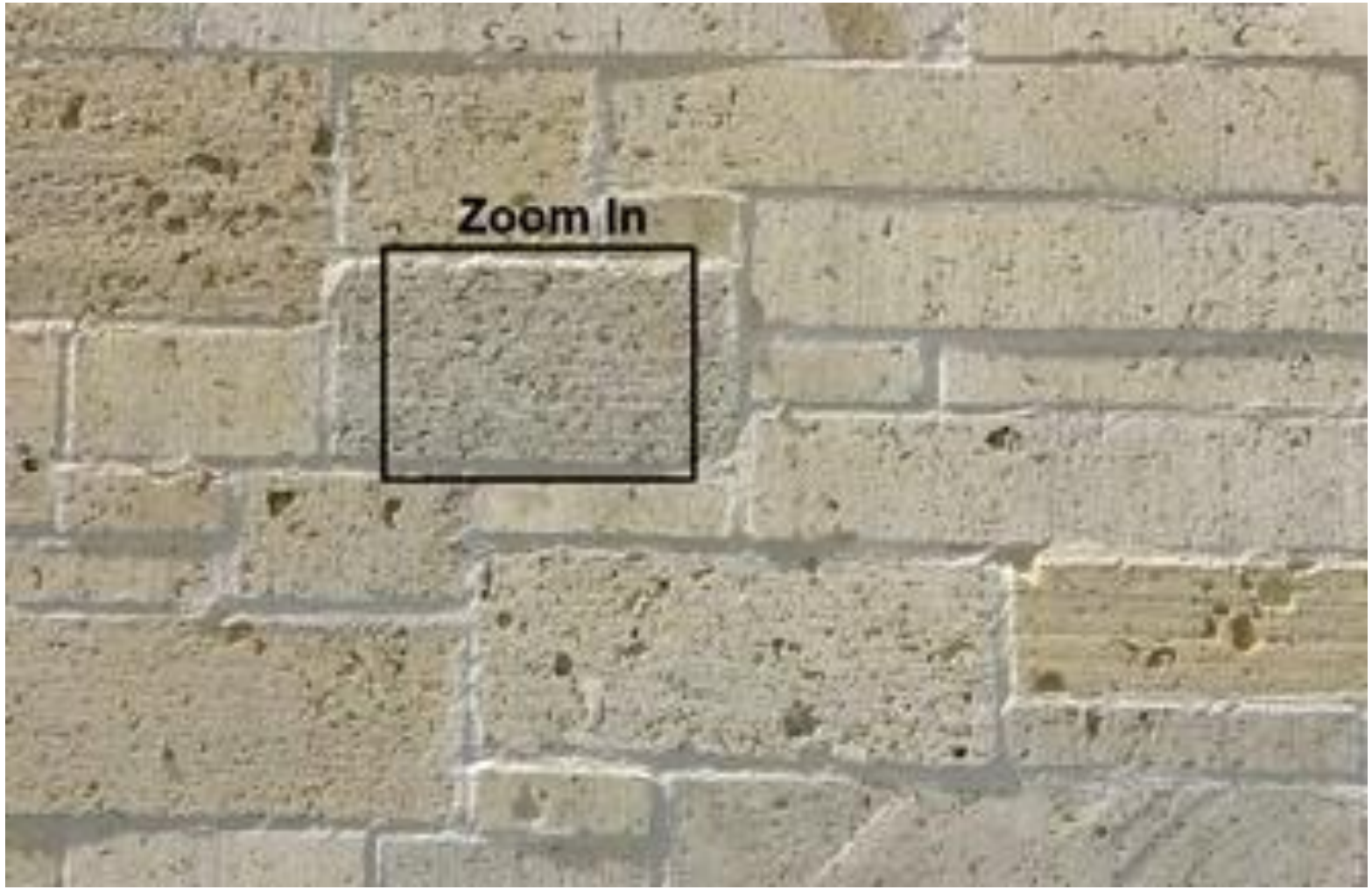


En geoestadística es similar al concepto de estacionariedad

- Cuando el promedio y la varianza se mantienen constantes en el espacio.
- ¿Esta serie de tiempo presenta estacionariedad?
- ¿Qué métricas usariamos?

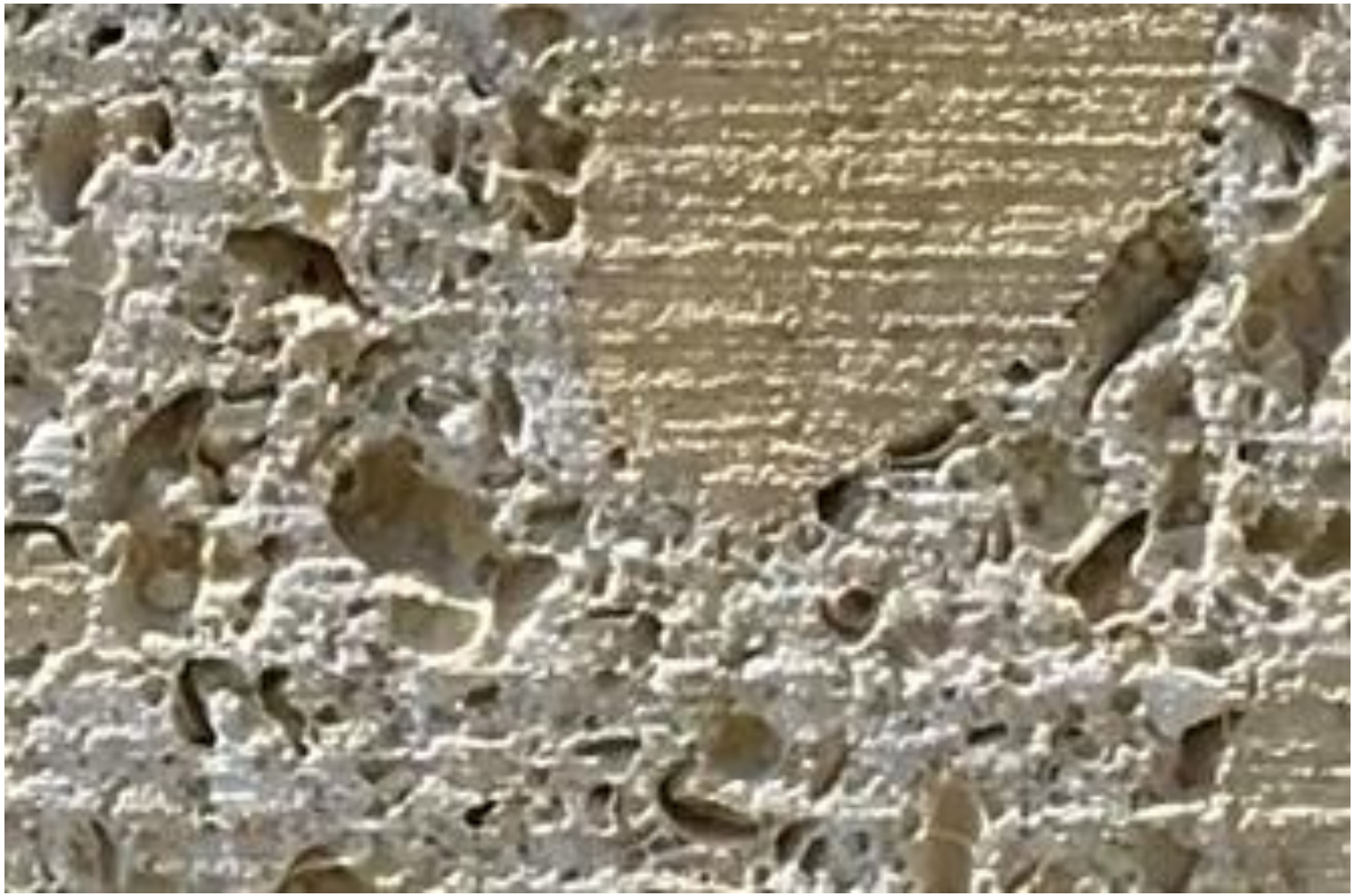


¿Es estacionario?



¿Es estacionario?

¿Es estacionario?

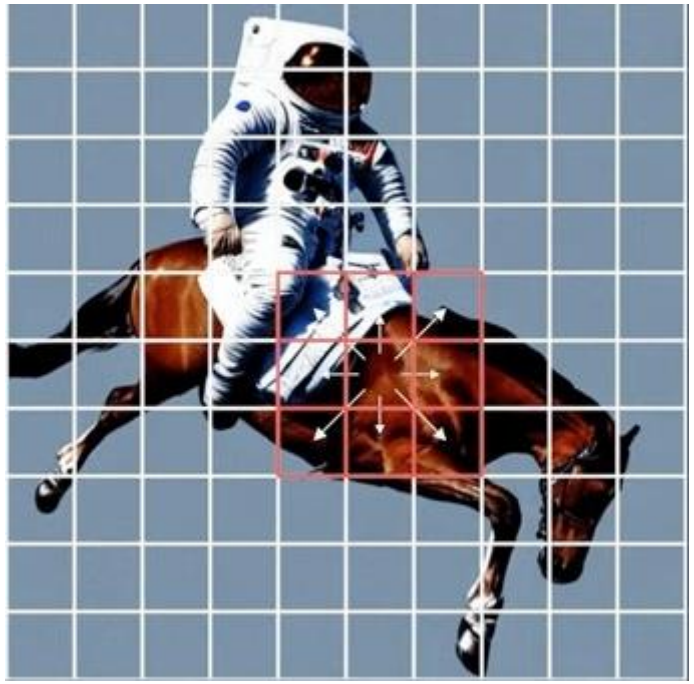


Cada escala tiene características particulares por lo que segmentar la imagen nos puede ayudar a capturar estas características.

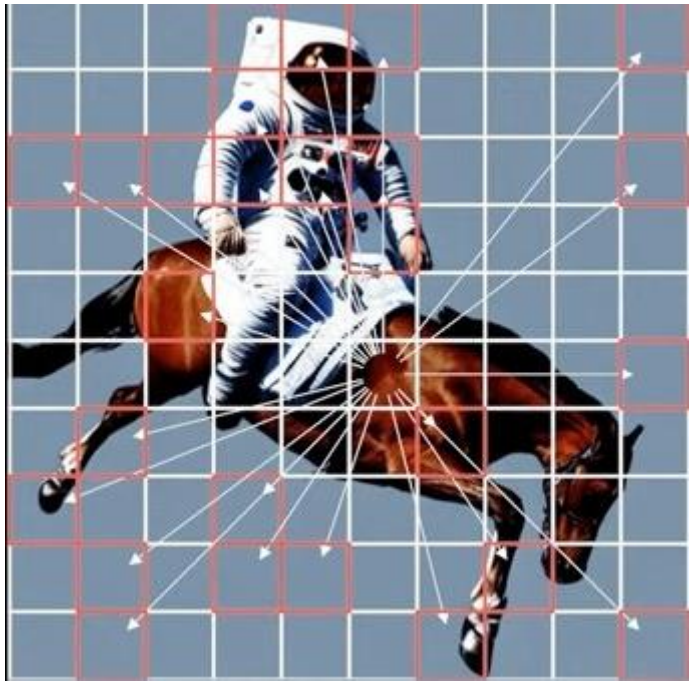
Transformers: Mejorando la Atención

En vez de enfocarse en ventanas específicas, la atención expande el enfoque en áreas particulares del dataset.

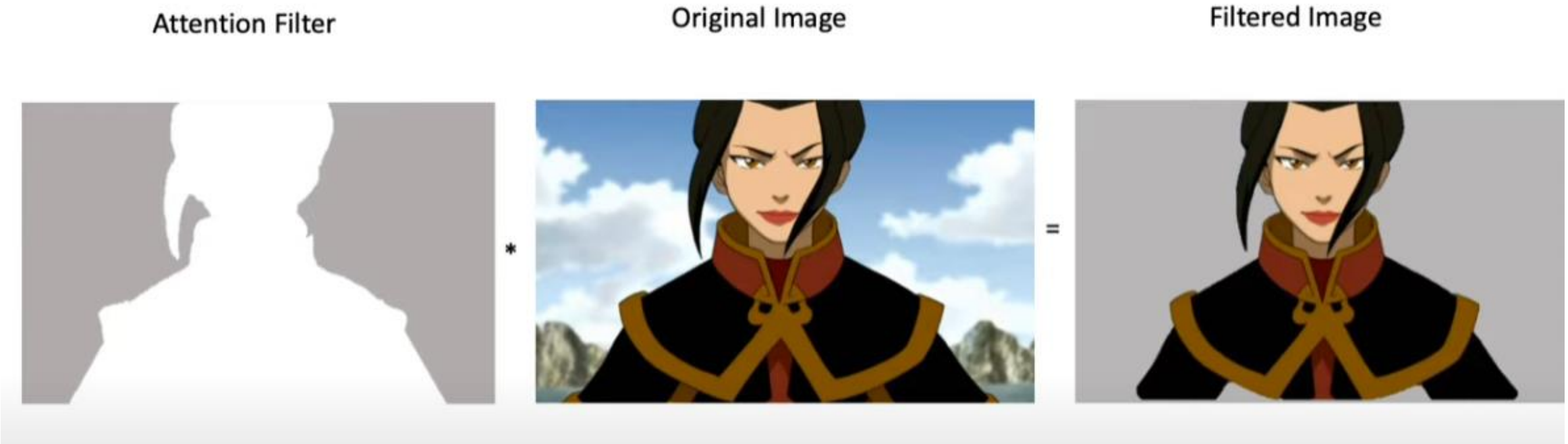
Convolución



Atención



La atención filtra la información necesaria a ser multiplicada



Visual Guide to Transformer Neural Networks - (Episode 2) Multi-Head & Self-Attention

https://www.youtube.com/watch?v=mMa2PmYJlCo

El Problema de encontrar los parámetros

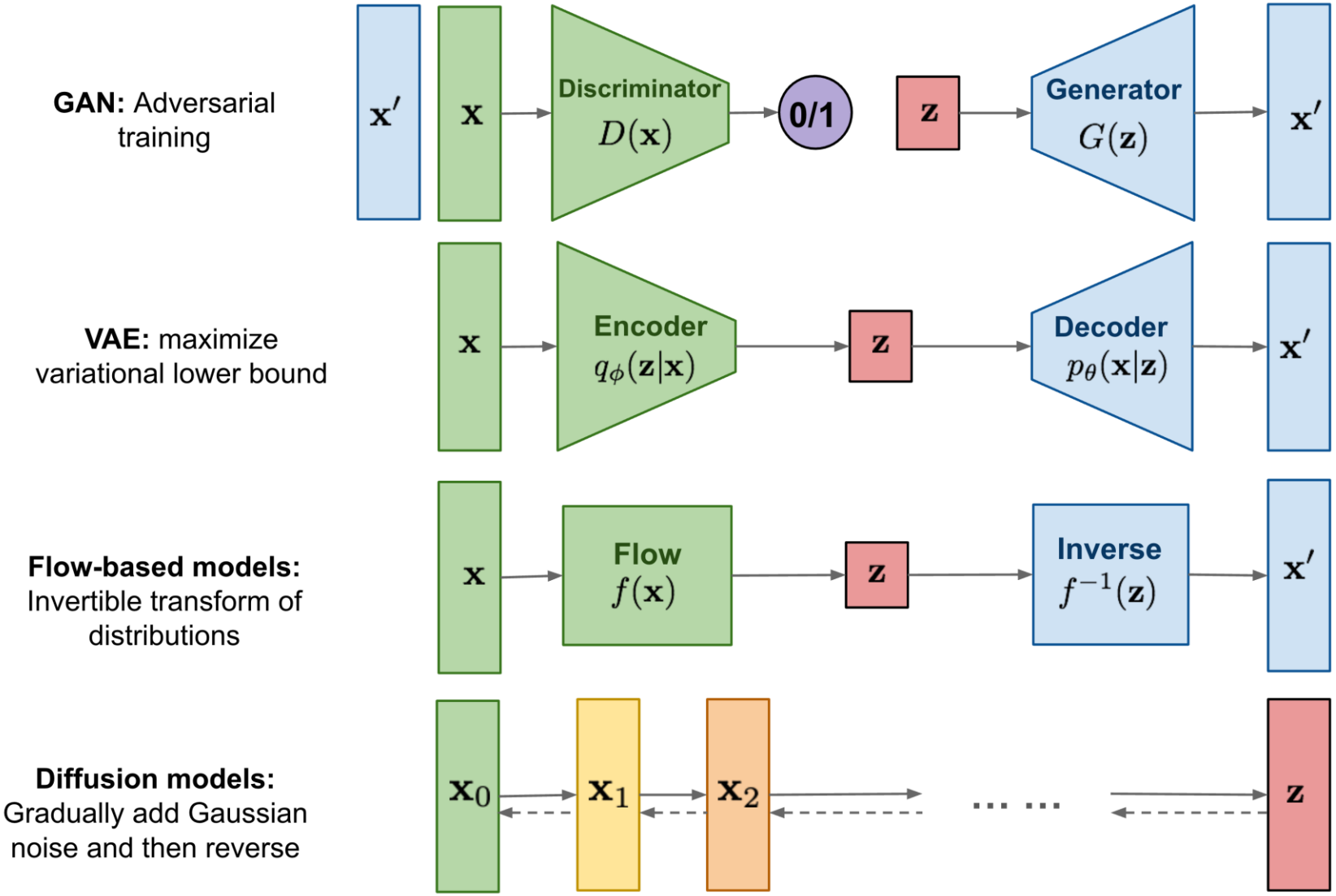
“AI has thus far gotten to where it is via brute force”

Competencia por poder computacional

- Para realizar muchas de las aplicaciones modernas de redes profundas es necesario poder computacional inmenso, creando una desventaja para países como Perú.
- Debido a cómo funcionan las redes de difusión se pueden conseguir resultados similares con hasta 10 veces menos parámetros.

3.

Intuición sobre Redes de Difusión ¿Cómo funcionan?

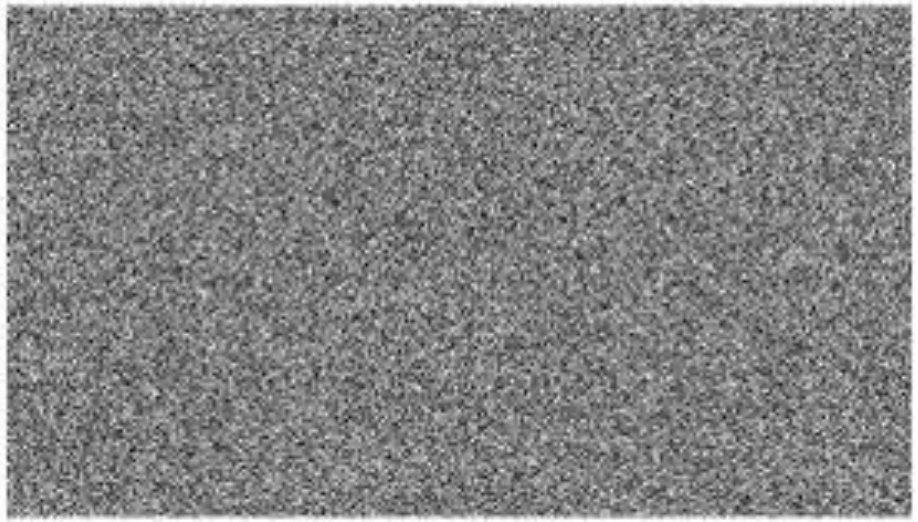


Las arquitecturas exitosas de redes neuronales profundas buscan reducir o encontrar variables que expliquen mejor el traning set. Sin embarg un modelo de difusión funciona distinto en Las arquitecturas avanzadas de redes neuronales profundas se enfocan en minimizar o identificar variables clave para optimizar el conjunto de entrenamiento. Por otro lado, los modelos de difusión operan bajo un principio distinto.

Entender el “ruido” es lo principal

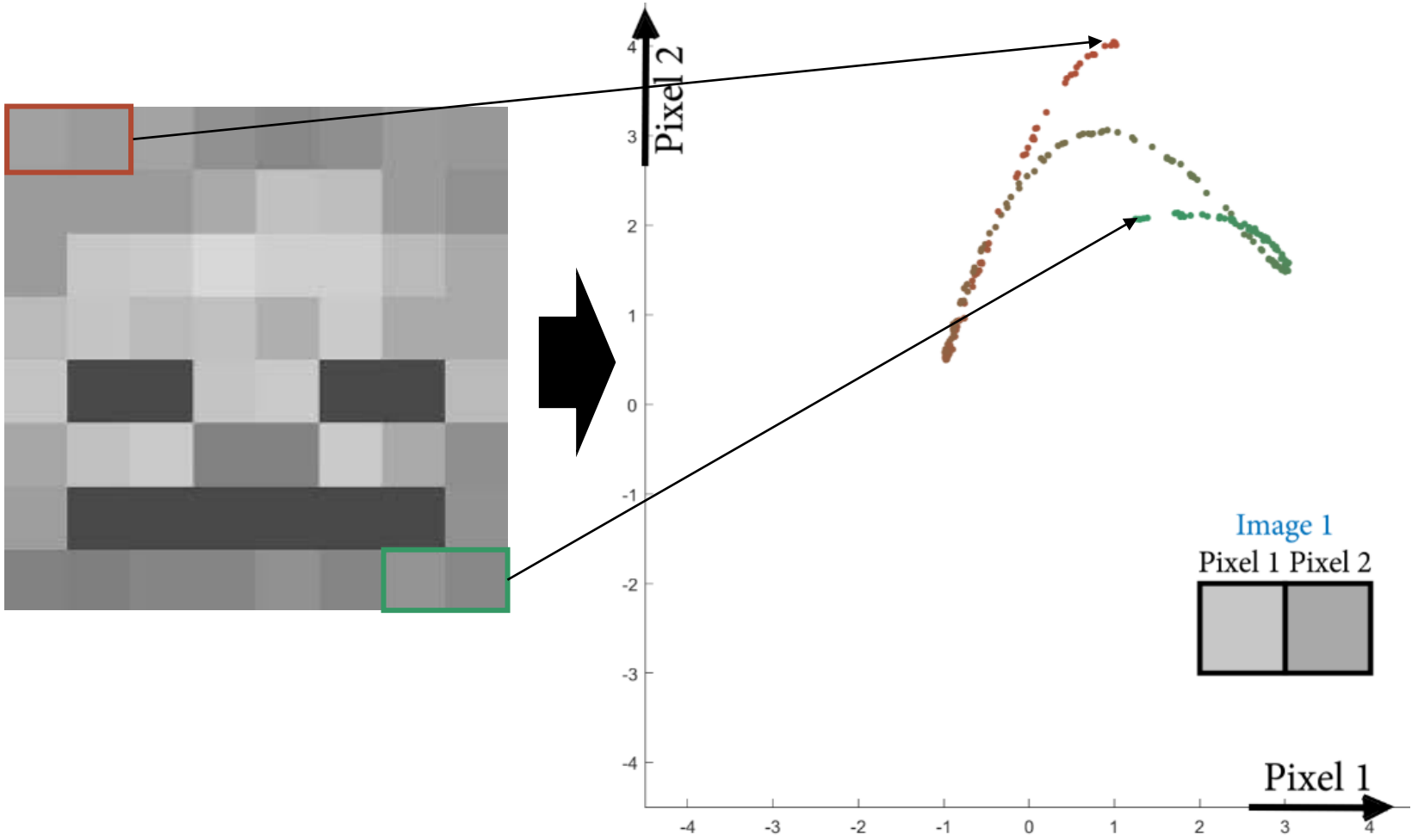
Add noise at training time

Denoise at synthesis time



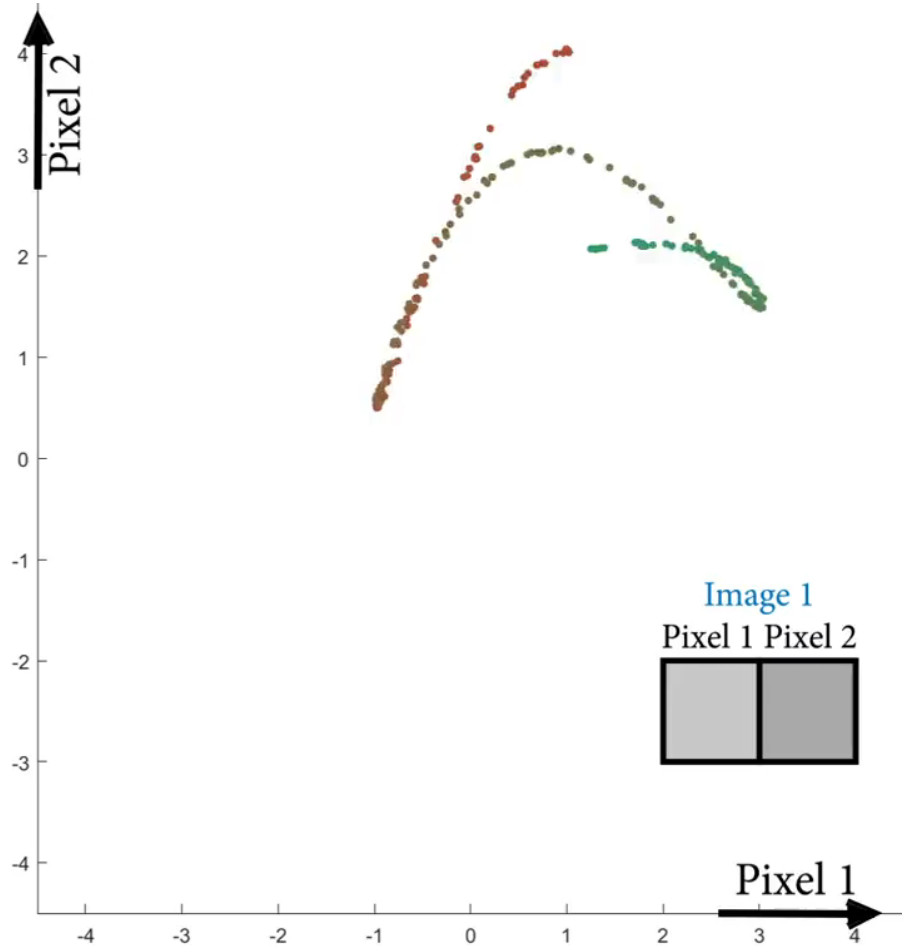
Entender el “ruido” es lo principal

Tomaremos un conjunto de de 2 píxeles en una imagen de escala de grises, cada uno representado como un punto 2D.



Agregando ruido “Puro”

Inicialmente, el conjunto de imágenes forma una variedad de "imágenes realistas" (aquí, ficticias). Progresivamente añado un valor aleatorio gaussiano a cada píxel, produciendo un movimiento browniano. Después de muchas iteraciones, la distribución de las imágenes todavía no es una gaussiana centrada: la variedad todavía está presente.



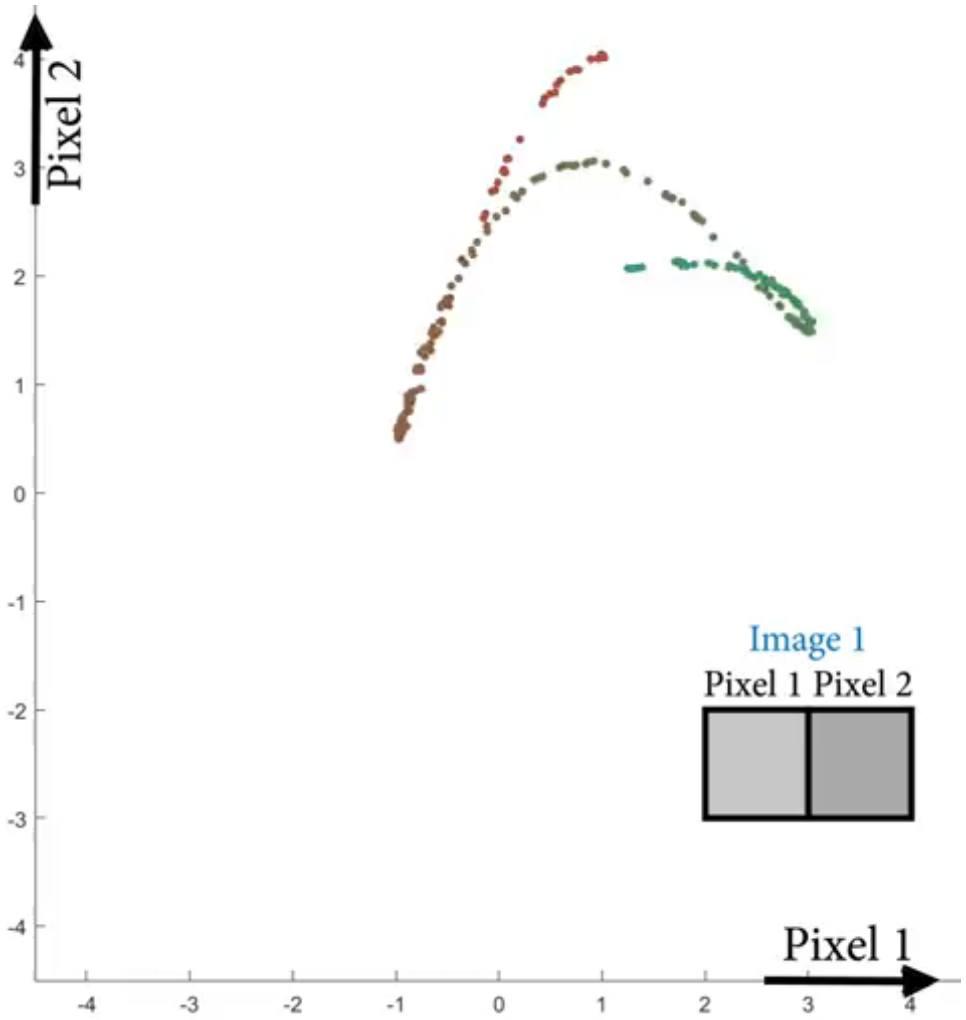
Combinando el ruido progresivamente

En cambio, para transformar la variedad de "imágenes realistas" en una distribución gaussiana, es necesario combinar progresivamente cada imagen de entrada con una gaussiana. El iterado resultante es simplemente una combinación lineal de la imagen en el paso de tiempo anterior y un valor aleatorio gaussiano.

$$I^{t+1} = \sqrt{\alpha_t}\, I^t + \sqrt{1-\alpha_t}\, \varepsilon_{t\to t+1}$$

with $\varepsilon_{t\to t+1} \sim \mathcal{N}(0,1)$

α_t some parameter (could be constant)



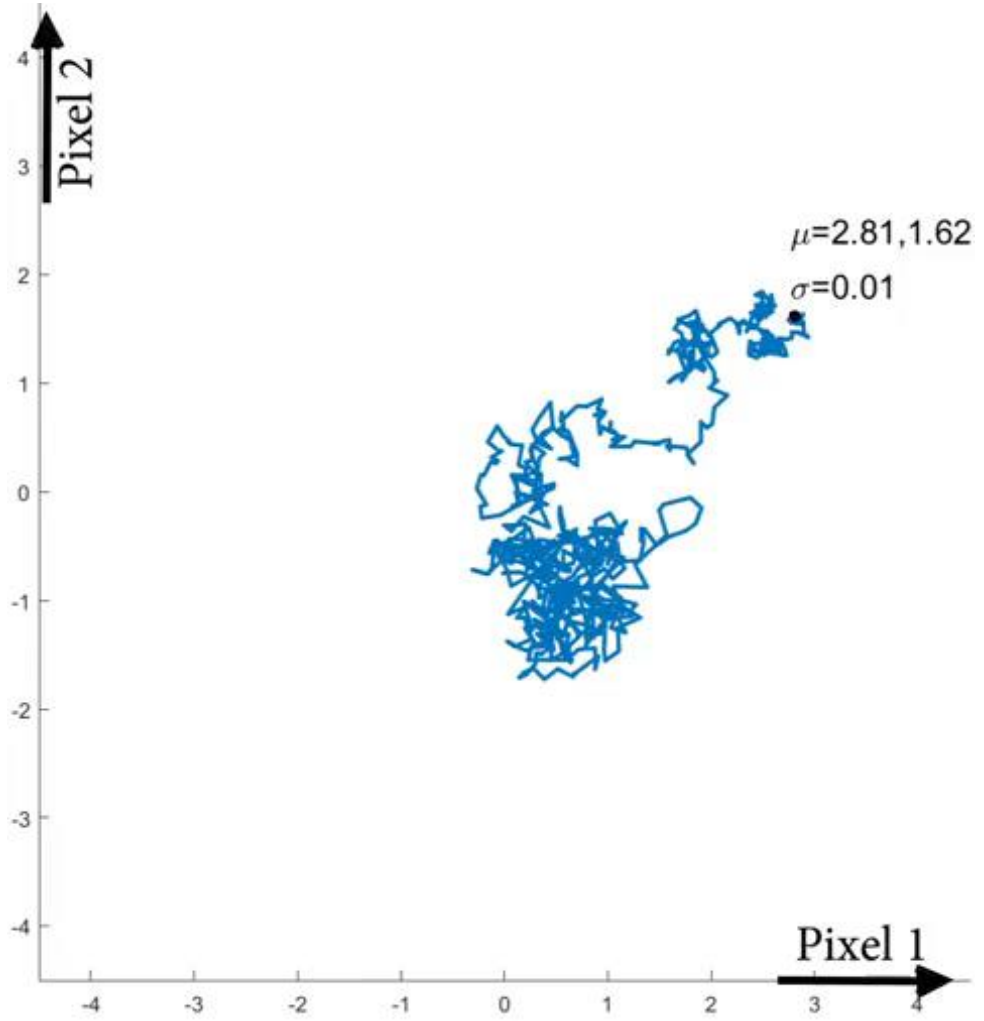
Combinando el ruido progresivamente

Para una imagen inicial I^0 sin ruido, podemos predecir su aspecto en cualquier momento t mezclándola directamente con ruido gaussiano, sin necesidad de iteraciones intermedias. Los valores \alpha_i son menores que 1 y disminuyen con el tiempo, lo que hace que la imagen original I^0 se desvanezca gradualmente.

$$I^t = \sqrt{\overline{\alpha_t}}\ I^0 + \sqrt{1-\overline{\alpha_t}}\ \varepsilon_{0\to t}$$

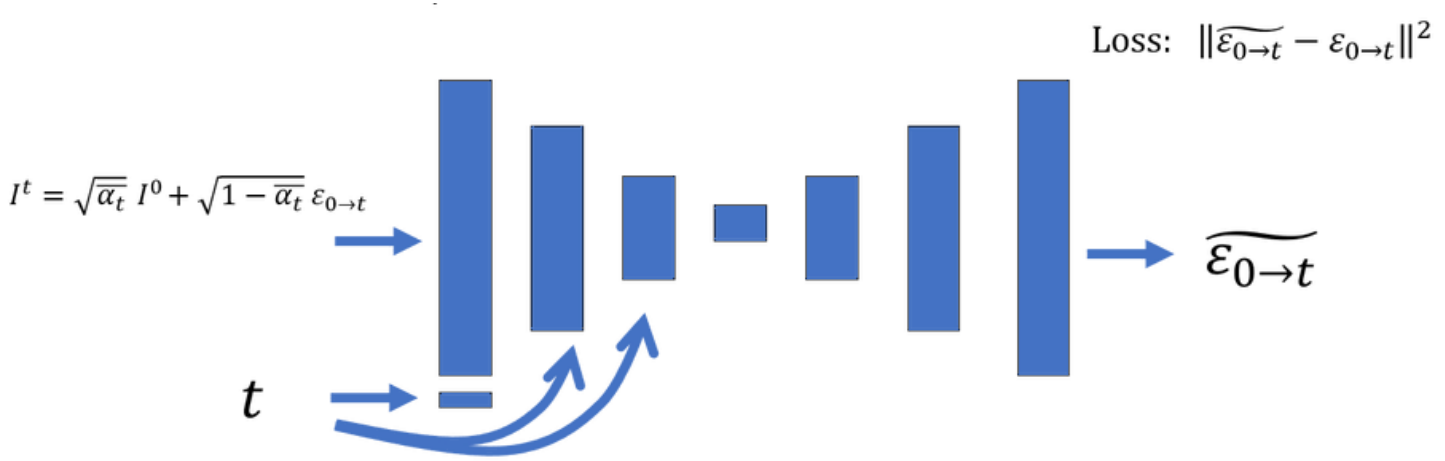
with $\varepsilon_{0\to t} \sim \mathcal{N}(0,1)$

$$\overline{\alpha_t} = \prod_{i=1}^{t} \alpha_i$$



Entrenar una red de difusión

Ahora podemos entrenar una red neuronal para eliminar ruido, por ejemplo, una U-Net. La alimentamos con un gran conjunto de datos de imágenes con cantidades aleatorias de ruido generadas por la fórmula anterior, e intentamos predecir el ruido. Minimizamos una pérdida que es simplemente el error cuadrático medio (MSE) entre el ruido de entrada y el ruido predicho.



- Feed the network with many noisy images at random times $t \in [0, t_{max}]$

Considerar una función que demore más en destruir la señal

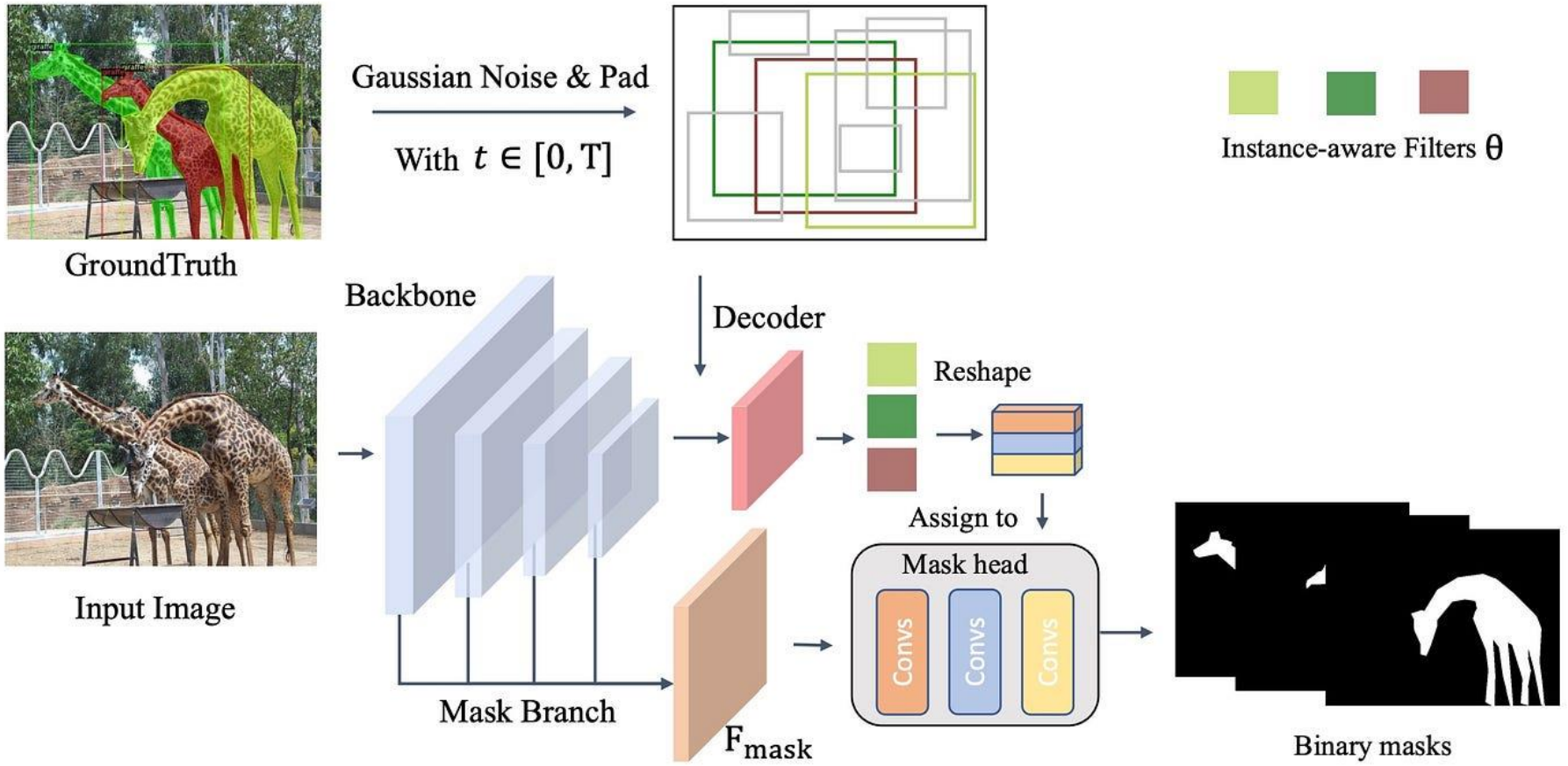
4.

Aplicaciones Diversas de Modelos de DifusiónA

Inpainting con modelos de difusión

Input Image.

Segmentación

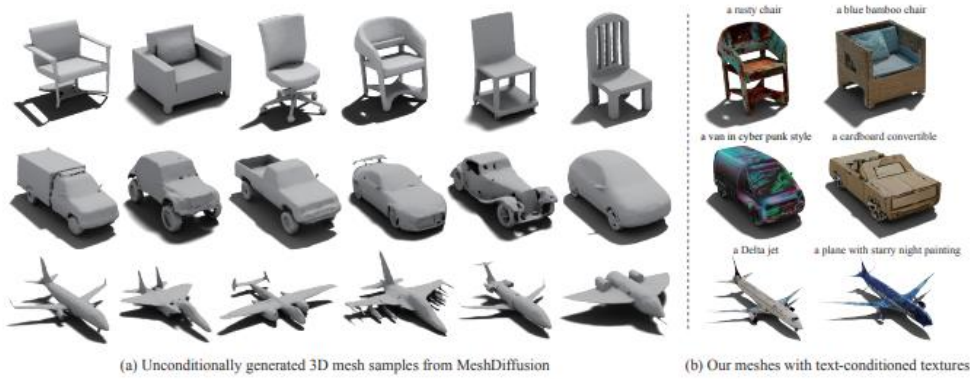


Se puede aplicar en 3D

MESHDIFFUSION: SCORE-BASED GENERATIVE 3D MESH MODELING

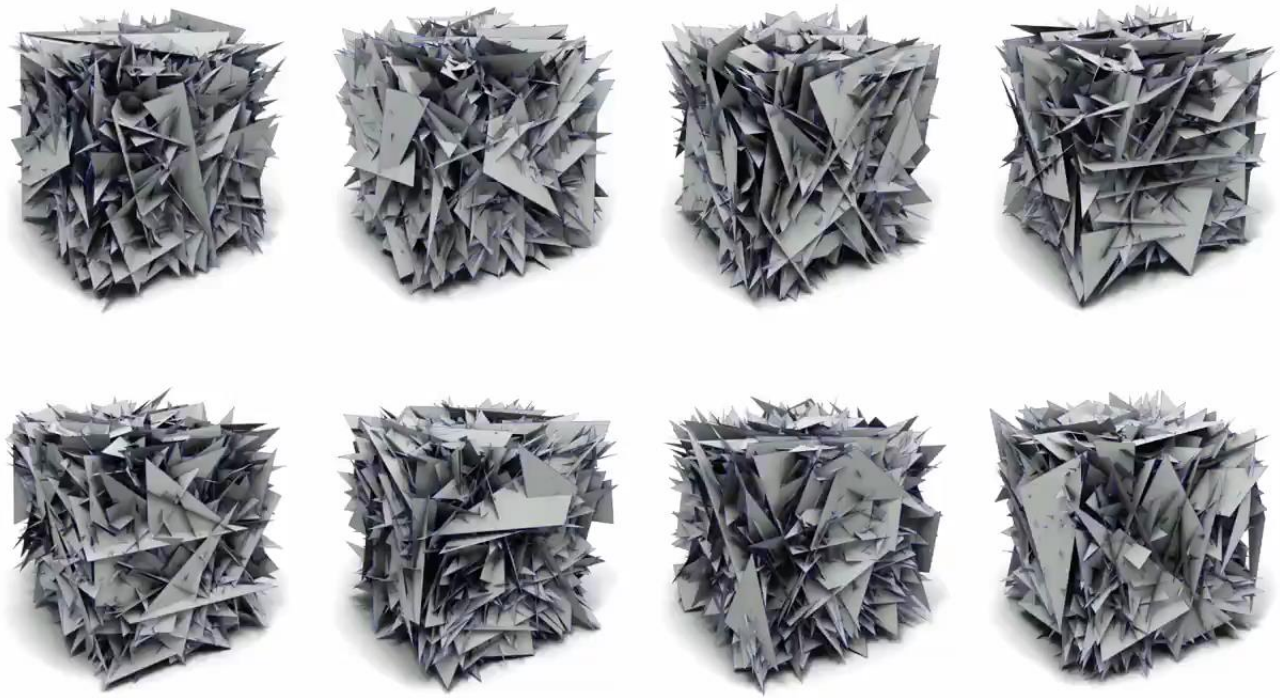
Zhen Liu[1,2*], Yao Feng[2,3], Michael J. Black[2], Derek Nowrouzezahrai[4], Liam Paull[1], Weiyang Liu[2,5]
[1]Mila, Université de Montréal [2]Max Planck Institute for Intelligent Systems - Tübingen
[3]ETH Zürich [4]McGill University [5]University of Cambridge

Project Page: meshdiffusion.github.io



(a) Unconditionally generated 3D mesh samples from MeshDiffusion (b) Our meshes with text-conditioned textures

Figure 1: (a) Unconditionally generated 3D mesh samples randomly selected from the proposed *MeshDiffusion*, a simple diffusion model trained on a direct parametrization of 3D meshes without bells and whistles. (b) 3D mesh samples generated by *MeshDiffusion* with text-conditioned textures from [39]. *MeshDiffusion* produces highly realistic and fine-grained geometric details while being easy and stable to train.



Generar código utilizable

CODEFUSION: A Pre-trained Diffusion Model for Code Generation

Mukul Singh
Microsoft
Delhi, India

José Cambronero
Sumit Gulwani
Vu Le
Microsoft
Redmond, US

Carina Negreanu
Microsoft Research
Cambridge, UK

Gust Verbruggen
Microsoft
Keerbergen, Belgium

Abstract

Imagine a developer who can only change their last line of code—how often would they have to start writing a function from scratch before it is correct? Auto-regressive models for code generation from natural language have a similar limitation: they do not easily allow reconsidering earlier tokens generated. We introduce CODEFUSION, a pre-trained diffusion code generation model that addresses this limitation by iteratively denoising a complete program conditioned on the encoded natural language. We evaluate CODEFUSION on the task of natural language to code generation for Bash, Python, and Microsoft Excel conditional formatting (CF) rules. Experiments show that CODEFUSION (75M parameters) performs on par with state-of-the-art auto-regressive systems (350M–175B parameters) in top-1 accuracy and outperforms them in top-3 and top-5 accuracy, due to its better balance in diversity versus quality.

approaches then select the vocabulary token with the closest embedding. In the code domain, where there are many syntactic and semantic constraints between tokens, independently projecting embeddings back to tokens can yield invalid programs.

We propose CODEFUSION, a natural language to code (NL-to-code) model that combines an encoder-decoder architecture (Raffel et al., 2020) with a diffusion process. The encoder maps the NL into a continuous representation, which is used by the diffusion model as an additional condition for denoising random Gaussian noise input. To generate syntactically correct code, we then feed the denoised embeddings to a transformer decoder, with full self-attention and cross attention with the embedded utterance, to obtain probability distributions over code tokens. Finally, we select the token with the highest probability at each index.

To pre-train CODEFUSION for code generation, we extend the continuous paragraph denoising

System description			Python (CodeBERT)		
System	**Model**	**#P**	**top-1**	**top-3**	**top-5**
T5	t5-large	770M	80.4	82.3	84.8
CodeT5	codet5-large	770M	80.5	83.1	85.0
GPT-3	text-davinci-003	175B	**82.5**	83.7	85.8
ChatGPT	gpt-3.5-turbo	20B	80.6	82.5	83.9
StarCoder	starcoder	15.5B	79.2	82.0	84.1
CodeT5+	codet5p-16b	16B	79.6	82.1	84.5
CodeGen	codegen-350m	350M	80.1	81.8	83.7
Diffusion-LM	Custom	50M	70.4	74.3	76.5
GENIE	Custom	93M	73.2	77.1	80.3
CODEFUSION	**Custom**	75M	80.7	**86.3**	**90.3**

NL:	`Copy the content of file 'file.txt' to file 'file2.txt'`
Target:	`shutil.copy('file.txt', 'file2.txt')`
$t = 1200$	`Ss ec between Useful 144 Location copyright pen they destinat Path caret corev adapter NameAnd ġiterating Ġface Ġpass ĠFH`
$t = 1000$	`(.* for copyInt filefiletxt, filefiletxt queryResult SE Screen`
$t = 800$	`destutil..copyInt'filetxt', 'filetxt') return 'filetxt'`
$t = 600$	`destutil.copyInt'file.txt', 'fi2.txt')`
$t = 400$	`shutil.copyInt'file.txt', 'file.txt')`
$t = 200$	`shutil.copy('file.txt', 'file2.txt')`
$t = 0$	`shutil.copy('file.txt', 'file2.txt')`

No es necesario que sea ruido

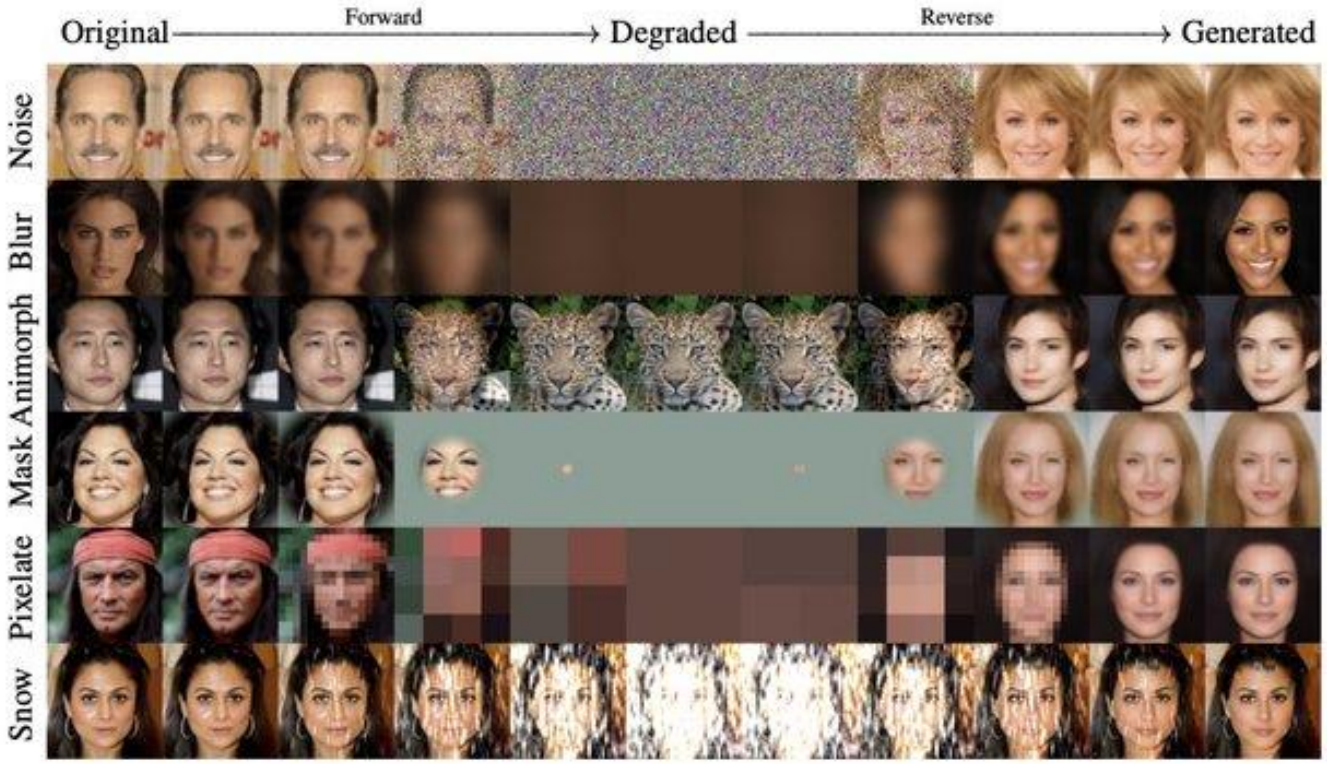


Figure 1: Demonstration of the forward and backward processes for both hot and cold diffusions. While standard diffusions are built on Gaussian noise (top row), we show that generative models can be built on arbitrary and even noiseless/cold image transforms, including the ImageNet-C *snowification* operator, and an *animorphosis* operator that adds a random animal image from AFHQ.

Flujo de la Charla

Parte Teórica

- Evolución del Estado del Arte
- Intuición detrás de las Redes Profundas
- Intuición detrás de los Modelos de Difusión
- Principios Básicos
- Aplicaciones Diversas

Coffee Break

(25 minutos)

Parte Práctica (4:00-6:00 pm)

- Lab 1: Uso Básico de Modelos
- Lab 2: Entrenamiento Básico
- Lab 3: Control sobre el Modelo
- Lab 4: Mejorando Eficiencia
- Recomendaciones Generales

Labs Basados en “How Diffusion Models Work”

https://learn.deeplearning.ai/diffusion-models/

Objetivo de la parte práctica

- **Tienes: muchas imágenes de pixel art.**

Estos son tus datos de entrenamiento.

- **Quieres: aún más pixel art.**

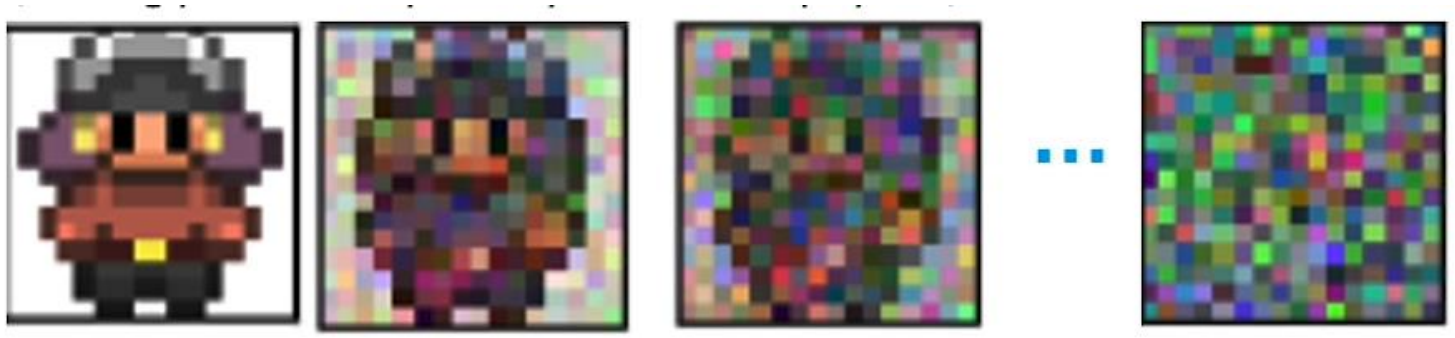
Esta es una red neuronal que genera más pixel art para ti.

Creando imágenes útiles para una red neuronal

Quieres que una red neuronal aprenda:

- Detalles finos
- Contornos generales
- Todo lo intermedio

Una forma es añadir diferentes niveles de ruido a los datos de entrenamiento



Entrenando una red neuronal

La red neuronal aprende a tomar diferentes imágenes con ruido y convertirlas de nuevo en sprites.

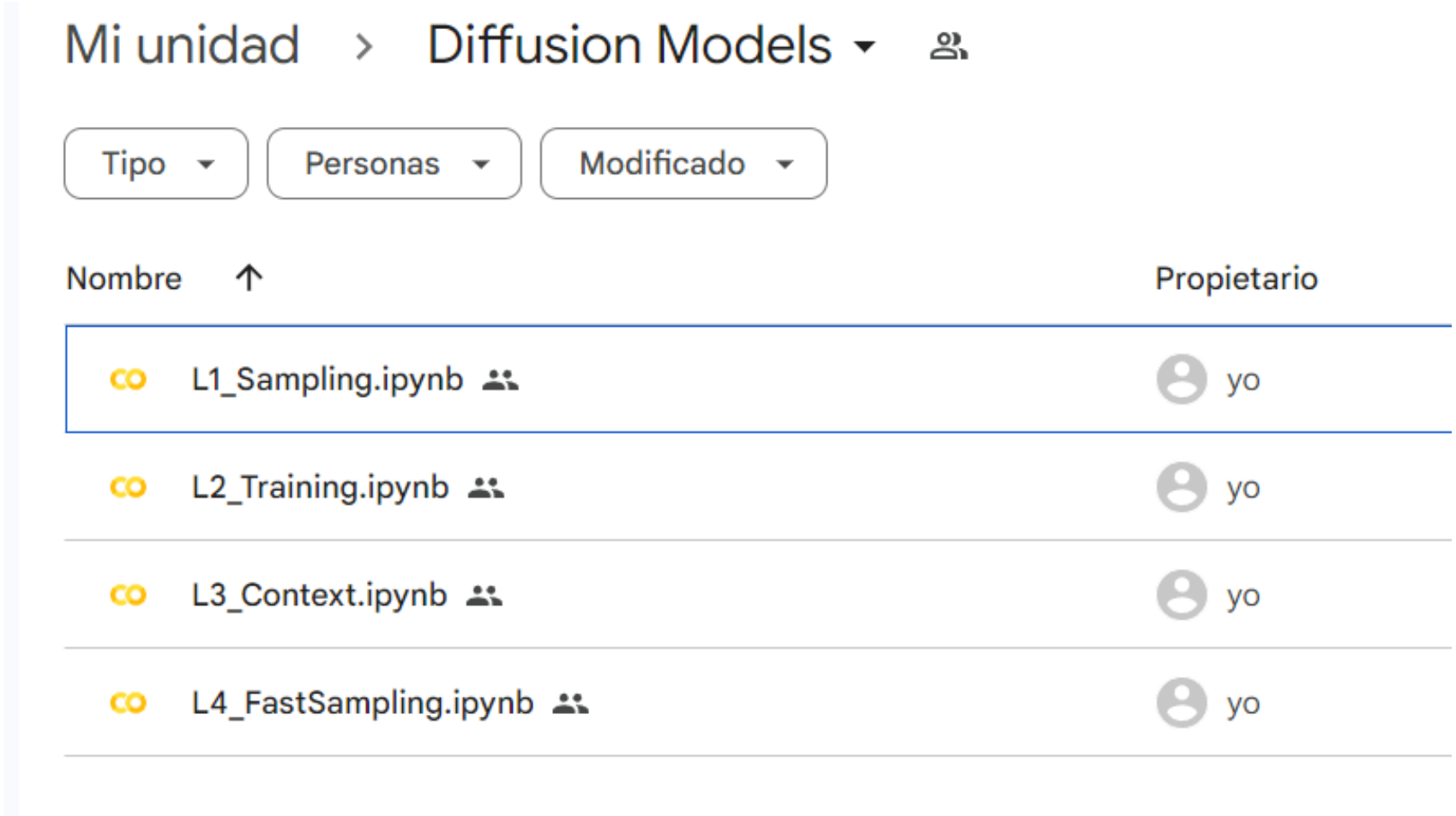
Aprende a eliminar el ruido que añadiste.

El nivel de ruido "Sin idea" es importante porque está **normalmente distribuido**.

Entonces, cuando le pides a la red neuronal un nuevo sprite:

- Puedes muestrear ruido de la distribución normal
- Obtener un sprite completamente nuevo usando la red para eliminar el ruido
- Ahora, puedes obtener aún más sprites, más allá de tus datos de entrenamiento.

Notebooks a Utilizar



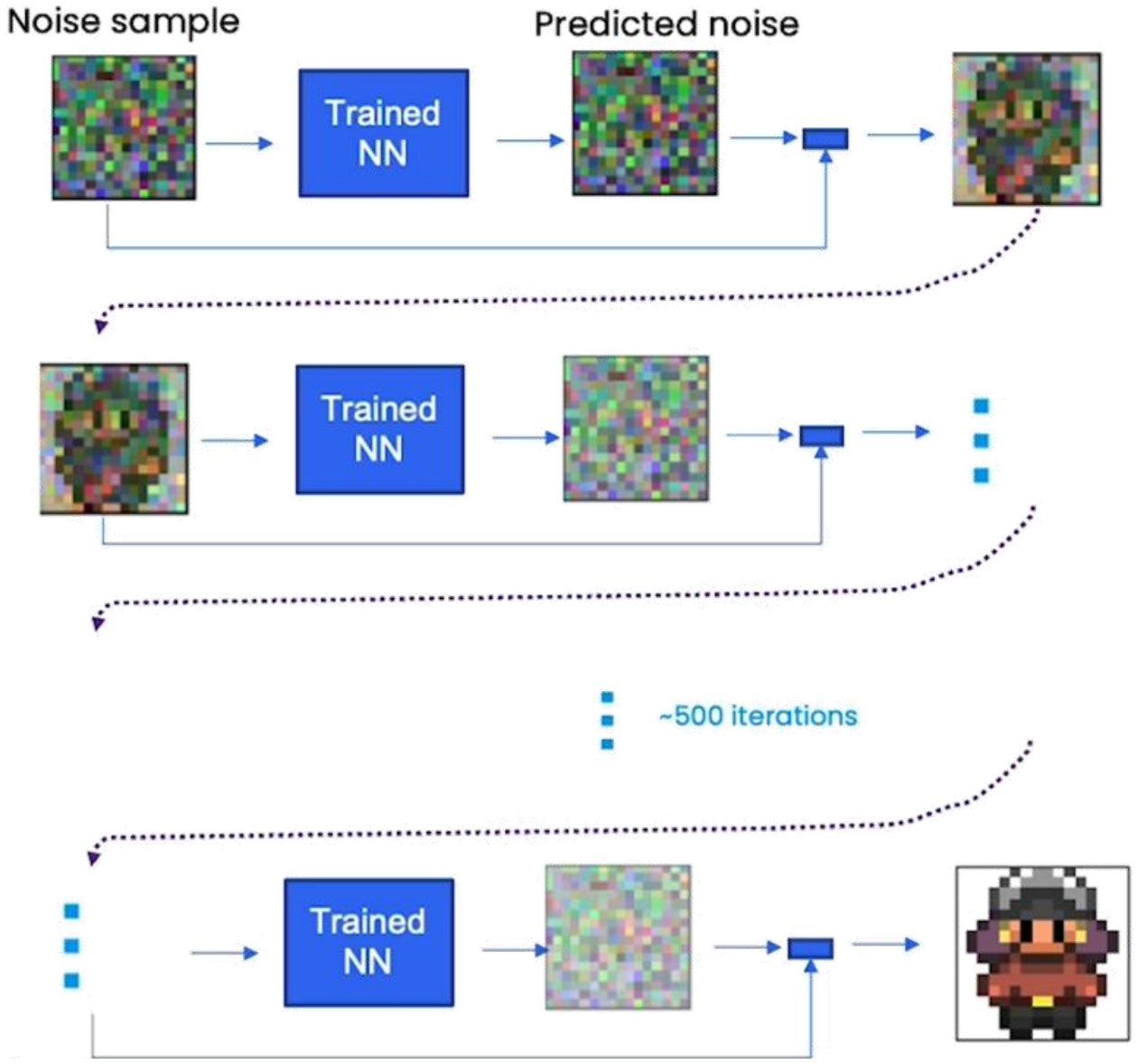
https://shorturl.at/bkJ49

7.

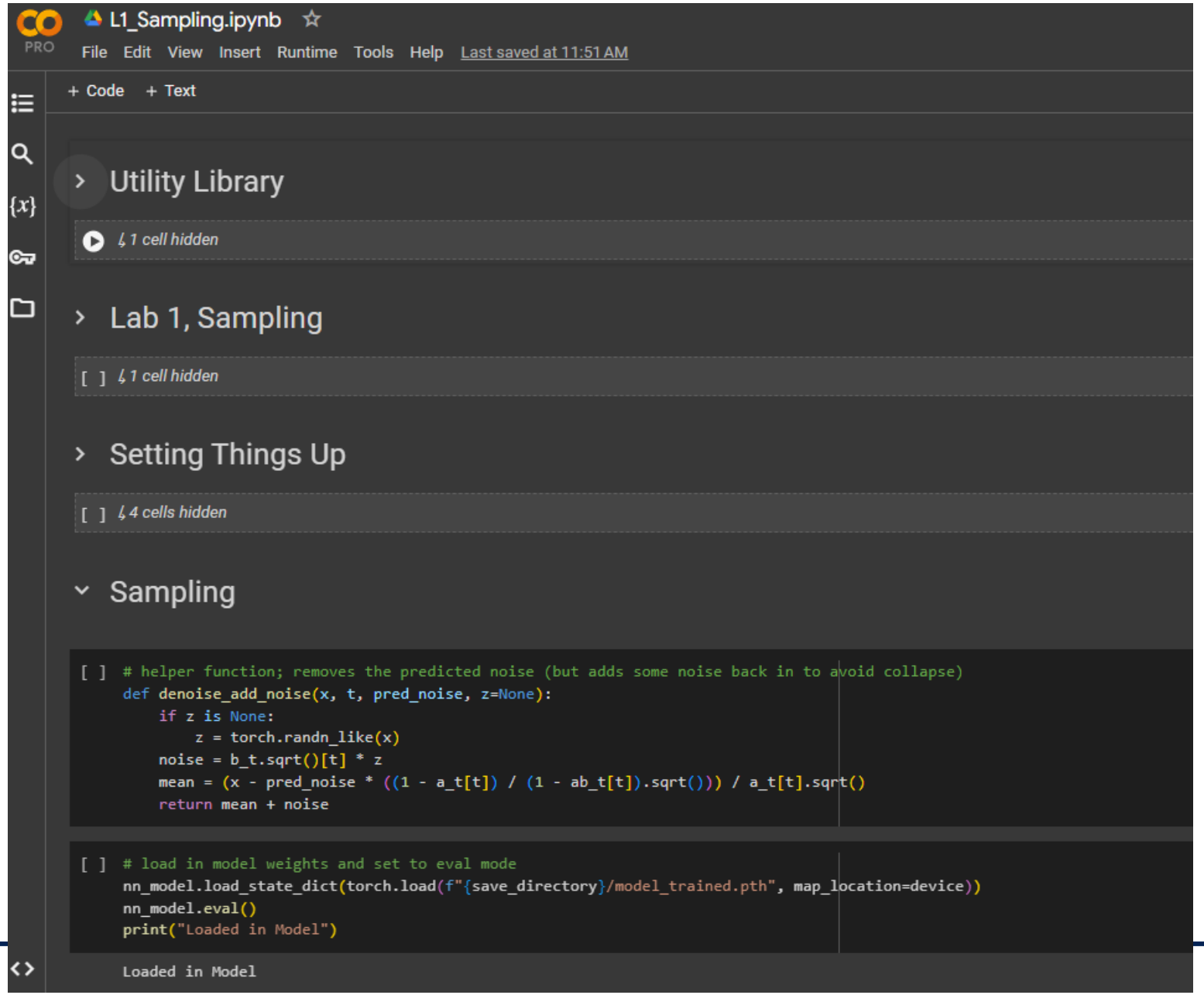
Lab 1: Uso Básico de Modelos de Difusión

Muestreo

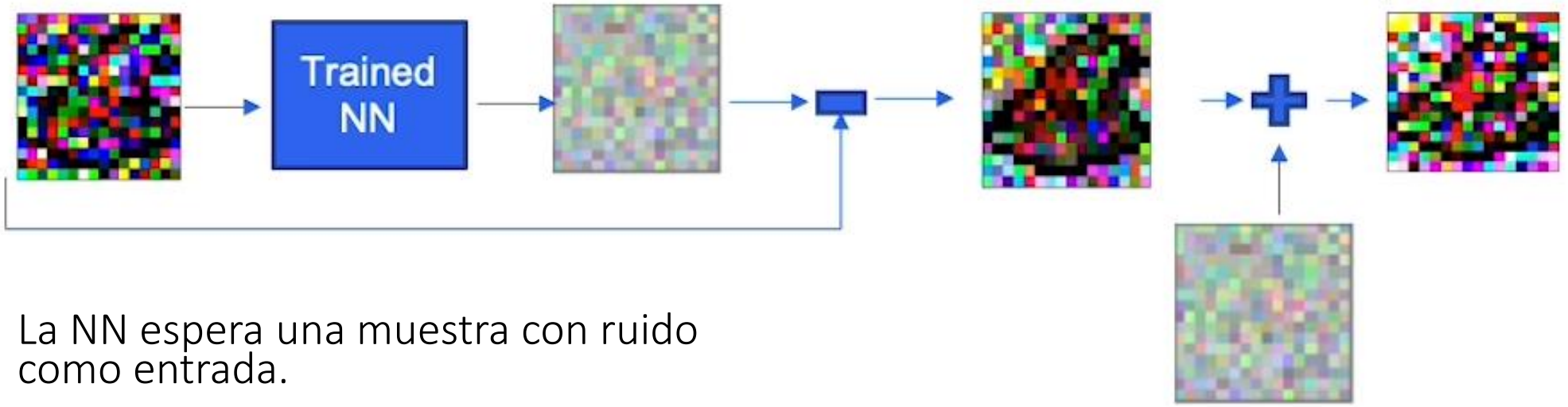
La red neuronal intenta predecir completamente el ruido en cada paso. Realísticamente, es solo una predicción. Necesitas múltiples pasos para obtener sprites de alta calidad.



```
# sample using standard DDPM algorithm
@torch.no_grad()
def sample_ddpm(n_sample, device, save_rate=20):
    # x_T ~ N(0, 1), sample initial noise
    samples = torch.randn(n_sample, 3, height, height).to(device)
    ...
    for i in range(timesteps, 0, -1):

        # reshape time tensor
        t = torch.tensor([i / timesteps])[:, None, None, None].to(device)

        # sample some random noise to inject back in. For i = 1, don't add
back in noise
        z = torch.randn_like(samples) if i > 1 else 0

        eps = nn_model(samples, t)    # predict noise e_(x_t,t)
        samples = denoise_add_noise(samples, i, eps, z)
        ...
```


L1_Sampling.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:51 AM
+ Code + Text
Utility Library
1 cell hidden
Lab 1, Sampling
1 cell hidden
Setting Things Up
4 cells hidden
Sampling
# helper function; removes the predicted noise (but adds some noise back in to avoid collapse)
def denoise_add_noise(x, t, pred_noise, z=None):
if z is None:
z = torch.randn_like(x)
noise = b_t.sqrt()[t] * z
mean = (x - pred_noise * ((1 - a_t[t]) / (1 - ab_t[t]).sqrt())) / a_t[t].sqrt()
return mean + noise
# load in model weights and set to eval mode
nn_model.load_state_dict(torch.load(f"{save_directory}/model_trained.pth", map_location=device))
nn_model.eval()
print("Loaded in Model")
Loaded in Model

Detalles de la Iteración de Muestreo

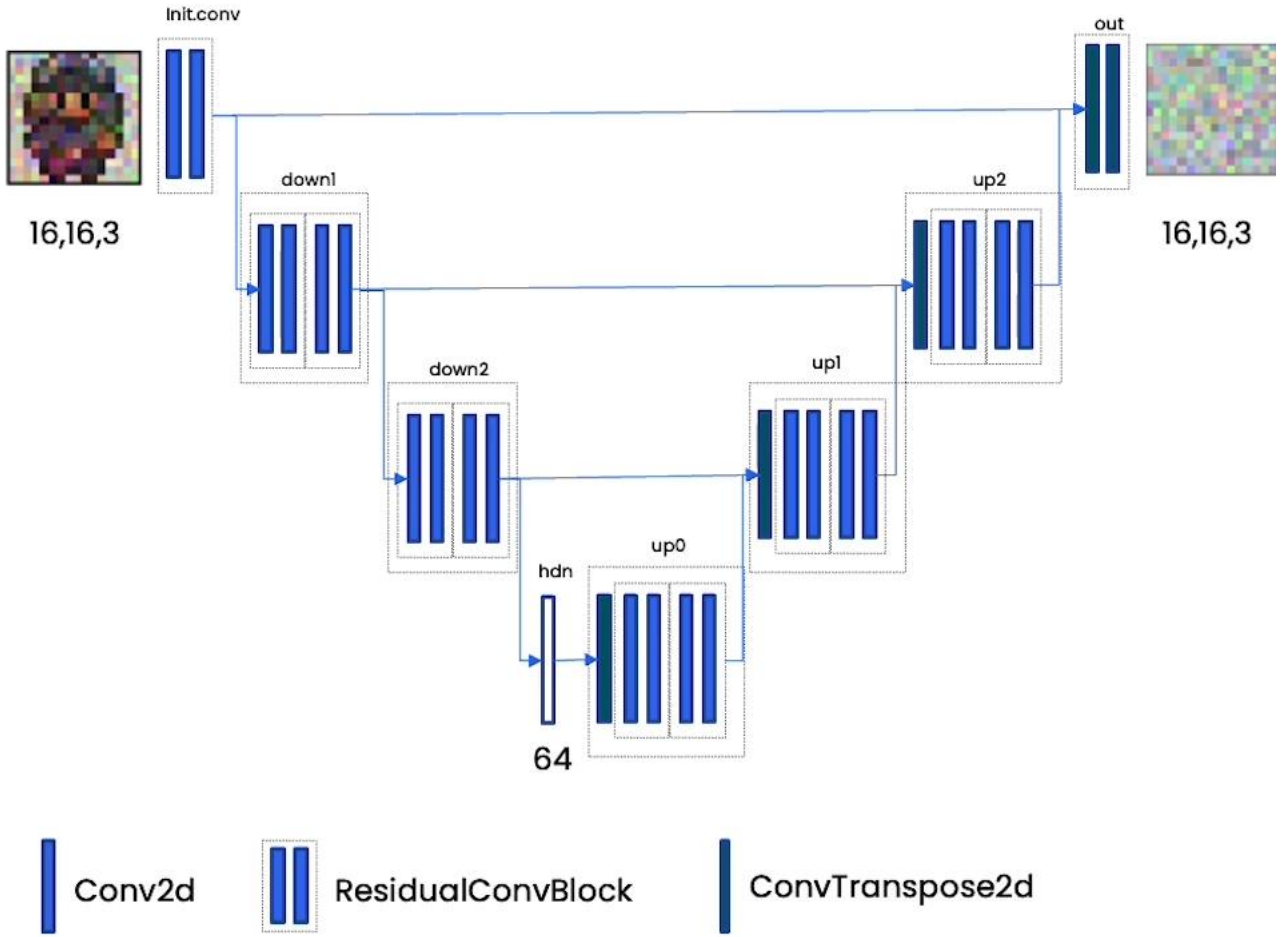


- La NN espera una muestra con ruido como entrada.
- Puedes añadir ruido adicional antes de que se pase al siguiente paso.
- Empíricamente, esto estabiliza la NN para que no colapse a algo más cercano al promedio del conjunto de datos.

2.

Lab 2: Entrenamiento

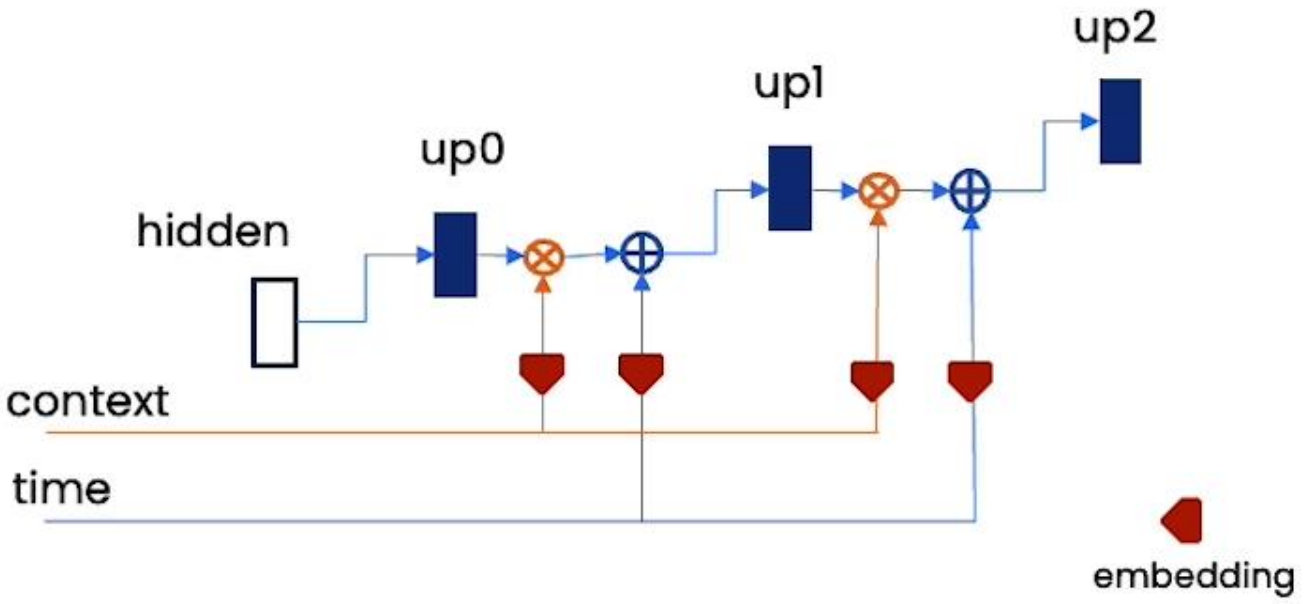
U-Net para identificar el ruido



Incorporación de Más Información

El U-Net puede incorporar más información en la forma de incrustaciones:

- Incrustación de tiempo: relacionada con el paso de tiempo y el nivel de ruido.
- Incrustación de contexto: relacionada con el control de la generación, por ejemplo, descripción de texto o factor (más adelante).

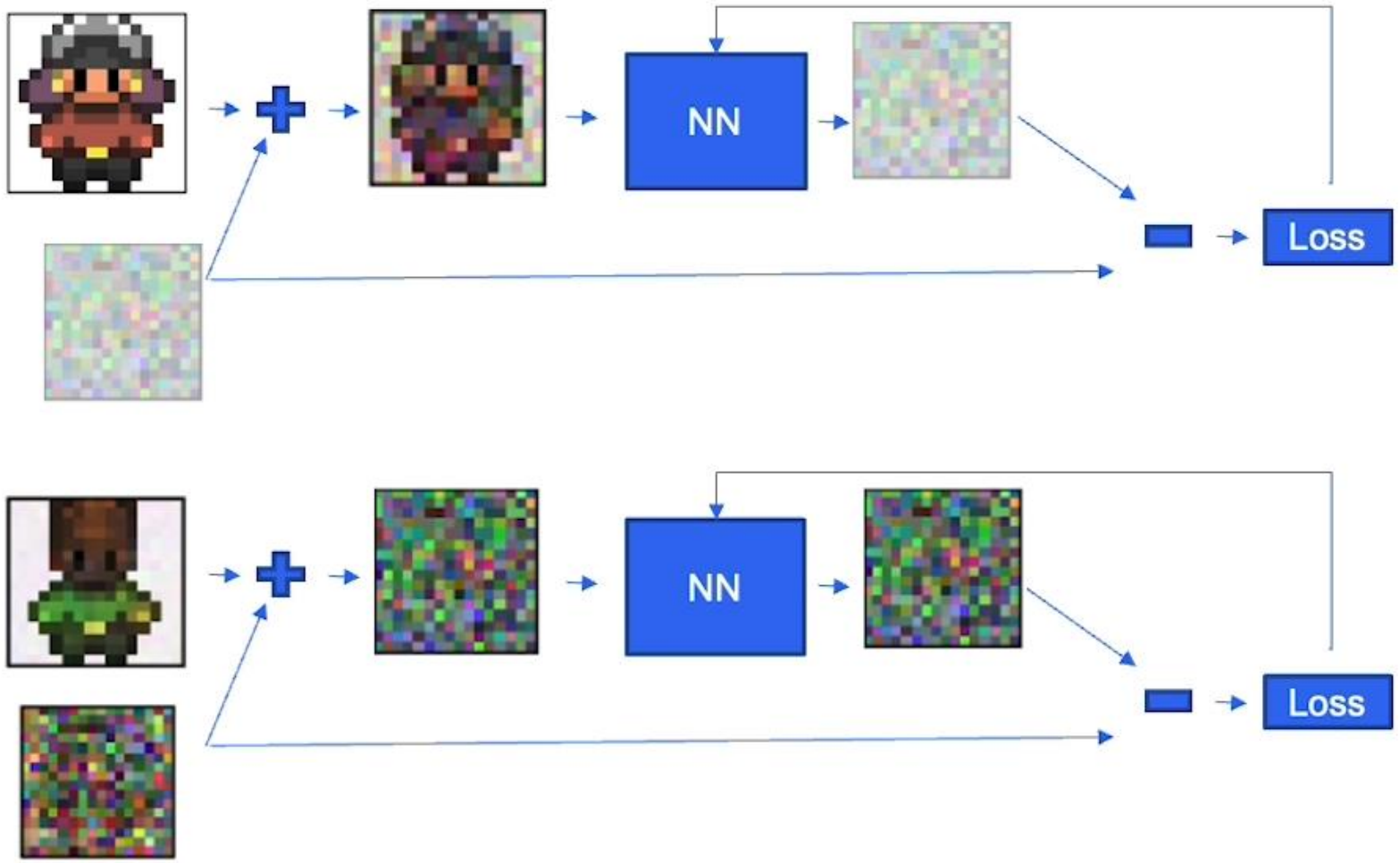



```
# embed context and timestep
cemb1 = self.contextembed1(c).view(-1, self.n_feat * 2, 1, 1)
temb1 = self.timeembed1(t).view(-1, self.n_feat * 2, 1, 1)

up2 = self.up1(cemb1*up1 + temb1, down2)
```

Entrenamiento

La NN aprende a predecir el ruido, lo que realmente significa aprender la distribución de lo que no es ruido.

Se toma una muestra aleatoria del paso de tiempo (nivel de ruido) por imagen para entrenar más estabilizadamente.



Real image

Input

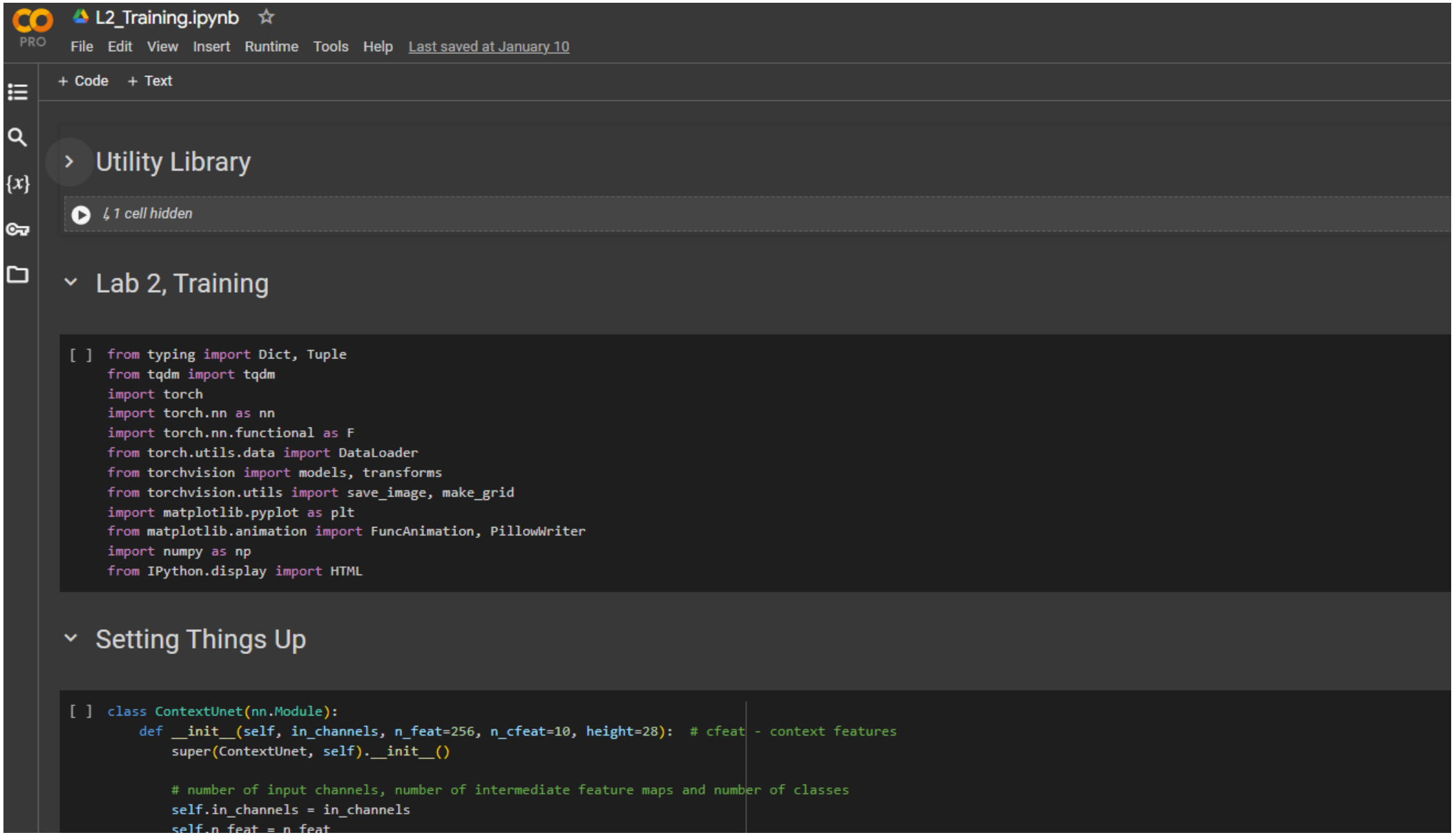
Epoch 0

Epoch 31

El Algoritmo

- Tomar una imagen de entrenamiento.
- Tomar una muestra del paso de tiempo t. Esto determina el nivel de ruido.
- Tomar una muestra del ruido.
- Añadir ruido a la imagen.
- Introducir esto en la red neuronal. La red neuronal predice el ruido.
- Calcular la pérdida entre el ruido predicho y el ruido verdadero.
- Retropropagación y aprendizaje.

```
for ep in range(n_epoch):

    # linearly decay learning rate
    optim.param_groups[0]['lr'] = lrate*(1-ep/n_epoch)

    pbar = tqdm(dataloader, mininterval=2 )
    for x, _ in pbar:   # x: images
        optim.zero_grad()

        # perturb data
        t = torch.randint(1, timesteps + 1, (x.shape[0],)).to(device)
        noise = torch.randn_like(x)
        x_pert = perturb_input(x, t, noise)

        # use network to recover noise
        pred_noise = nn_model(x_pert, t / timesteps)

        # loss is mean squared error between the predicted and true noise
        loss = F.mse_loss(pred_noise, noise)
        loss.backward()
        optim.step()
```


L2_Training.ipynb
File Edit View Insert Runtime Tools Help Last saved at January 10
+ Code + Text
Utility Library
1 cell hidden
Lab 2, Training
[ ] from typing import Dict, Tuple
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import models, transforms
from torchvision.utils import save_image, make_grid
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
import numpy as np
from IPython.display import HTML
Setting Things Up
[ ] class ContextUnet(nn.Module):
def __init__(self, in_channels, n_feat=256, n_cfeat=10, height=28): # cfeat - context features
super(ContextUnet, self).__init__()
# number of input channels, number of intermediate feature maps and number of classes
self.in_channels = in_channels
self.n_feat = n_feat

3.

Lab 3: Control sobre el Modelo

Embeddings o Incrustaciones

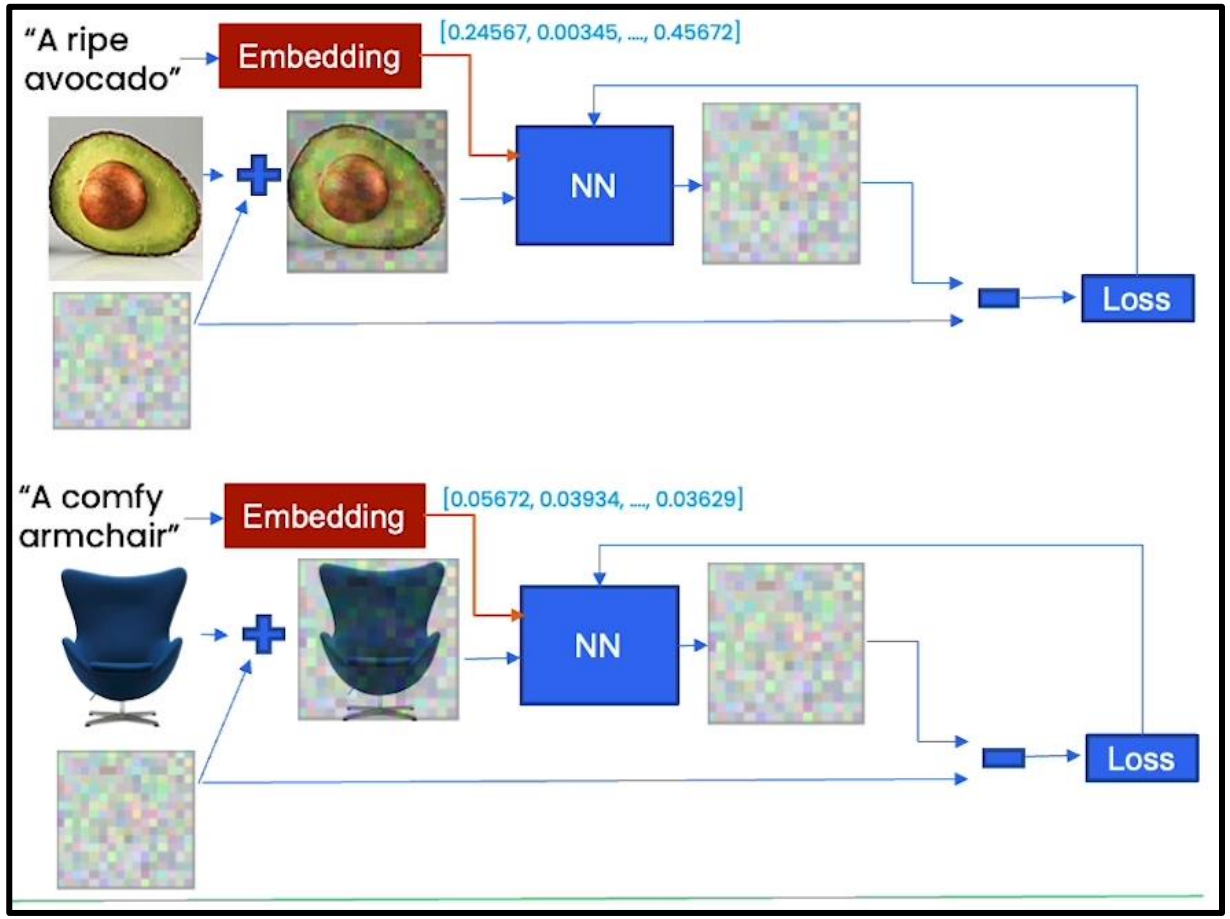
Los vectores de incrustación capturan el significado.

Textos con contenido similar tendrán vectores similares.

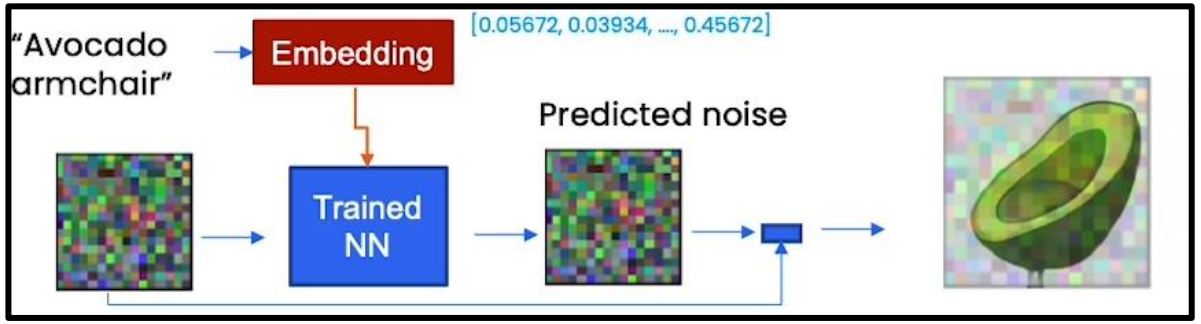
	Paris	[-0.003530, -0.010379, ..., 0.005863]
−	France	[-0.058398, -0.063971, ..., 0.007234]
+	England	[-0.039857, -0.023971, ..., 0.038608]
=	London	[0.0150108, 0.045863, ..., 0.037236]

Sumando Contextos

Entrenar

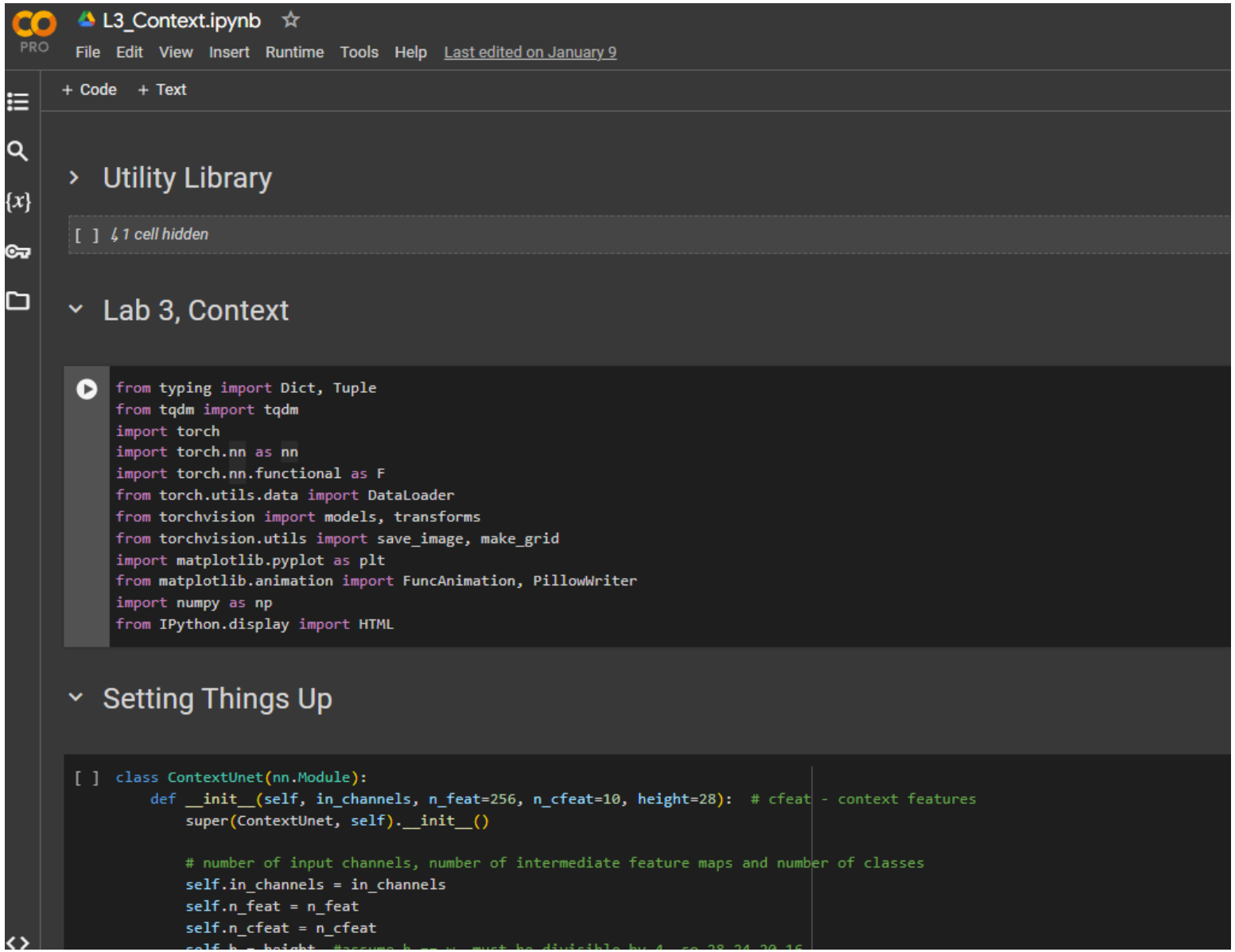


Muestrear



Contexto

- El contexto es un vector que controla la generación.
- El contexto puede ser incrustaciones de texto, por ejemplo, de más de 1000 de longitud.
- El contexto también puede ser categorías, por ejemplo, de 5 de longitud.


L3_Context.ipynb
File Edit View Insert Runtime Tools Help Last edited on January 9
+ Code + Text
Utility Library
[ ] 1 cell hidden
Lab 3, Context
from typing import Dict, Tuple
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import models, transforms
from torchvision.utils import save_image, make_grid
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
import numpy as np
from IPython.display import HTML
Setting Things Up
[ ] class ContextUnet(nn.Module):
def __init__(self, in_channels, n_feat=256, n_cfeat=10, height=28): # cfeat - context features
super(ContextUnet, self).__init__()
# number of input channels, number of intermediate feature maps and number of classes
self.in_channels = in_channels
self.n_feat = n_feat
self.n_cfeat = n_cfeat

4.

Lab 4: Mejorando Eficiencia

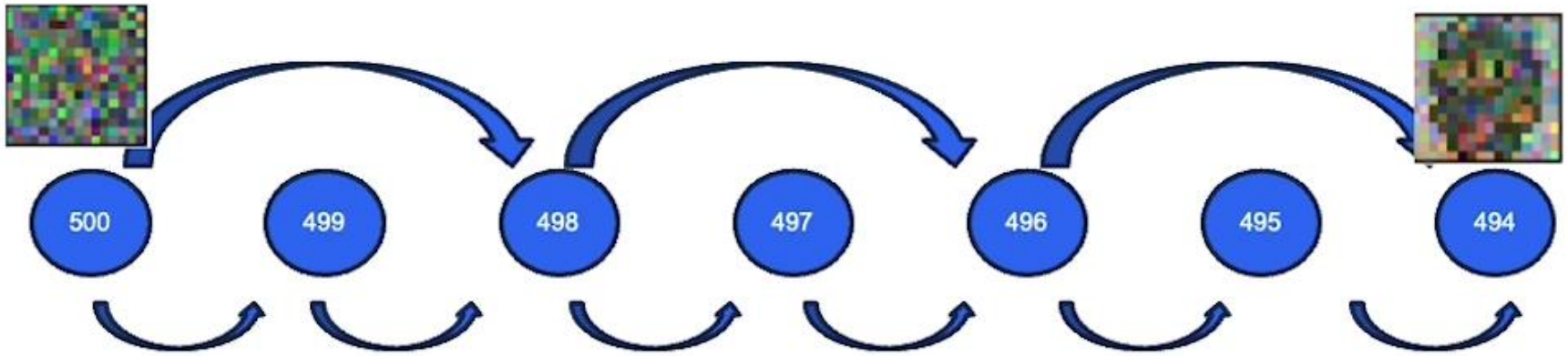
El Muestreo es Lento

- Muestreo es lento
- Quieres más imágenes rápido.

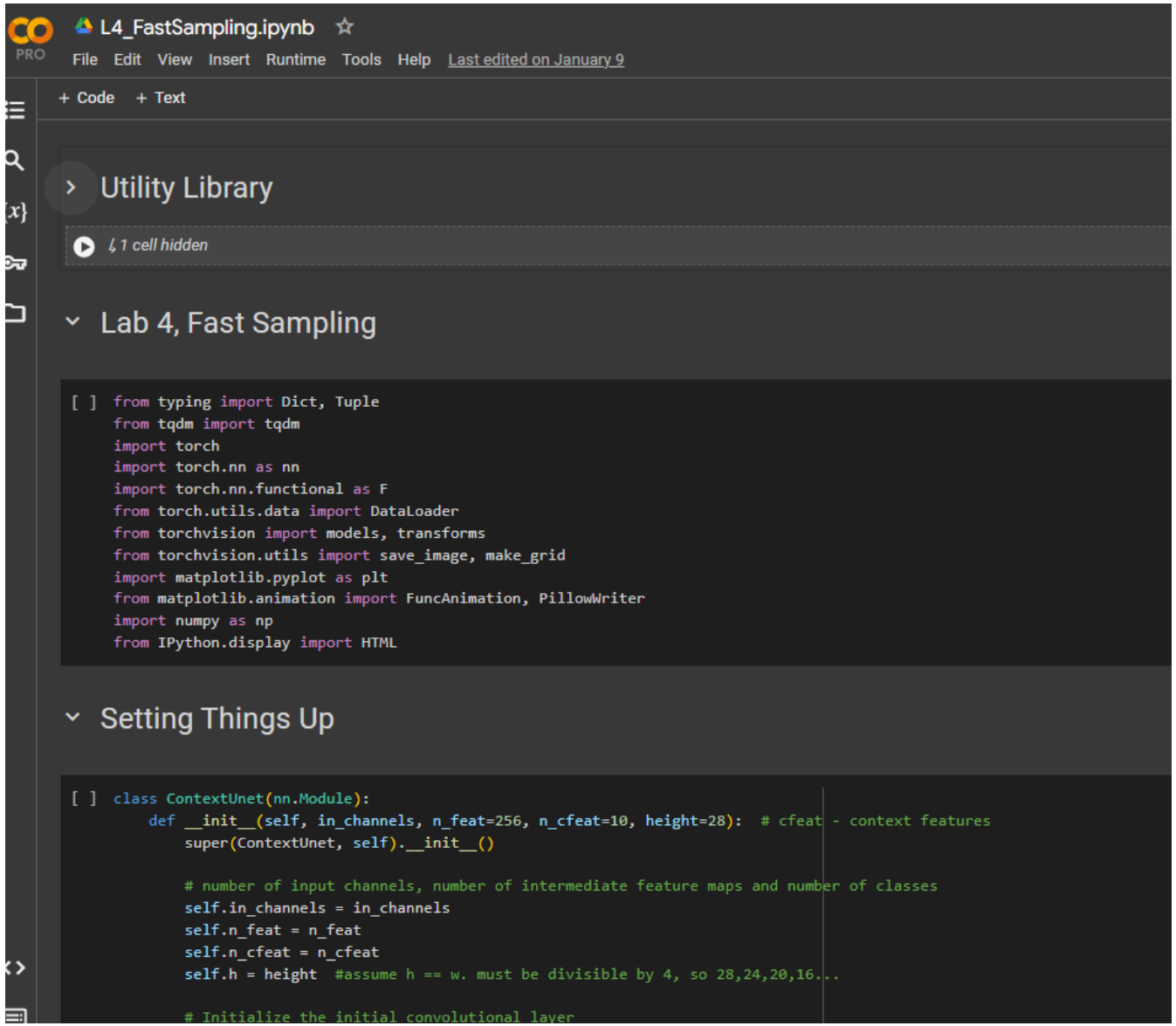
Pero el muestreo es lento porque:

- Hay muchos pasos de tiempo.
- Cada paso de tiempo depende del anterior (Markoviano).
- Muchos nuevos muestreadores abordan este problema de velocidad.

- Uno se llama DDIM: Modelos Implícitos de Difusión de Denoising (¡solo el nombre del artículo!)



DDIM es más rápido porque omite pasos de tiempo.


L4_FastSampling.ipynb
File Edit View Insert Runtime Tools Help Last edited on January 9
+ Code + Text
Utility Library
1 cell hidden
Lab 4, Fast Sampling
[ ] from typing import Dict, Tuple
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import models, transforms
from torchvision.utils import save_image, make_grid
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
import numpy as np
from IPython.display import HTML
Setting Things Up
[ ] class ContextUnet(nn.Module):
def __init__(self, in_channels, n_feat=256, n_cfeat=10, height=28): # cfeat - context features
super(ContextUnet, self).__init__()
# number of input channels, number of intermediate feature maps and number of classes
self.in_channels = in_channels
self.n_feat = n_feat
self.n_cfeat = n_cfeat
self.h = height #assume h == w. must be divisible by 4, so 28,24,20,16...
# Initialize the initial convolutional layer

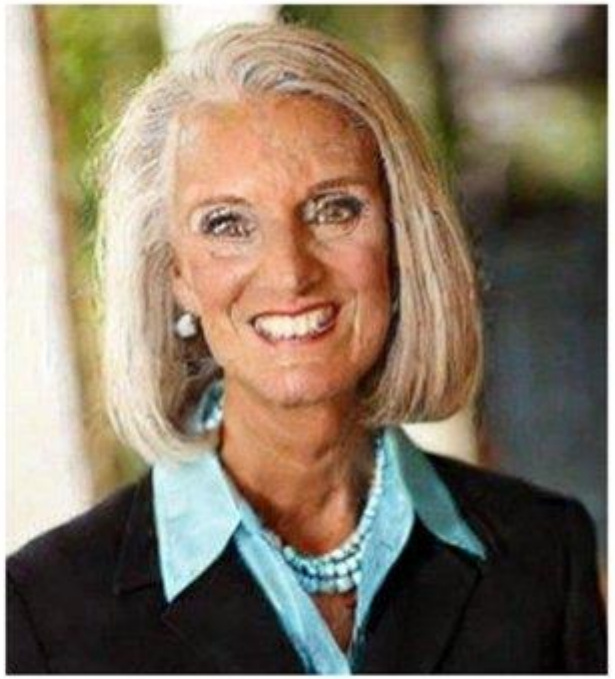
Recomendaciones Generales

Problemas a considerar

Training Set

Caption: Living in the light with Ann Graham Lotz

Generated Image



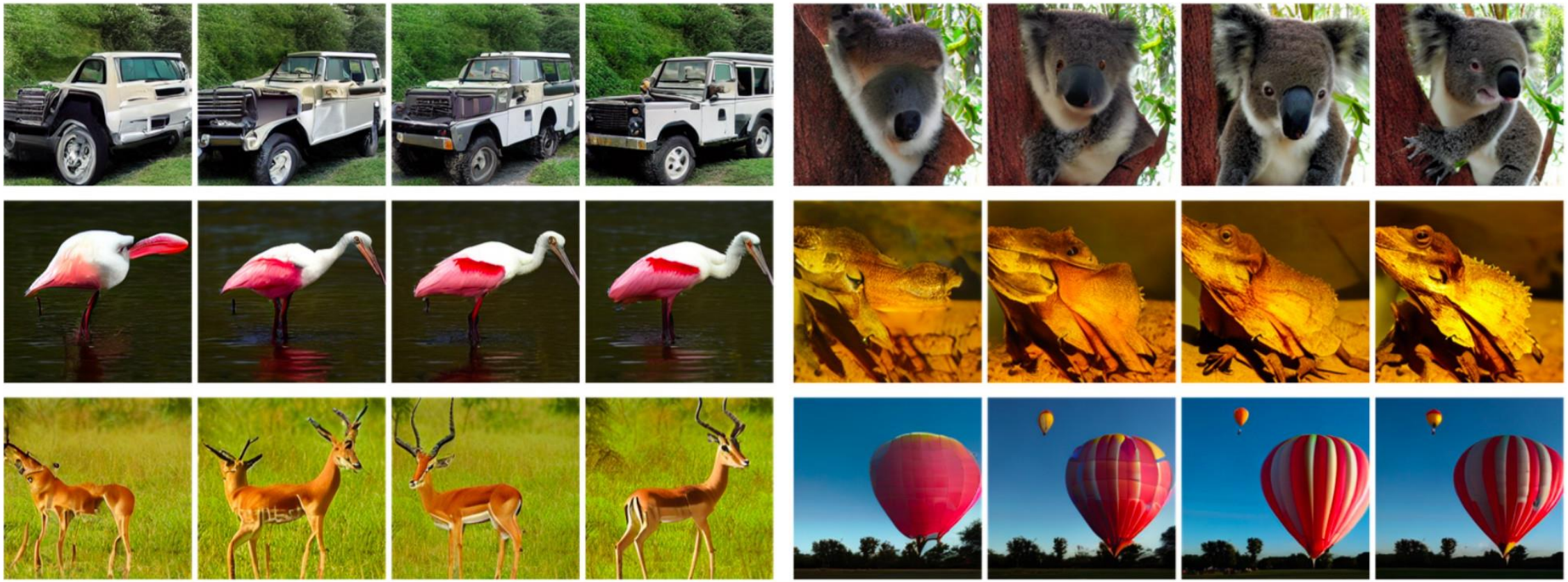
Prompt: Ann Graham Lotz

Consideraciones

- Muchas posibilidades de errores en la codificación.
- Utiliza un Modelo de Media Móvil Exponencial.
- Ten cuidado con los tiempos "cerca de los datos", usa un mínimo/máximo numérico.
- Los algoritmos de muestreo pueden cambiar mucho las cosas para un modelo fijo.
- Los modelos de difusión son una oportunidad de tener resultados importantes con modelos no tan grandes.

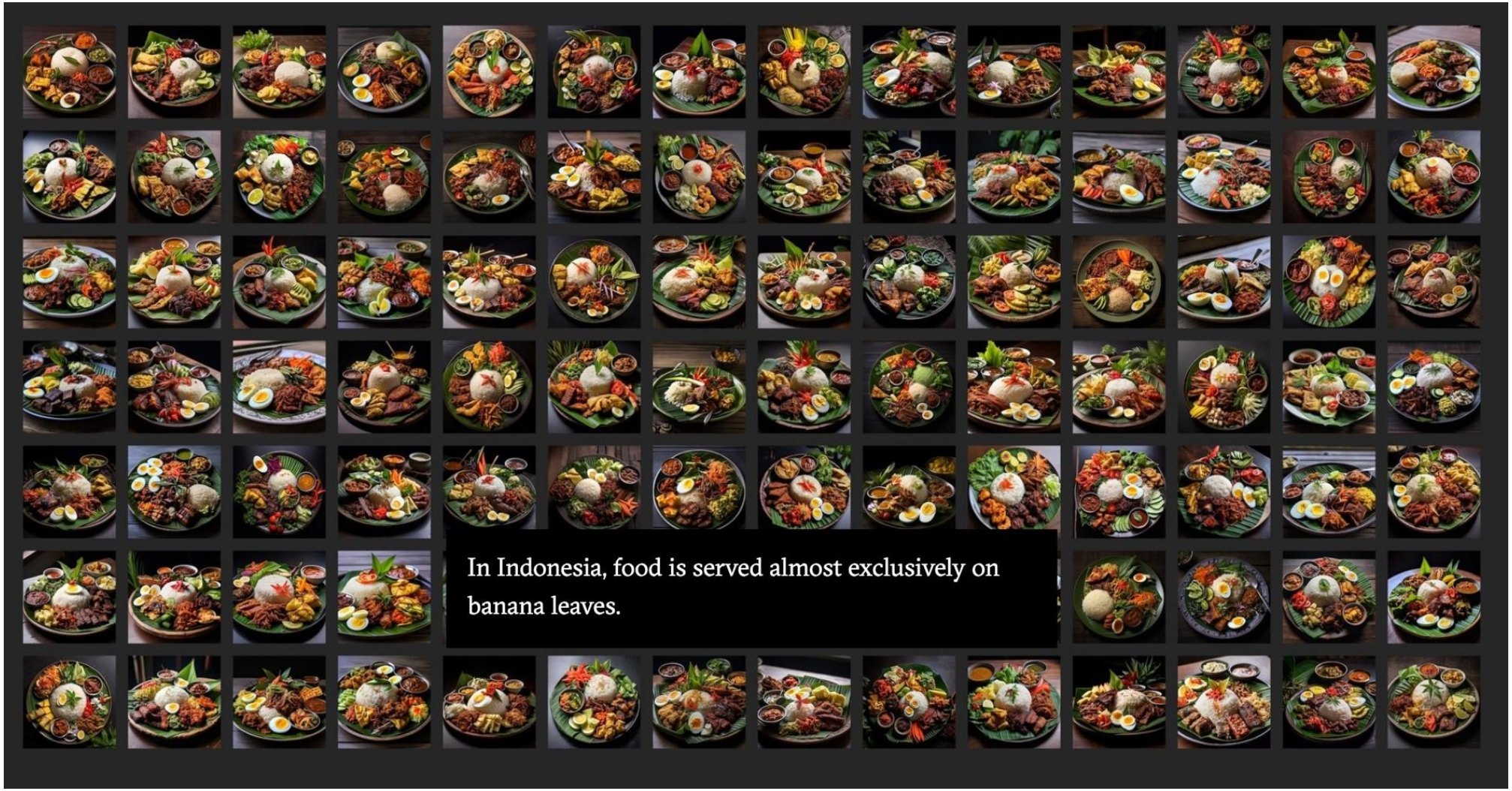
Consideraciones

Tamaño del modelo →



Consideraciones

Consideraciones



¡Modelos de difusión desbloqueados!

- Durante la creación de esta presentación, ninguna persona ha sido reemplazada.
- Invito a todos a participar y explorar juntos cualquier duda o curiosidad o contactarme por:

a20234215@pucp.edu.pe
Rodrigo Alonso Uribe Ventura

55 AÑOS

PUCP
Departamento Académico de Ingeniería

IA PUCP

Winter Camp

Unidos creando un legado