Sporender: Analyzing Latent Space Patterns in Diffusion Models using Mushroom Images

Mg Beastrom · Follow

12 min read · Dec 3, 2024

By Zoeexelbert, Ella Visconti, and Mg Beastrom

Link to our code repository here

Motivation

Recently, there has been a lot of excitement and talk around General Adversarial Networks (GANs) and Diffusion Models in the machine learning world. With growing advances to generate and re-create data from a given input data set, this new field of image generation allows researchers and developers to create highly realistic synthetic images, pushing the boundaries of creativity and automation in the technological sphere.

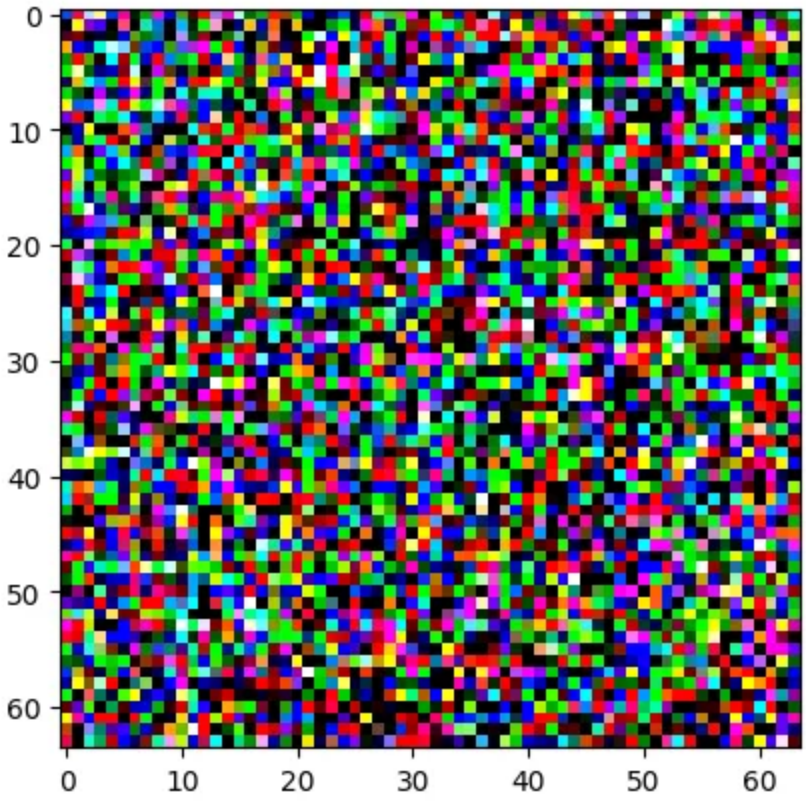
We wanted to explore the power of these sorts of models in generating mushroom images, a dataset with an extremely diverse range of visual

features. From different gill structures, spore patterns, cap and stem sizes, and various forest environments, mushrooms present an interesting dataset for diffusion models to train on.

What is a Diffusion Model?

In a nutshell, a diffusion model adds random noise to an image over a series of steps. Then, the neural network is used to denoise the sample image and return it to its original form.

Diffusion process: We sample a random data point from the real image and then add Gaussian noise to create a new latent variable. As timestamp T (where T represents time) becomes larger and larger as it nears $T = \infty$ the image loses more and more of its distinguishing features leaving it to look something like the image below.



Reverse Diffusion Process: To conduct the reverse diffusion process in which the model denoises the images to return to the original sample images we need to approximate $q(\mathbf{x}t-1|\mathbf{x}t)$ which is done by using the the p0 model that when for a chosen p0 that is small enough it will be Gaussian thus allowing us to isolate and find the mean and variance parameters for each timestep T. The algorithm for the model p0 is defined below.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

This allows for the reversing of the denoising process on mushrooms to subsequently create images that should look closer and closer to its true representation.

GAN vs. Diffusion

Given minimal computational power and time to complete this experiment, we chose to test our experiment on a Diffusion Model, as it provides us with a more stable, less prone to collapse environment. Since these models are more catered toward tasks that require detailed image synthesis, this choice helped curb the possibility of producing limited diversity in image outputs and avoiding the Vanishing Gradient Problem. While the process is slower due to the iterative denoising process and more complex architecture, this model explicitly learns to handle and remove noise. Thus, this model choice allows us to better investigate how the model navigates and learns the structure of the latent space, giving us deeper insights into the underlying patterns being learned that are not directly observable in the input space.

Data

For our data, we used a publicly available Kaggle dataset that consisted of 16000 mushroom images, which can be found here. Building inspiration from Keras implementation of a Diffusion model, we followed this approach with our own custom dataset that stored all of our Kaggle data.

To accomplish this, we wrote a Python script to build a custom TensorFlow dataset in order to comply with Keras code structure. Following TensorFlow's guide to writing custom datasets, we first imported our mushroom data and created a local directory to store our data.

```
!ls /content/drive/My\ Drive/Mushrooms  # Update with your folder name
mushroom_data = '/content/drive/MyDrive/Mushrooms'
!mkdir ~/tfds_mushrooms
!cd ~/tfds_mushrooms
!tfds new my_mushrooms
```

After accessing this directory and creating tfds template files for our dataset 'my_mushrooms', our script built the dataset structure we needed for training.

```
= "tensorflow"

plt

s tfds


 import train_test_split


GeneratorBasedBuilder):
```

```
_mushrooms dataset."""


ion('1.0.0')

lease.',




et metadata."""
tent/drive/MyDrive/Mushrooms'
(os.listdir(dataset_path))  # Get class names from folder names
asetInfo(

ataset of mushroom images organized into species.",
atures.FeaturesDict({
s.features.Image(shape=(256, 256, 3)),  # Fixed size
s.features.ClassLabel(names=class_names),

('image', 'label'),
//example.com',

r_dataset,
hrooms Dataset},
}




lf, dl_manager):
rators."""
tent/drive/MyDrive/Mushrooms'


all image paths and their corresponding labels

dir(dataset_path):
path.join(dataset_path, label)
sdir(label_path):

os.listdir(label_path):
s.path.join(label_path, img_file)
lower().endswith('.jpg') or img_path.lower().endswith('.jpeg'):
.append((img_path, label))


o 75% training and 25% validation
ain_test_split(examples, test_size=0.2, random_state=42, stratify=[x[1] for x in exa
```

```
est_split(train_val, test_size=0.25, random_state=42, stratify=[x[1] for x in train_



enerate_examples(train),
lf._generate_examples(val),



elf, examples):
rom a list of file paths and labels."""
in examples:


{
_path,  # Tensor representation of image
el,  # Label (species) for each image



set builder

ataset
()

ataset(split='train', as_supervised=True)
aset(split='validation', as_supervised=True)
```

By creating a custom DatasetBuilder class in this script, we comply with the Tensorflow dataset structure, allowing us to successfully build our mushroom data. Here, we generate our data splits as well, dividing our dataset into 75% training data and 25% validation data. At this point, we have a dataset that aligns with Keras Diffusion Model, replacing their use of the Tensorflow flower dataset with ours.

```
train_dataset = tfds.load('my_mushrooms', split='train')
val_dataset = tfds.load('my_mushrooms', split='validation')
```

We can now load in our data using the tfds.load() function, as we have a fully functioning Tensorflow dataset at this point. This puts us in position to run the preprocess_image() and prepare_dataset() functions Keras has written for us, which properly processes all of our mushroom images and shuffles our training and validation splits. It is important here to change our 'dataset_name' to the name of our dataset, 'my_mushrooms', and then load in our split datasets to be shuffled like so:

```
train_dataset = prepare_dataset("train")
val_dataset = prepare_dataset("validation")
```

Model

This model utilizes Kernal Inception Distance (KID) which is a quality metric that measures the similarity between the images generated by the model and the real images in the latent space. This is essentially just ensuring that the images generated are realistic and diverse.

The architecture of the model is a U-Net which is a convolutional neural network that progressively downsamples and upsamples the input image. It receives two inputs; the input image (noisy image) and the noise levels associated with that image. Then it puts the image through a sequence of downsamplings until the image is at its lowest resolution when processes the image's features at it's lowest resolution, learns the complex transformations to extract high-level features necessary for reconstructing the output during upsampling, and preserves the identity mapping via residual connections. Then it reconstructs the high-resolution image through upsampling which involves upscaling the features using bilinear interpolation. Finally, it

outputs a 3-channel output with the final convolution that matches the input image.

One key feature to note about U-Net that is crucial for the diffusion model is that it skips connections between layers of the same resolution which allows it to pass the information directory from the downsampling path to the upsampling path, meaning that it does not lose detailed information and features of the images. Thus the reconstruction can create fine-grain accurate features and details in the images.

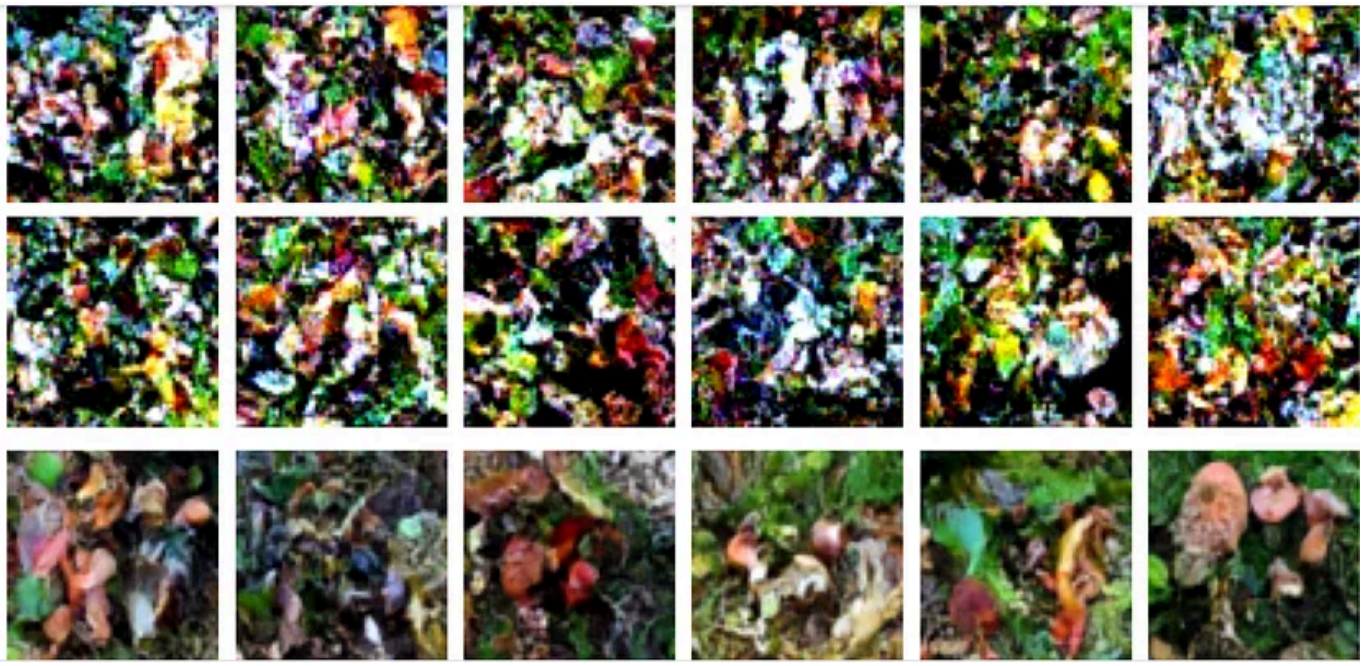
The training of the model involves first sampling random diffusion times from the diffusion schedule (read more in Keras Documentation) and combining training images with random Gaussian noise. Then the model trains by trying to separate the image from the Gaussian noise. The key methods of this portion of the code are

1. diffusion_schedule: converts diffusion times into signal rates and noise rates
2. denoise: predicts the noise rate and reconstructed image
3. reverse_diffusion: generates images by reversing the diffusion process by starting at random noise noise and iteratively denoising by computing the noise rates for the current step, predicting the noise, and reconstructing the image
4. generate: calls these other methods to produce images from the Gaussian noise and denormalize the output image for visualization

Finally, the trained model can be used to generate new images by starting from noise and implementing reverse diffusion. Thus giving us results in generating mushrooms that you can find below.

Training

After all of this implementation preparation, our data is normalized and we can run our training process, periodically plotting our generated images to track model performance.



Examples of the images generated by the model while training

These images help us to see how the model is progressing in generation from the initial denoising. From the images we can see the progression of features that it picks up on to create realistic-looking mushrooms.

So now, after training, we can begin to investigate exactly what the model is picking up on that is allowing it to recreate the mushrooms in our dataset. While still somewhat grainy, we still get mostly discernible image generations with the computation units we had access to.

100 of our generated images after training is complete

Results

After training, we wanted to investigate our model and somehow take a step into the “brain” of the neural network. In order to do this we analyzed and manipulated data from the latent space of the generated images. From this we were hoping to see that certain configurations of noise in the initial generated images resulted in discernible patterns in the clear/denoised

versions of the images. We expected that the latent space of initial noise generations would capture features such as the quantity of mushrooms, class/species of mushrooms, the color, or the size. We hoped that with a big enough sample size of generated mushrooms we could observe these features in clusters. From there, we wanted to discover a way to change a feature of the image(i.e. color, cap size, number of mushrooms) by adding or subtracting a vector to an original noisy image. All experiments are performed on the initial noisy images generated by the model but are presented in their denoised version to allow for visual observation.

We began by hand selecting images for clusters based on observation of the generated set of images. For this experiment we *foraged* for 10 of the "red-est" and 10 of the "yellow-est" mushrooms.

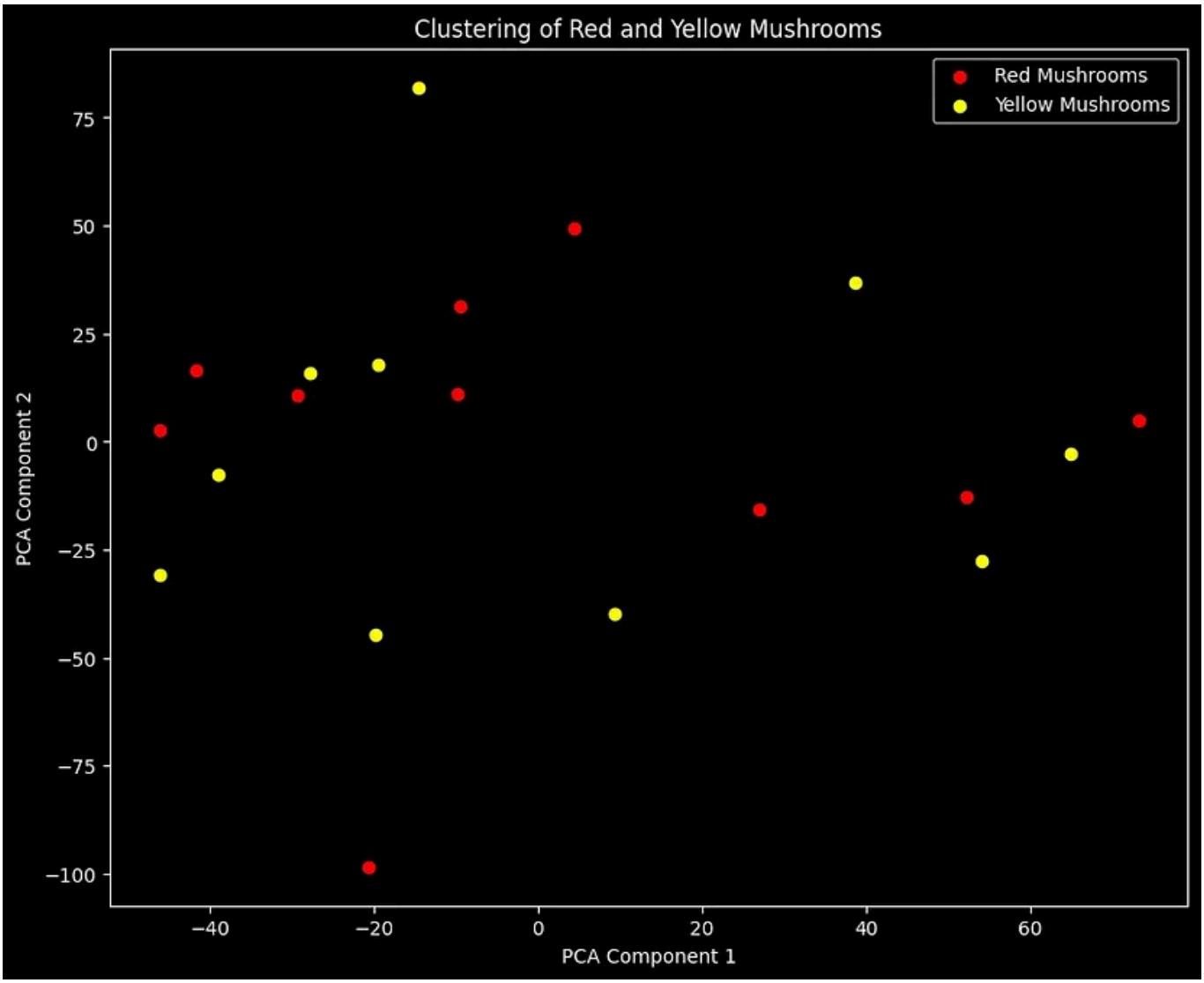
Hand picked red mushrooms

Hand picked yellow mushrooms

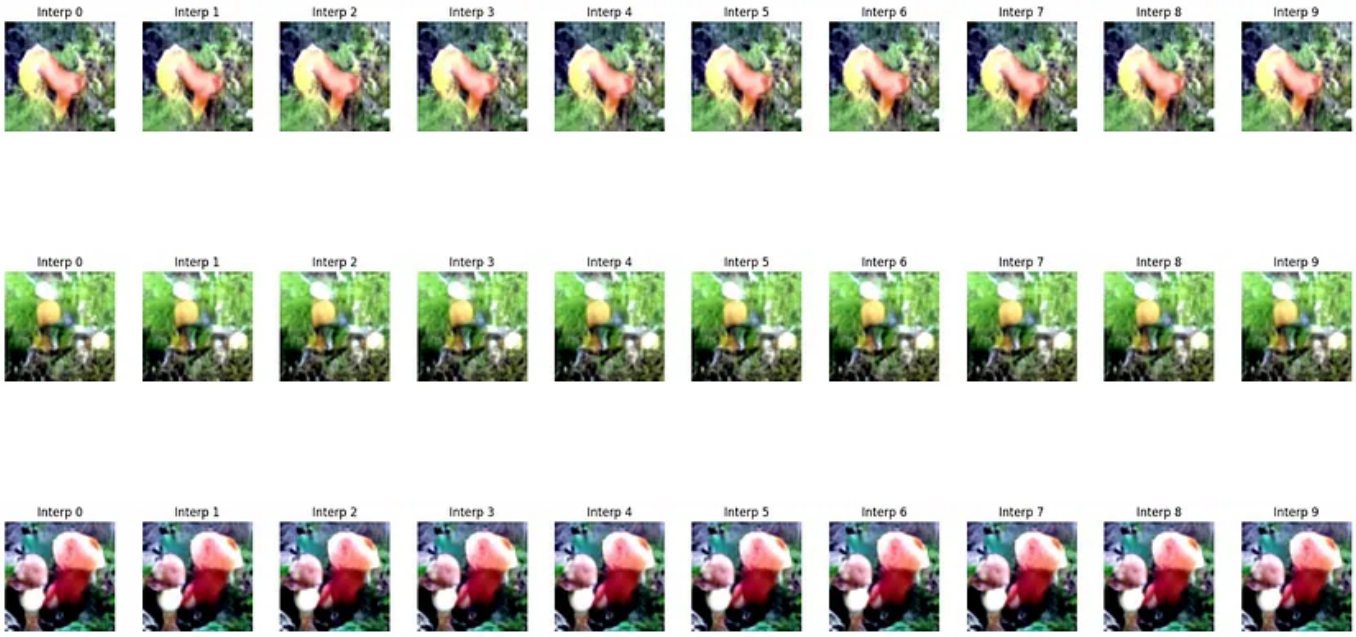
From each cluster we can grab the center vector, and use this as an anchor that represents a pattern within our clusters of mushrooms. After hand clustering, we calculated the distance between each of the two cluster centers. We hypothesized that these clusters being farther from each other on the RGB spectrum would give us a large enough vector to use to manipulate the color of other mushrooms in our generated set.

```
Yellow Cluster Center to Red Cluster Center Distance Vector:
[-5.5879363e-09 -4.7683717e-08  3.5762788e-08 ... -3.5762788e-08
 -2.3841858e-08 -2.9802322e-08]
```

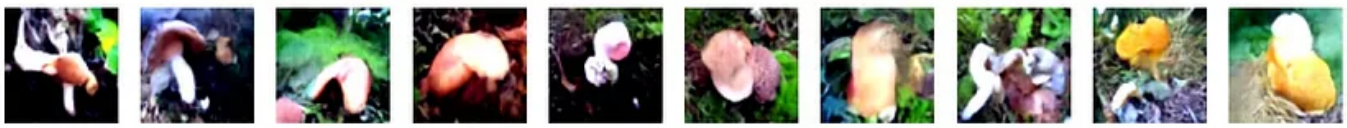
This number was a lot smaller than we were expecting, meaning that the generated mushrooms likely were NOT grouped by color within the initial noisy latent space. Plotting the two handpicked clusters in a reduced dimension space confirmed their proximity in the space as well.



We did continue the experiment and printed out the interpolation of a mushroom before and after adding the distance value to see if the images became more red or yellow. There was no observable differences in color within these trials. We came to the conclusion that color is not a defining feature of the denoising process, or at the very least is not something determined from the early stages of denoising. The results of the red-yellow interpolations are below.



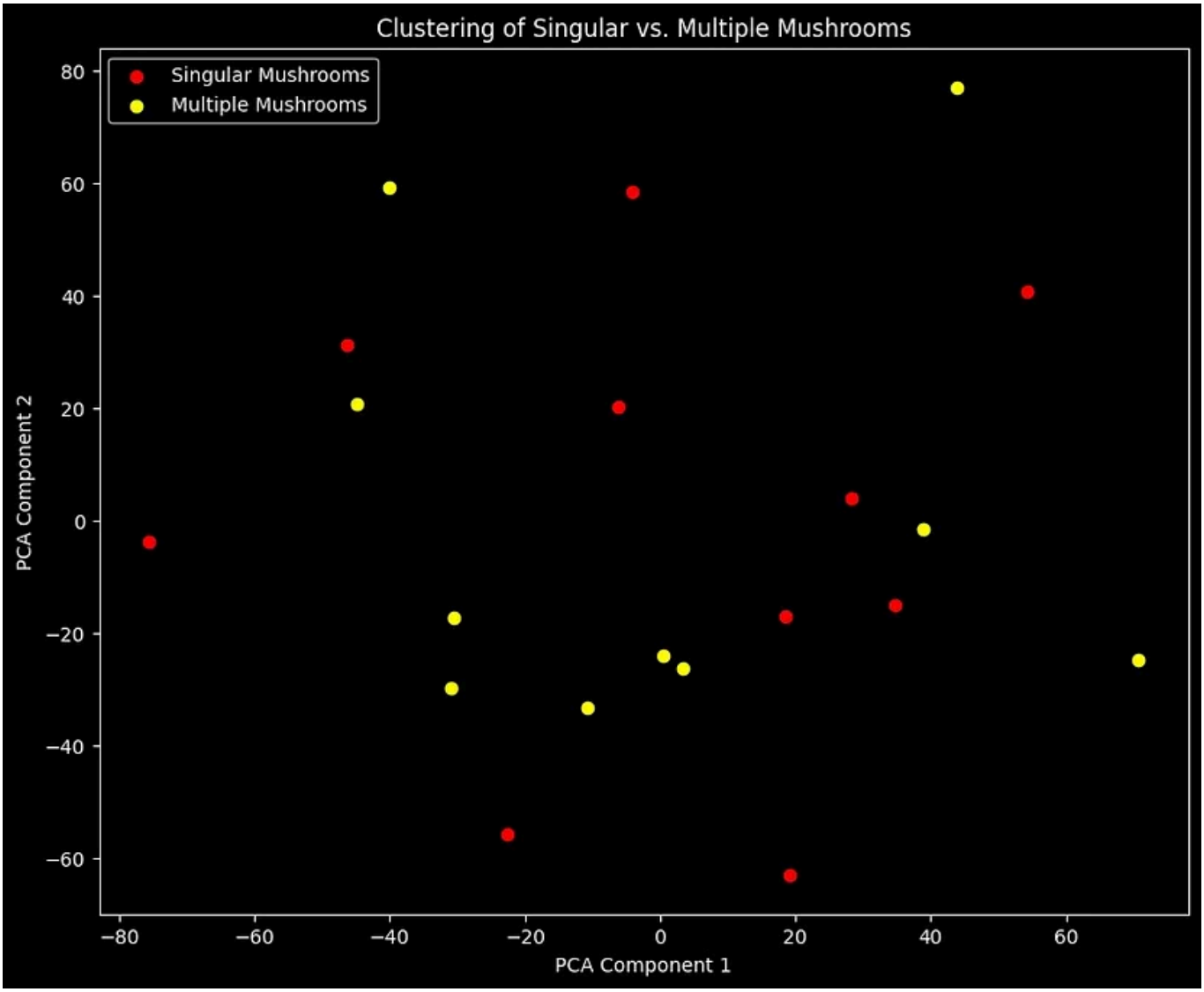
Additionally, we performed alternative groupings and feature selection on our hand picked cluster experiments. The images below show the results when we isolated the number of mushrooms in the image as a feature within the latent space.



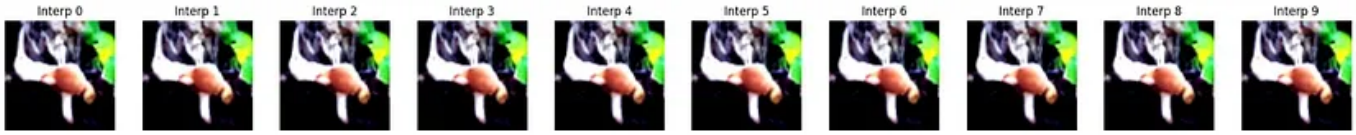
Hand picked single mushrooms

Hand picked multiple mushrooms

```
Singular to Multiple Mushrooms Distance Vector:
[-1.7881392e-08 -2.0861625e-08 -2.3841858e-08 ... -1.7881394e-08
 -3.5762788e-08 -4.0978193e-08]
```

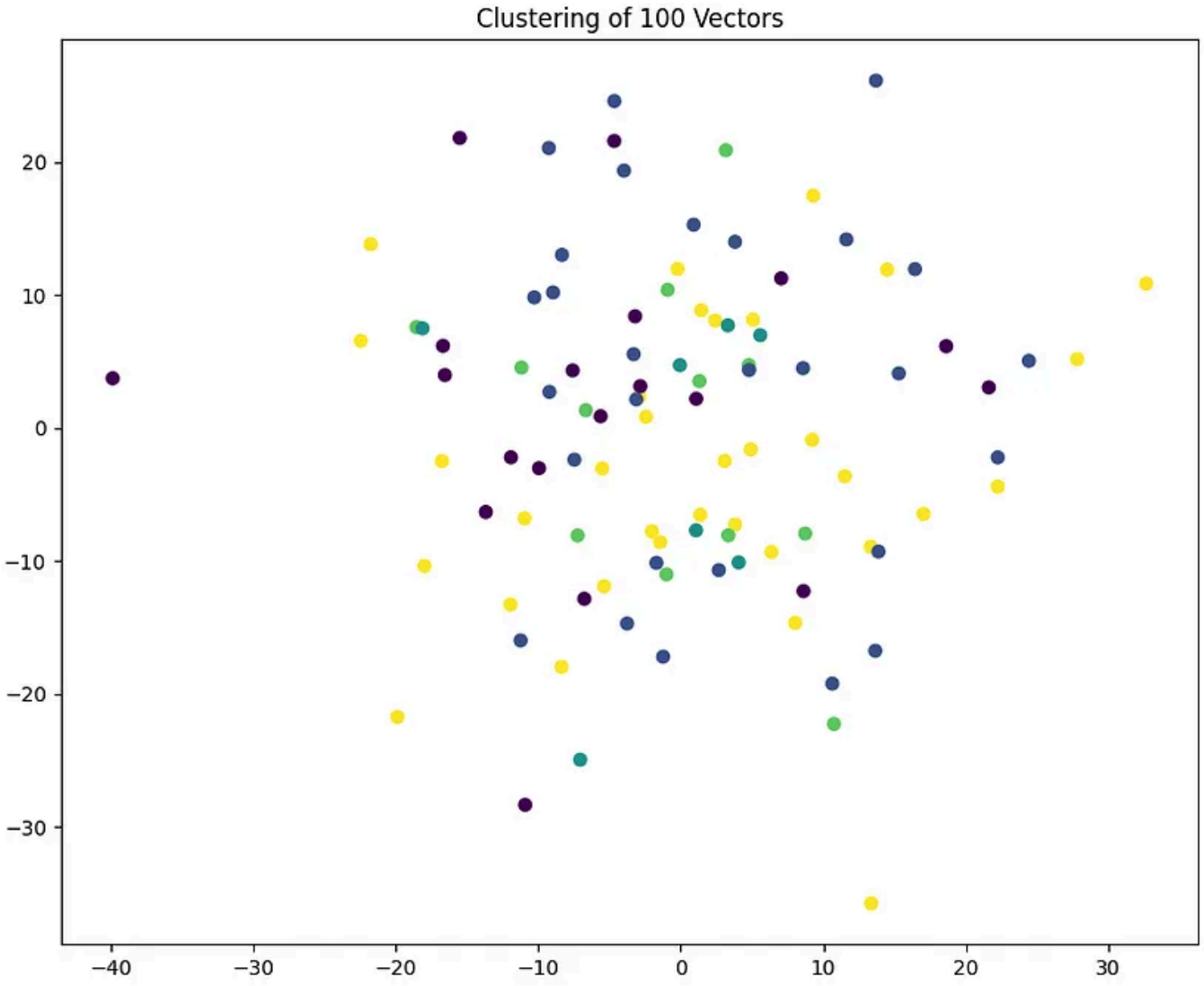


Once again we were getting a very small(exponentially small) distance vector, so we did not expect the interpolations betwen original and modified images to be noticable.



Interpolation example from single/multiple feature experiment

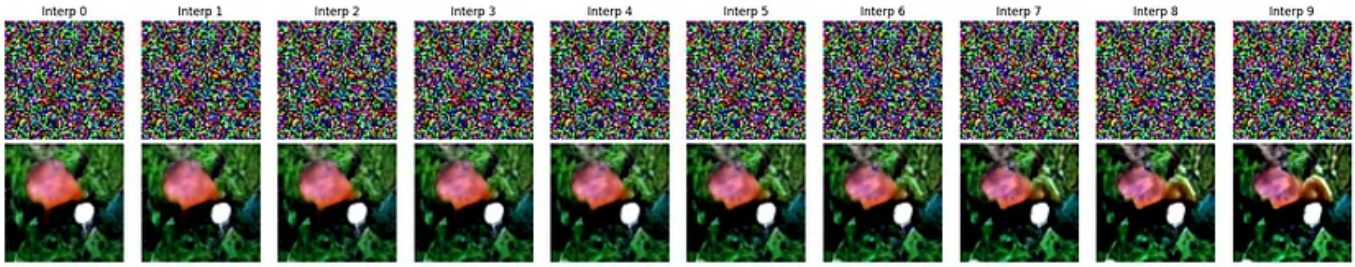
After our hand selected clusters revealed less insights about the latent space organization than we were hoping, we decided to try some automated clustering to see if we could discover an interpretable visual pattern. In this example the images are clustered into 5 groups using K-means clustering.



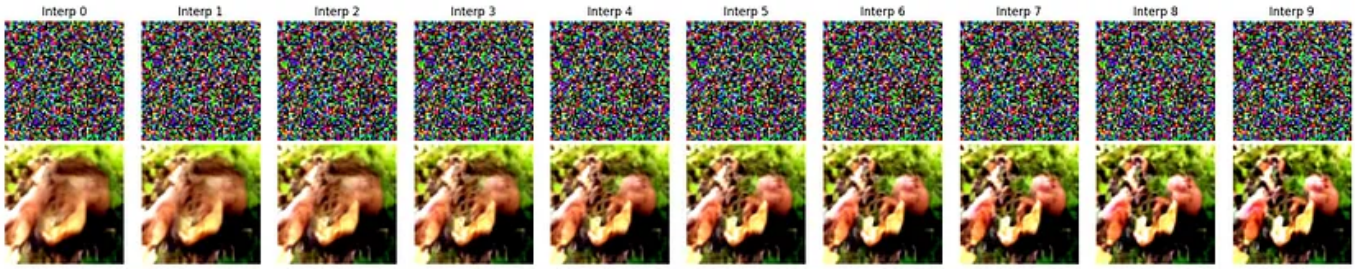

```
Cluster 0 to Cluster 1 Distance Vector:
[-0.45079643  0.34426603  0.12330034 ... -0.41173935 -0.16131556
 -0.13271257]
```

The distance vectors in this clustering example were much larger than the ones from our hand selected experiments. The following images show the images generated when the mushroom on the left is added to the distance vector between Cluster 0 and Cluster 1. Let's call this distance x, by looking at these outputs, we think that adding x to any of our mushroom images changes the mushrooms from a larger, more homogenous one, to a grouping

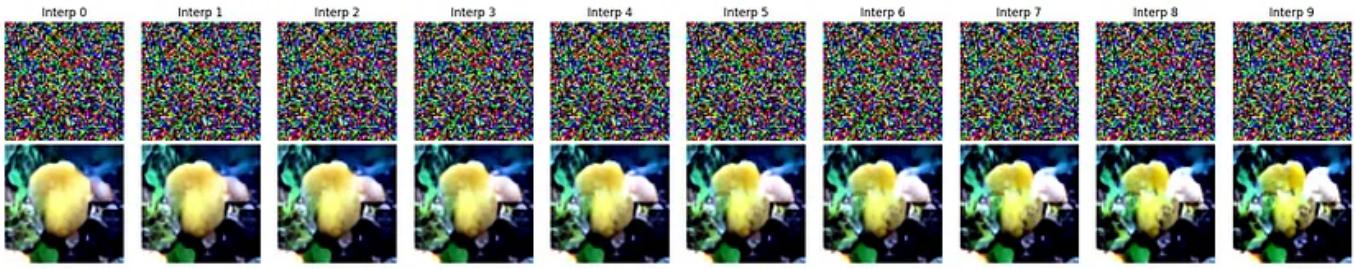
of smaller mushrooms. While not all of the data from the 100 generated showed this, just over 30 of them displayed this pattern.



Goes from 2 mushrooms to 3

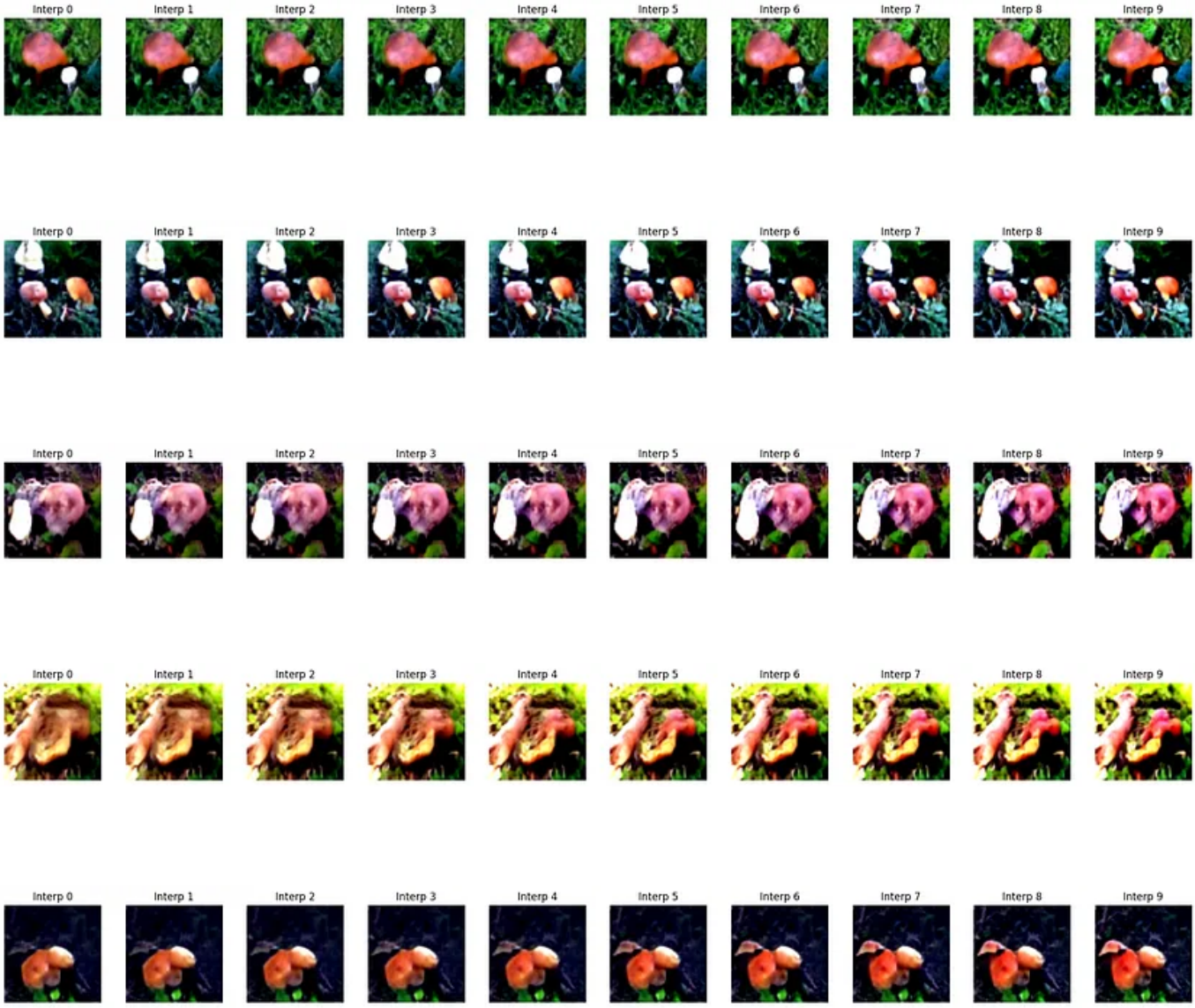


Goes from 1 larger mushroom to 3 or 4 smaller ones



Goes from 1 larger mushroom to 2 or 3 smaller ones

The following images were altered using the distance vector between Cluster 1 and Cluster 4. Results were still less conclusive than we had hoped but at least 20 images seemed to follow a pattern of becoming brighter when added to the distance vector between Clusters 1 and 4.



Limitations

We used Google Colab for the training, running, and analysis of the model. And while it was great for sharing code, and storing data used for training, we were severely limited by the CPU/GPU offerings. Even after purchasing, not one, not two, but three Colab Pro subscriptions we we subject to long dataset building and training times. Because of this computing constraint were also limited for how many images we could generate and analyze.

Conclusions

We look forward to furthering the experiments by playing around with the clustering parameters as well as as the number of images generated. Further exploration would also focus on the latent space of the images half way through diffusion in addition to just the initial noise. If color or number of mushrooms is determined later in the diffusion process, it would be beneficial to conduct this experiment between diffusion steps. Throughout this project, the latent space proved to be an invaluable resource in determining what features a diffusion model understands and operates on. Ultimately, our experiments highlighted the capacity of diffusion models and the ways in which we can evaluate the decision making process of these models.

Our Sources

Learn more about our resources here

Written by Mg Beastrom

0 Followers · 2 Following

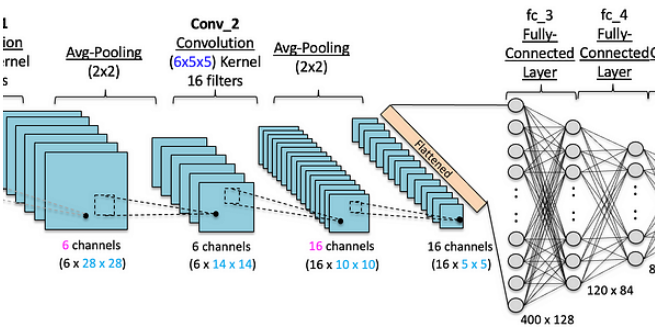
Follow

No responses yet

What are your thoughts?

Respond

Recommended from Medium

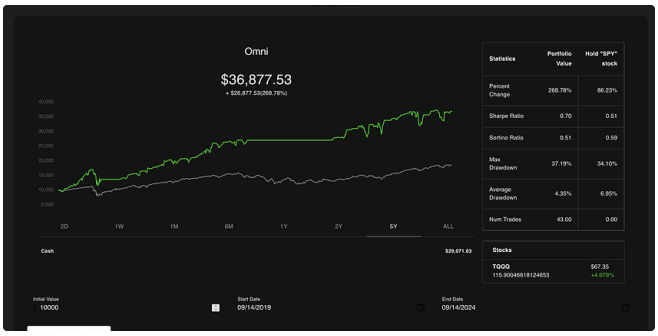


LM Po

The Evolution of Convolutional Neural Networks: From LeNet to...

Convolutional Neural Networks (CNNs) have been the backbone of computer vision...

Sep 21, 2024 51



In DataDrivenInvestor by Austin Starks

I used OpenAI's o1 model to develop a trading strategy. It is...

It literally took one try. I was shocked.

Sep 15, 2024 8.1K 205

Lists

Staff picks

793 stories · 1554 saves

Stories to Help You Level-Up at Work

19 stories · 909 saves

Self-Improvement 101

20 stories · 3190 saves

Productivity 101

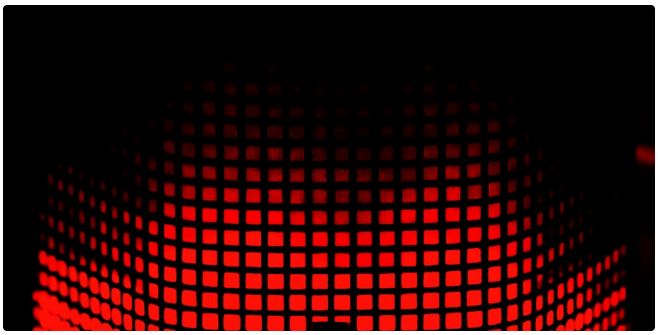
20 stories · 2701 saves

Abisha

Image Feature Extraction using Python - Part I

Basics of Image feature extraction techniques using python

Sep 5, 2024 312 7

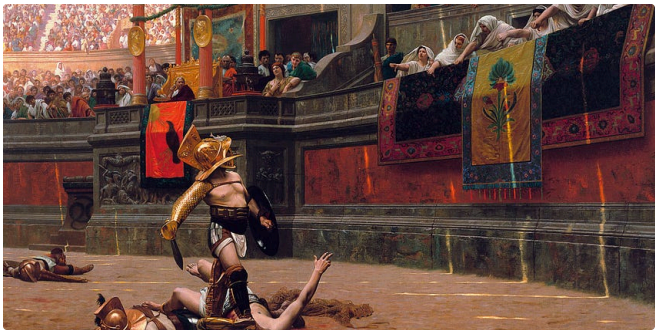


In Cubed by Manish Gupta

Understanding Diffusion Models

An Illustrated Dive into Their Process and Design

Nov 30, 2024 5 1

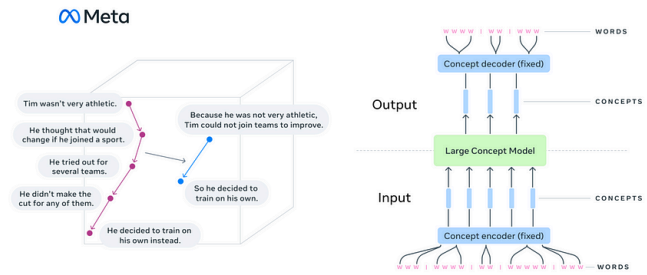


Carlyn Beccia

Trump Fiddles While Musk Burns Down MAGA

Musk unleashes his anger on MAGA Republicans and vows "war."

Dec 28, 2024 18.2K 398



In Level Up Coding by Dr. Ashish Bamania

Meta's Large Concept Models (LCMs) Are Here To Challenge An...

A deep dive into 'Large Concept Model', a novel language processing architecture and...

6d ago 471 7