1 Making a Video from Prompts with Stable Diffusion

by John Robinson

I created this notebook while taking Jeremy Howards's fantastic course, "From Deep Learning Foundations to Stable Diffusion" (https://www.fast.ai/posts/part2-2022.html).

This notebook demonstrates using StableDiffusion (https://stability.ai/blog/stable-diffusion-public-release) to generate a movie from nothing more than a seed image and a sequence of text prompts.

@jeremyphoward (https://twitter.com/jeremyphoward) gives a great explanation of how it works here. (https://twitter.com/jeremyphoward/status/1583667503091548161)

Follow me on twitter johnrobinsn (https://twitter.com/johnrobinsn)

In [1]:
```
# I used conda with python 3.9

def install_dependencies():
    !pip install -qq numpy
    !pip install -qq matplotlib
    !pip install -qq fastai
    !pip install -qq --upgrade transformers diffusers ftfy
    !conda install -y -qq ffmpeg

# Uncomment this line if you'd like to install the dependencies.
#install_dependencies()
```

1.1 Imports and Setup

In [2]:
```
import os
import numpy

# For video display:
from IPython.display import HTML
from base64 import b64encode

import matplotlib.pyplot as plt
from tqdm.auto import tqdm

from PIL import Image
import torch, logging
from torch import autocast
from torchvision import transforms as tfms

from fastcore.all import concat
from pathlib import Path

from huggingface_hub import notebook_login
from transformers import CLIPTextModel, CLIPTokenizer
from transformers import logging
from diffusers import AutoencoderKL, UNet2DConditionModel, LMSDiscrete

# Set device
torch_device = "cuda" if torch.cuda.is_available() else "cpu"
```

1.2 Authenticate with Hugging Face

To run Stable Diffusion on your computer you have to accept the model license. It's an open CreativeML OpenRail-M license that claims no rights on the outputs you generate and prohibits you from deliberately producing illegal or harmful content. The model card (https://huggingface.co/CompVis/stable-diffusion-v1-4) provides more details. If you do accept the license, you need to be a registered user in 🤗 Hugging Face Hub and use an access token for the code to work. You have two options to provide your access token:

- Use the `huggingface-cli login` command-line tool in your terminal and paste your token when prompted. It will be saved in a file in your computer.

- Or use `notebook_login()` in a notebook, which does the same thing.

In [3]:
```
torch.manual_seed(1)
if not (Path.home()/'.huggingface'/'token').exists(): notebook_login()
```

1.3 Load Pretrained Hugging Face Models

In [4]:
```
# Load the autoencoder model which will be used to decode the latents
vae = AutoencoderKL.from_pretrained("CompVis/stable-diffusion-v1-4", s

# Load the tokenizer and text encoder to tokenize and encode the text.
tokenizer = CLIPTokenizer.from_pretrained("openai/clip-vit-large-patch
text_encoder = CLIPTextModel.from_pretrained("openai/clip-vit-large-pa

# The UNet model for generating the latents.
unet = UNet2DConditionModel.from_pretrained("CompVis/stable-diffusion-

# The noise scheduler
# hyper parameters match those used during training the model
scheduler = LMSDiscreteScheduler(beta_start=0.00085, beta_end=0.012, b

# To the GPU we go!
vae = vae.to(torch_device)
text_encoder = text_encoder.to(torch_device)
unet = unet.to(torch_device);
```

```
Some weights of the model checkpoint at openai/clip-vit-large-patch14
were not used when initializing CLIPTextModel: ['vision_model.encode
r.layers.2.layer_norm2.weight', 'vision_model.encoder.layers.6.mlp.fc
2.weight', 'vision_model.encoder.layers.1.layer_norm1.bias', 'vision_
model.encoder.layers.4.self_attn.out_proj.bias', 'vision_model.encode
r.layers.1.self_attn.k_proj.bias', 'vision_model.encoder.layers.1.sel
f_attn.out_proj.bias', 'vision_model.encoder.layers.12.self_attn.out_
proj.weight', 'vision_model.encoder.layers.21.layer_norm1.bias', 'vis
ion_model.encoder.layers.5.mlp.fc2.bias', 'vision_model.encoder.layer
s.9.self_attn.q_proj.weight', 'vision_model.encoder.layers.11.self_at
tn.q_proj.bias', 'vision_model.encoder.layers.18.layer_norm1.weight',
'vision_model.encoder.layers.8.self_attn.q_proj.weight', 'vision_mode
l.encoder.layers.11.mlp.fc1.weight', 'vision_model.encoder.layers.6.s
elf_attn.q_proj.weight', 'vision_model.encoder.layers.10.self_attn.k_
proj.bias', 'vision_model.encoder.layers.20.layer_norm1.bias', 'visio
n_model.encoder.layers.10.self_attn.q_proj.weight', 'vision_model.enc
oder.layers.23.layer_norm2.bias', 'vision_model.encoder.layers.3.self
_attn.k_proj.bias', 'vision_model.encoder.layers.6.layer_norm1.weigh
t', 'vision_model.encoder.layers.11.self_attn.q_proj.weight', 'vision
```

In [5]:
```
vae_magic = 0.18215 # vae model trained with a scale term to get close
```

1.4 Functions to Convert between Latents and Images

In [6]:
```
def image2latent(im):
    im = tfms.ToTensor()(im).unsqueeze(0)
    with torch.no_grad():
        latent = vae.encode(im.to(torch_device)*2-1)
    latent = latent.latent_dist.sample()*vae_magic
    return latent

def _latents2images(latents):
    latents = latents/vae_magic
    with torch.no_grad():
        imgs = vae.decode(latents).sample
    imgs = (imgs/2+0.5).clamp(0,1)
    return imgs

def latents2images(latents):
    latents = latents/vae_magic
    with torch.no_grad():
        imgs = vae.decode(latents).sample
    imgs = (imgs/2+0.5).clamp(0,1)
    imgs = imgs.detach().cpu().permute(0,2,3,1).numpy()
    imgs = (imgs*255).round().astype("uint8")
    imgs = [Image.fromarray(i) for i in imgs]
    return imgs
```

In [7]:
```
def clamp(n,smallest,largest): return max(smallest,min(n,largest))
```

In [76]:

```
def generate_image_from_embedding(text_embeddings,noise,im_latents=Non
    height = 512                          # default height of Stable Dif
    width = 512                           # default width of Stable Diff
    num_inference_steps = 50  #30           # Number of denoising step
    guidance_scale = 7.5                  # Scale for classifier-free gu

    max_length = tokenizer.model_max_length
    uncond_input = tokenizer(
      [""], padding="max_length", max_length=max_length, return_tensor
    )
    with torch.no_grad():
        uncond_embeddings = text_encoder(uncond_input.input_ids.to(tor
    text_embeddings = torch.cat([uncond_embeddings, text_embeddings])

    # Prep Scheduler
    scheduler.set_timesteps(num_inference_steps)

    # Prep latents

    if im_latents != None:
        # img2img
        start_step = 10
        #pass in noise...
        #noise = torch.randn_like(im_latents)
        latents = scheduler.add_noise(im_latents,noise,timesteps=torch
        latents = latents.to(torch_device).float()
    else:
        # just text prompts
        start_step = -1 # disable branching below
        latents = torch.randn((1,unet.in_channels,height//8,width//8))
        latents = latents.to(torch_device)
        latents = latents * scheduler.init_noise_sigma # scale to init

    for i, t in tqdm(enumerate(scheduler.timesteps),total=num_inferenc
        if i > start_step:
            # expand the latents if we are doing classifier-free guida
            latent_model_input = torch.cat([latents] * 2)
            latent_model_input = scheduler.scale_model_input(latent_mo

            # predict the noise residual
            with torch.no_grad():
                noise_pred = unet(latent_model_input, t, encoder_hidde

            # perform guidance
            noise_pred_uncond, noise_pred_text = noise_pred.chunk(2)
            noise_pred = noise_pred_uncond + guidance_scale * (noise_p

            # compute the previous noisy sample x_t -> x_t-1
            latents = scheduler.step(noise_pred, t, latents).prev_samp

    return latents2images(latents)[0],latents
```

In [122]:
```
def get_embedding_for_prompt(prompt):
    tokens = tokenizer([prompt], padding="max_length", max_length=toke
    with torch.no_grad():
        embeddings = text_encoder(tokens.input_ids.to(torch_device))[0
    return embeddings

def generate_image_from_embeddings(embeddings,noise,im_latents=None,po
    # integer part of pos is used for prompt index;
    # fractional part of pos is used to "lerp" between the embeddings
    l = len(embeddings)
    if l > 1:
        index = clamp(int(pos),0,len(embeddings)-2)
        mix = clamp(pos-index,0,1)
        mixed_embeddings = (embeddings[index]*(1-mix)+embeddings[index
        return generate_image_from_embedding(mixed_embeddings,noise,im
    elif l == 1:
        return generate_image_from_embedding(embeddings[0],noise,im_la
    else:
        raise Exception("Must provide at least one embedding")

def generate_movie_from_prompts(prompts,outdir,im_latents=None,fps=12,
                                update_latents=False,update_noise=0.):
    if not os.path.exists(outdir): os.mkdir(outdir)
    num_prompts = len(prompts)
    num_frames = (num_prompts-1) * seconds_per_prompt * fps
    embeddings = [get_embedding_for_prompt(p) for p in prompts]
    generator = torch.manual_seed(seed)
    noise = torch.randn_like(im_latents)
    baseline_noise = noise.clone()
    for f in tqdm(range(0,num_frames)):
        im,new_latents = generate_image_from_embeddings(embeddings,noi
        if update_latents:
            im_latents = new_latents
        if update_noise > 0:
            noise = baseline_noise*(1.-update_noise) + torch.randn_lik
        im.save(f'{outdir}/{f:04}.jpg')
```

In [90]:
```
def show_frames(dir):
    frame_files = Path(dir).ls().sorted(key=lambda f: int(f.with_suffi
    f, axarr = plt.subplots(1,len(frame_files),figsize=(3*len(frame_fi
    for i,f in enumerate(frame_files):
        axarr[i].imshow(Image.open(f))
    plt.show()
    plt.gcf().canvas.draw() # hack to force rendering while cell execu
```

1.5 Create Video from Images

In [12]:
```
def create_movie(dir,movie_name,fps=12):
    if os.path.exists(movie_name): os.remove(movie_name)
    !ffmpeg -v 1 -y -f image2 -framerate {fps} -i {dir}/%04d.jpg -c:v

def embed_movie(movie_name):
    mp4 = open(movie_name,'rb').read()
    data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
    return"""
    <video width=600 controls>
          <source src="%s" type="video/mp4">
    </video>
    """ % data_url
```

2 Show Problem Behavior

There appears to be some numeric problem caused when adding the same noise to a latent generated using that noise as demonstrated below.

In [13]:
```
img = Image.open('./images/oak_tree.jpg').resize((512,512));img
```

Out[13]:

In [14]:
```
img
```

Out[14]:

In [15]:
```
encoded = image2latent(img); encoded.shape
```

Out[15]:
```
torch.Size([1, 4, 64, 64])
```

In [16]:
```
tree_prompts = [
    "An oak tree with bare branches in the winter snowing blizzard ble
    "A barren oak tree with no leaves and grass on the ground",
    "An oak tree in the spring with bright green leaves",
    "An oak tree in the summer with dark green leaves with a squirrel
    "An oak tree in the fall with colorful leaves on the ground",
    "An barren oak tree with no leaves in the fall leaves on the groun
    "An oak tree with bare branches in the winter snowing blizzard ble
]
```

2.0.1 Demonstrate Original "Good (but invalid) Movie" Behavior

Unfortunately due to a silly bug (readonly global variables are evil) every frame of my movie was generated using the encoded image as the input.

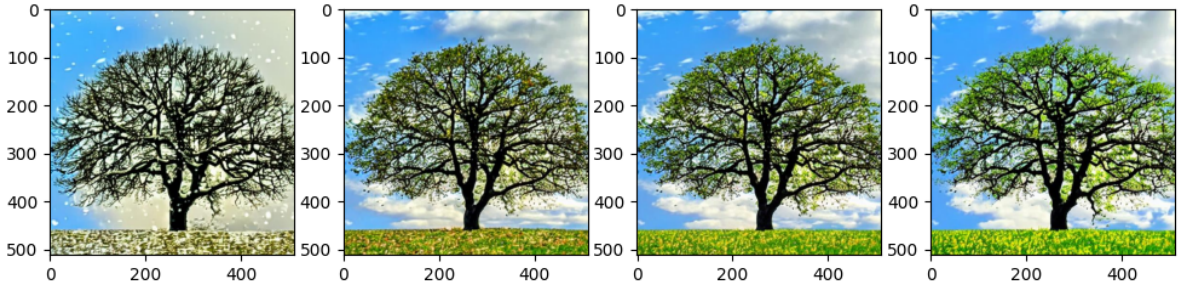
While this did result in "good looking/stable" video. It didn't demonstrate as I'd hoped using the generated latent from the previous frame as input to this frame.

As a starting point, let's demonstrate the "good looking" but invalid baseline by passing in False to both the update_latents and update_noise parameters.

In [17]:
```
# Let's only do the first 3 prompts since it's enough to demonstrate t
tree_prompts_subset = tree_prompts[:3]
```

In [18]:
```
# Let's make a quick movie to help debug the prompts

img_dir = 'tree_baseline'
movie = 'tree_baseline.mp4'

generate_movie_from_prompts(tree_prompts_subset,img_dir,encoded,fps=2,
                            update_latents=False,update_noise=False)


show_frames(img_dir)

#create_movie(img_dir,movie,fps=1)
#HTML(embed_movie(movie))
```

```
0%|          | 0/4 [00:00<?, ?it/s]

0%|          | 0/50 [00:00<?, ?it/s]

0%|          | 0/50 [00:00<?, ?it/s]

0%|          | 0/50 [00:00<?, ?it/s]

0%|          | 0/50 [00:00<?, ?it/s]
```



2.0.2 Demonstrate Degenerate Behavior

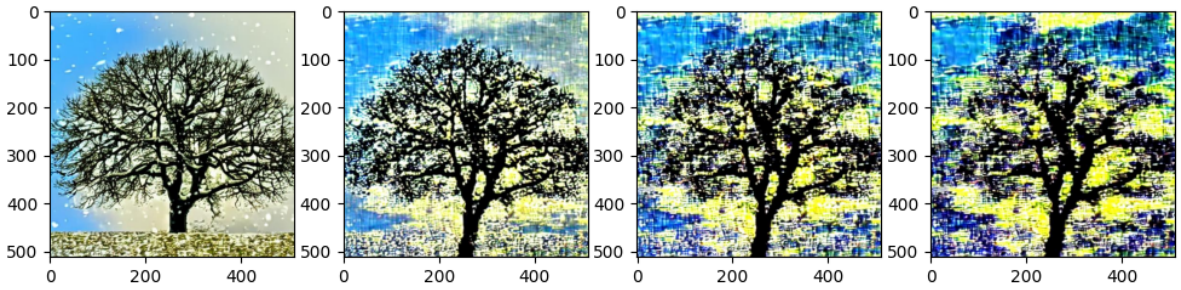
In order to get a "similarly" generated image from frame to frame, I use the same noise on every generated frame. However if you apply this noise to a latent generated with that same noise, the generated latents degrade over time.

To demonstrate this pass in True for the update_latents parameter and False for the update_noise parameter.

This will have the effect of using the latent from the previously generated frame as input to generating this frame.

In [62]:
```
# Let's make a quick movie to help debug the prompts

img_dir = 'tree_update_latent'
movie = 'tree_update_latent.mp4'

generate_movie_from_prompts(tree_prompts_subset,img_dir,encoded,fps=2,
                            update_latents=True,update_noise=False)

show_frames(img_dir)

#create_movie(img_dir,movie,fps=1)
#HTML(embed_movie(movie))
```

```
0%|          | 0/4 [00:00<?, ?it/s]

0%|          | 0/50 [00:00<?, ?it/s]

0%|          | 0/50 [00:00<?, ?it/s]

0%|          | 0/50 [00:00<?, ?it/s]

0%|          | 0/50 [00:00<?, ?it/s]
```



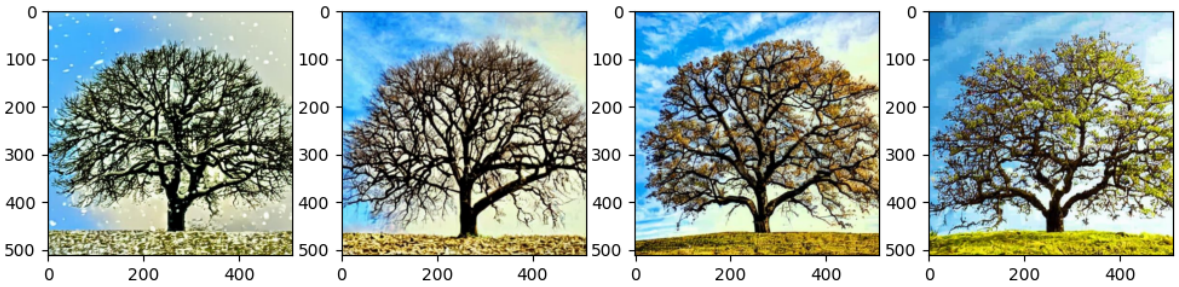
2.0.3 Demonstrate Updating Noise and Latents on Every Frame

This fixes the problem with degradation. But has the undesired effect of causing the "style" of the generated images to vary wildly from frame to frame.

To demonstrate this pass in True for the update_latents parameter and 1.0 for the update_noise parameter.

In [80]:
```
# Let's make a quick movie to help debug the prompts

img_dir = 'tree_update_noise'
movie = 'tree_update_noise.mp4'

generate_movie_from_prompts(tree_prompts_subset,img_dir,encoded,fps=2,
                            update_latents=True,update_noise=1.0)

show_frames(img_dir)

#create_movie(img_dir,movie,fps=1)
#HTML(embed_movie(movie))
```

```
0%|          | 0/4 [00:00<?, ?it/s]
0%|          | 0/50 [00:00<?, ?it/s]
0%|          | 0/50 [00:00<?, ?it/s]
0%|          | 0/50 [00:00<?, ?it/s]
0%|          | 0/50 [00:00<?, ?it/s]
```



3 Playing Around to Get a Deeper Understanding

I've tried a number of things so far. Here is an incomplete list. I haven't retained all the results of these experiments but none have worked out so far. I will try to capture my experiments going forward in this section.

- ☑ Stay in latent space as much as possible... don't do any interim l2i(i2l(l)) transformations
- ☑ Try blending in original image to generated image at each frame to try to "blend out" compression artifacts
- ☑ Try blending original latent with each frame latent (average).
- ☑ Try adding small amounts of new noise when passing in the previous frames latent.

3.1 Experiment with Adding Some Amount of New Noise for each Frame

Since the degenerate behavior *seems* to go away when use new noise on every frame. The problem to solve is that with completely new noise the generated images change significantly in style on each frame. So the idea here is that rather than using a completely new noise pattern, can we instead use the original noise pattern and add in some **small amount** of additional noise and preserve our desired similarity with the original image and the previous frame?

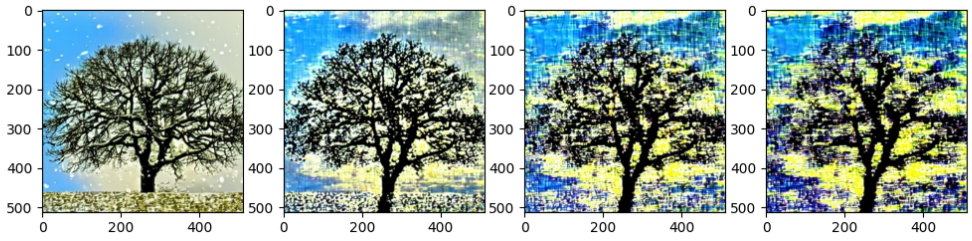
Since there might be different ways of "adding in new noise" for each frame, first we'll try:

```
noise = baseline_noise*(1.-update_noise) + torch.randn_like(im_
latents)*update_noise
```

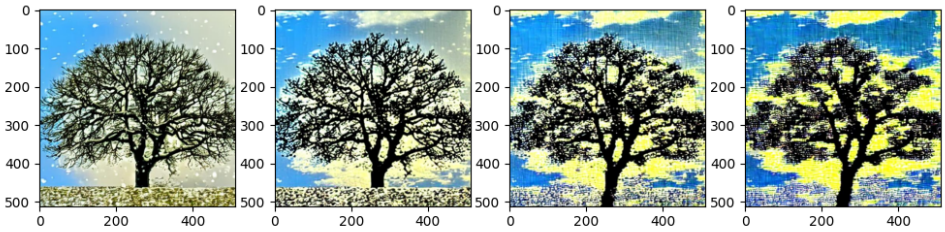
In [87]:
```
%matplotlib notebook
```

In [91]:
```
# Let's make a quick movie to help debug the prompts

# amount of noise to blend in.  small to large.  We know that 1.0 fixe
noise_amounts = [0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

for i,n in enumerate(noise_amounts):

    print(f'Adding in {n} new noise.')

    img_dir = f'tree_update_noise{i}'
    movie = f'tree_update_noise{i}.mp4'

    generate_movie_from_prompts(tree_prompts_subset,img_dir,encoded,fp
                                update_latents=True,update_noise=n)

    show_frames(img_dir)
```

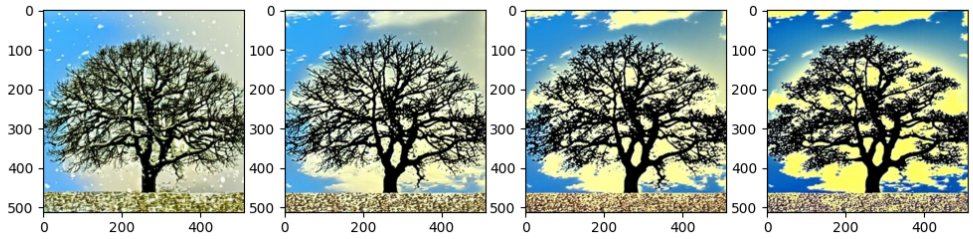
```
Adding in 0.01 new noise.

  0%|          | 0/4 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

<IPython.core.display.Javascript object>
```



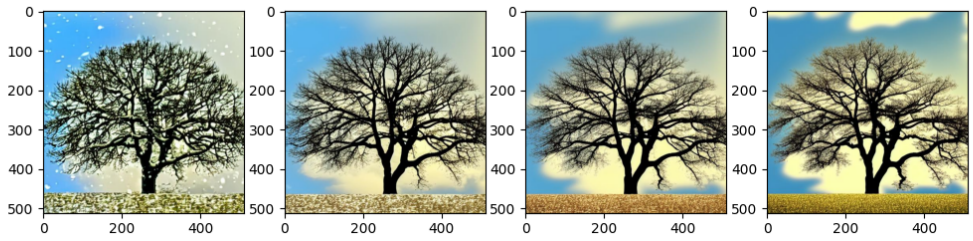

```
Adding in 0.05 new noise.

  0%|          | 0/4 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

<IPython.core.display.Javascript object>
```



```
Adding in 0.1 new noise.
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
<IPython.core.display.Javascript object>
```

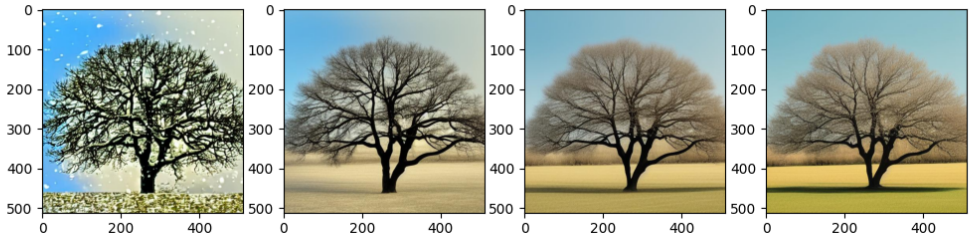


```
Adding in 0.2 new noise.
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
<IPython.core.display.Javascript object>
```

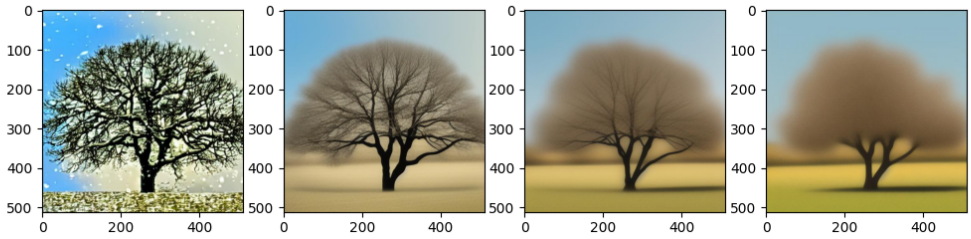


```
Adding in 0.3 new noise.
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
```

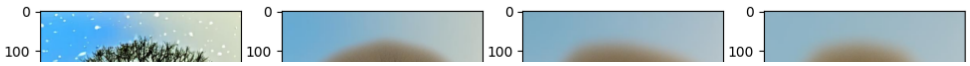
```
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
<IPython.core.display.Javascript object>
```



```
Adding in 0.4 new noise.
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
<IPython.core.display.Javascript object>
```



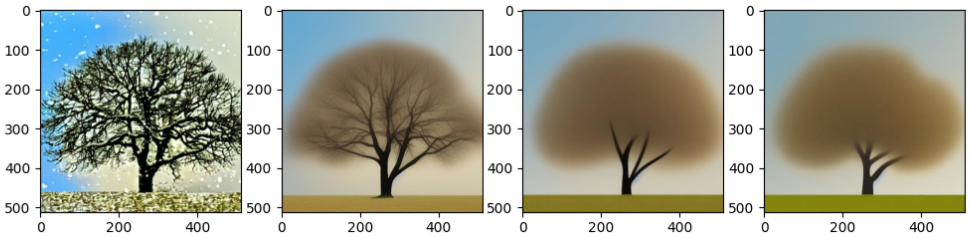
```
Adding in 0.5 new noise.
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
<IPython.core.display.Javascript object>
```



```
Adding in 0.6 new noise.
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
```

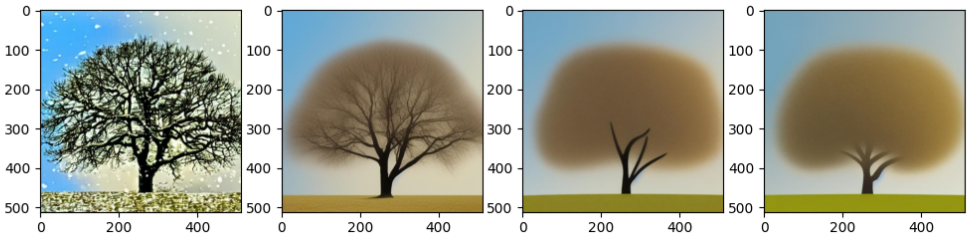
```
<IPython.core.display.Javascript object>
```



```
Adding in 0.7 new noise.
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
```

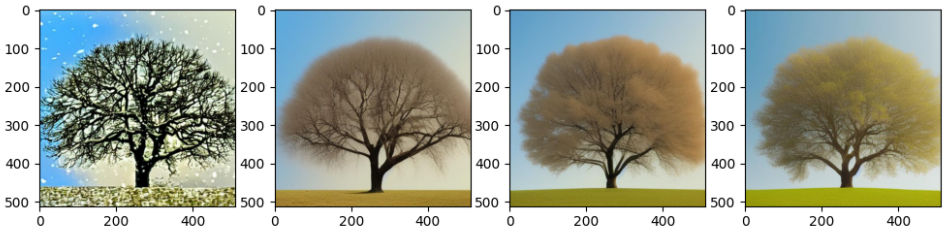
```
<IPython.core.display.Javascript object>
```



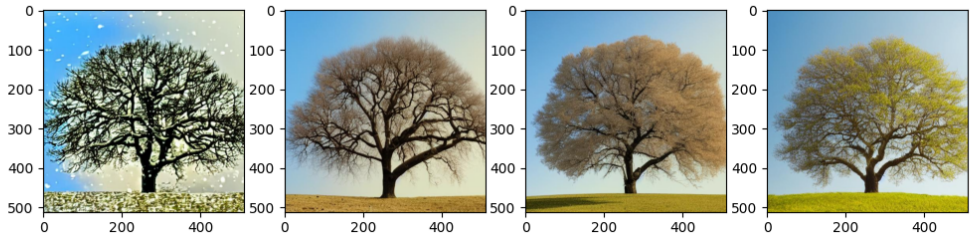
```
Adding in 0.8 new noise.
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
```

```
<IPython.core.display.Javascript object>
```



```
Adding in 0.9 new noise.
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
<IPython.core.display.Javascript object>
```



3.1.1 Summary

Still seeing degenerate behavior with small amounts of noise and relatively large amounts of noise up thr 0.8. But the degenerate behavior changes as we get larger amounts of noise you can funny "diffuse" image generatiion.

At 0.9*new noise the generated images start to "come into focus" but they are not similar to the previous frame.

Given that we get these get these degenerate behavior. I'm not sure if blending in the new noise this way works the way I think it does.

3.2 Blending Gaussian Distributions

Can I blend two gaussian samples together and still get a gaussian?

In [111]:
```
a = torch.randn(100000);a.mean(),a.std()
```

Out[111]:
```
(tensor(-0.0039), tensor(1.0021))
```

In [112]:
```
b = torch.randn(100000);b.mean(),b.std()
```

Out[112]:
```
(tensor(0.0007), tensor(1.0014))
```

In [107]:
```
mixed = a*0.9 + b*0.1;mixed.mean(),mixed.std()  # This gives a suspici
```

Out[107]:
```
(tensor(0.0029), tensor(0.9045))
```

In [117]:
```
mixed2 = a + b*0.1;mixed2.mean(),mixed2.std()  # This looks better; le
```

Out[117]:
```
(tensor(-0.0038), tensor(1.0071))
```

In [126]:
```
# a version for easy sharing...
c = torch.randn(100000)*0.9+torch.randn(100000)*0.1;c.mean(),c.std()
```

Out[126]:
```
(tensor(0.0074), tensor(0.9041))
```

In [126]:
```
# a version for easy sharing...
c = torch.randn(100000)*0.9+torch.randn(100000)*0.1;c.mean(),c.std()
```

Out[126]:
```
(tensor(0.0074), tensor(0.9041))
```

3.3 Try a different noise blending approach

From the section above we have to be careful about how we blend in new noise to preserve the gaussian distribution. The following linear interpolation between two gaussian noise samples doesn't work since the samples aren't uniformly distributed.

```
noise = baseline_noise*(1.-update_noise)+torch.randn_like(im_la
tents)*update_noise
```

So let's try this instead... baseline_noise is not scaled to preserve it's gaussian distribution and we add in a small scaled amount from a new gaussian sample. *Note: the new noise probably should be from some uniform distribution?*

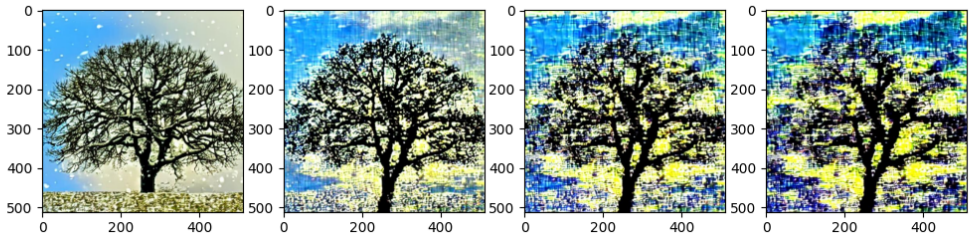
```
noise = baseline_noise+torch.randn_like(im_latents)*update_nois
e
```

In [127]:

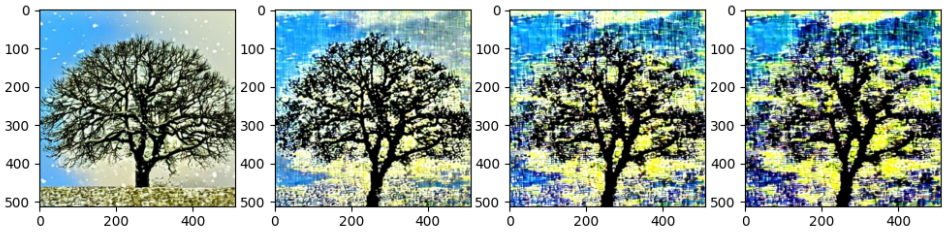
```
# a variant on the movie generation function that uses a different "no
def _generate_movie_from_prompts_noise2(prompts,outdir,im_latents=None
                                        update_latents=False,update_noise=0.):
    if not os.path.exists(outdir): os.mkdir(outdir)
    num_prompts = len(prompts)
    num_frames = (num_prompts-1) * seconds_per_prompt * fps
    embeddings = [get_embedding_for_prompt(p) for p in prompts]
    generator = torch.manual_seed(seed)
    noise = torch.randn_like(im_latents)
    baseline_noise = noise.clone()
    for f in tqdm(range(0,num_frames)):
        im,new_latents = generate_image_from_embeddings(embeddings,noi
        if update_latents:
            im_latents = new_latents
        if update_noise > 0:
            #noise = baseline_noise*(1.-update_noise) + torch.randn_li
            noise = baseline_noise + torch.randn_like(im_latents)*upda
        im.save(f'{outdir}/{f:04}.jpg')
```

In [125]:
```
# Let's make a quick movie to help debug the prompts

# amount of noise to blend in.  small to large.  We know that 1.0 fixe
# Given how we're blending in this scenario only small amounts of nois
noise_amounts = [0.01,0.05,0.1]

for i,n in enumerate(noise_amounts):

    print(f'Adding in {n} new noise.')

    img_dir = f'tree_update_noise2_{i}'
    movie = f'tree_update_noise2_{i}.mp4'

    _generate_movie_from_prompts_noise2(tree_prompts_subset,img_dir,en
                                update_latents=True,update_noise=n)

    show_frames(img_dir)
```

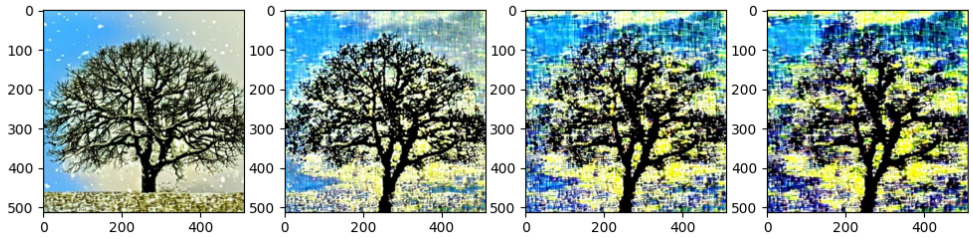
```
Adding in 0.01 new noise.

  0%|          | 0/4 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

<IPython.core.display.Javascript object>
```




```
Adding in 0.05 new noise.

  0%|          | 0/4 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

  0%|          | 0/50 [00:00<?, ?it/s]

<IPython.core.display.Javascript object>
```



```
Adding in 0.1 new noise.
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
  0%|          | 0/50 [00:00<?, ?it/s]
<IPython.core.display.Javascript object>
```



In [ ]: