Unconditional-to-conditional Transfer and Optimization for Web-based Skybox GAN

Crystal Kwong

Introduction: Project Goal

- Overall goal is to build an AI skybox generator application
  - Use transfer learning to train conditional StyleGAN2-ADA model, given a pre-trained unconditional model
  - Optimize the StyleGAN2-ADA model by quantizing it
  - Deploy model on web page
  - Use model to generate an image usable for a 3D skybox

Introduction: Generative AI

“Using AI effectively often starts with clearly defining your goal. What problem are you trying to solve or what task are you trying to accomplish? Once you have a clear objective, explore different AI tools and platforms that align with your needs, whether it's for writing, image generation, data analysis, or automation” *(AI-generated text)*

Everything on this page is AI generated. Images, text - AI can generate a lot of different content!

Introduction: AI in Games

Potentially use for:

- Characters
- Dialogue
- Behavior
- **Visual effects**
- **Stylized environments**

"Using AI effectively often starts with clearly defining your goal. What problem are you trying to solve or what task are you trying to accomplish? Once you have a clear objective, explore different AI tools and platforms that align with your needs, whether it's for writing, image generation, data analysis, or automation" (AI-generated text)

Introduction: AI in Games

Implementing ‘true’ AI in games comes with problems:

- High effort, cost prohibitive
  - Need more programmers to dedicate effort to set up the AI
- Performance expensive
  - Games want to be fast and responsive
- Unexpected results or AI misinterpretation
  - Not desired in video games
  - "If you care more about ‘plausibility’ than ‘intelligence’, experience shows that hand-tuned solutions go a long way further than emergent ones" [Pfau et al. (2020)]
    - hand-tuned solutions such as hard-coded if-statements

Introduction: AI in Games

Viable possibilities: visual effects? Environments?

An environment could be generated in the form of a skybox…

Introduction: Skybox

- A skybox:
  - helps build an environment
  - renders an illusion of an infinite 3D environment
    - is implemented by wrapping a six-sided texture over a cube, or a panoramic image over a sphere
      - view is from ‘inside’ the cube/sphere

Since a skybox is basically an image, a skybox can be created using an AI image generator.

source: freepik.com

Presentation Overview

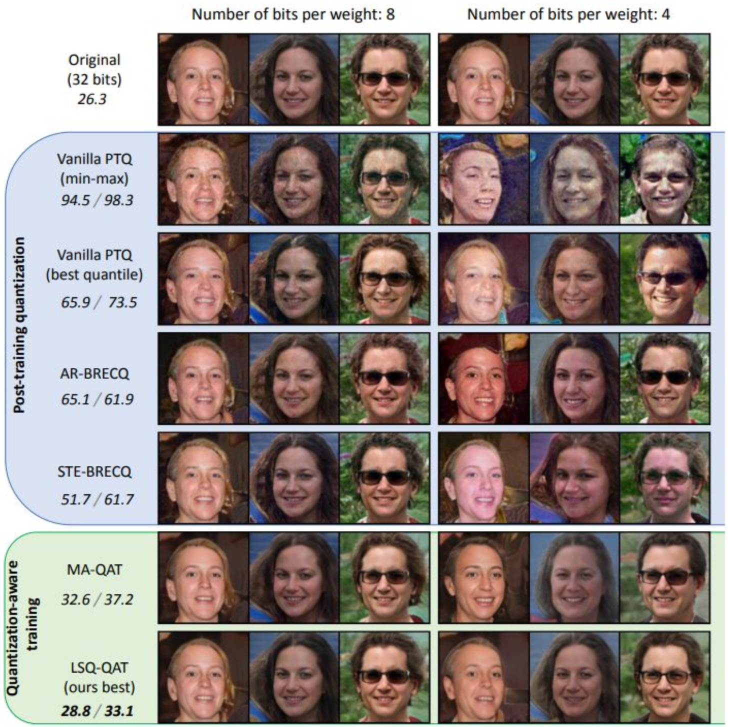
- Related Work
  - Hyper-modulation, quantization of GANs, quantization of sequential StyleGAN2, StyleGAN2-ADA Web deployment, Existing AI Skybox Generator Websites
- Background
  - GANs/cGANs, StyleGAN/StyleGAN2/StyleGAN2-ADA, Training a model, Frechet Inception Distance Score (FID score), Transfer learning, Quantization, Open Neural Network Exchange (ONNX) model, skybox, Unity
- Experiments
  - Transfer learning and Quantization
- Web page deployment
- Results
  - Demonstration of using generated skybox in Unity

Related Work - Unconditional to conditional GAN Transfer with Hyper-modulation

- Laria et al. (2022) proposes **hyper-modulation** to facilitate **unconditional-to-conditional** knowledge transfer for GANs, using StyleGAN as their example model
  - **Unconditional-to-conditional** GAN knowledge transfer is not as thoroughly researched
  - Hyper-modulation idea: on-the-fly weight modulation by the hypernetwork to produce target conditional model weights

Related Work - Quantization of GANs

- Andreev and Fritzler (2022) simulated StyleGAN quantization using PyTorch fake quantization library and recorded the results of 4-bit, 8-bit quantization
  - Demonstrated effect on image quality
    - Falls short of truly quantizing the GAN, since fake quantization only simulates quantization



source: https://arxiv.org/pdf/2108.13996

Related Work - Quantization of Sequential StyleGAN2

- An example of quantizing StyleGAN2
- Script to **quantize StyleGAN2,** by a project managed by Intel
  - However, not for original StyleGAN2 model; only quantized a **rewritten sequential version** of the StyleGAN2 model

Related Work - StyleGAN2-ADA model Web Deployment

- A web page example (https://www.guidodejong.nl/hack/running-stylegan2-ada-in-browser/) used ONNX Runtime to deploy a StyleGAN2-ADA model that was converted into ONNX format

Related Work - AI Skybox Generator Websites

- Two notable sites: **Skybox AI** (by Blockade Labs) and **Rosebud.ai**
- Both generators accept a text prompt and generate a skybox along with a 3D preview
- Text prompt input allows content of generated skyboxes to include more detail than just skies, such as landscapes, trees, or castles.
- These examples demonstrate the boundless potential of AI skybox generation

As for my project: aim to fulfill a use case of choice-based skybox generator, rather than text-based (why: see next slide)

Related Work - AI Skybox Generator Websites

Skybox AI sample (left); Rosebud.ai sample (right)

Background

This section defines:

- Generative Adversarial Network (GAN) and conditional GAN (cGAN)
- StyleGAN, StyleGAN2, StyleGAN2-ADA
- Training a model
- Frechet Inception Distance (FID) Score
- Transfer Learning
- Quantization
- Open Neural Network Exchange (ONNX) model
- Skybox
- Unity

Background - Generative Adversarial Network (GAN)

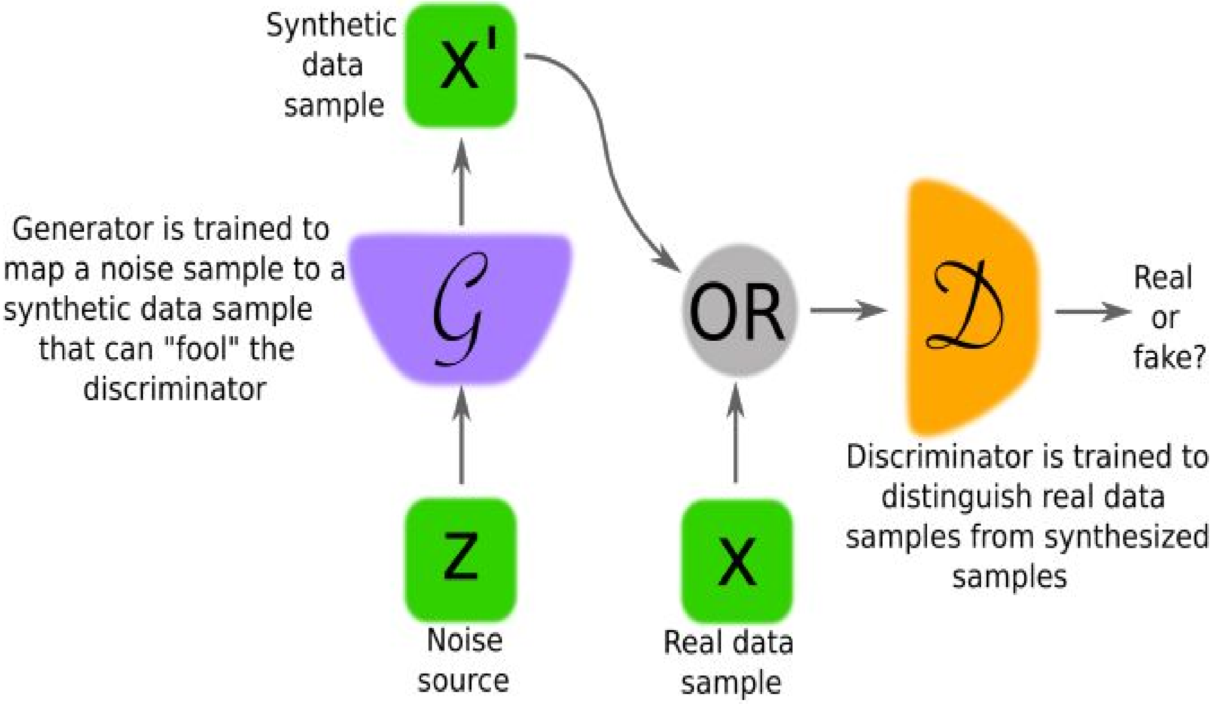
- **GAN** is a type of generative AI model particularly useful for generating high quality **images**

- GAN architecture consists of two models:
    - **Generator** - synthesizes output
    - **Discriminator** - classifies whether output is ‘real’ or ‘fake’, given a dataset of ‘real’ data

Background - Generative Adversarial Network (GAN)

How GAN works:

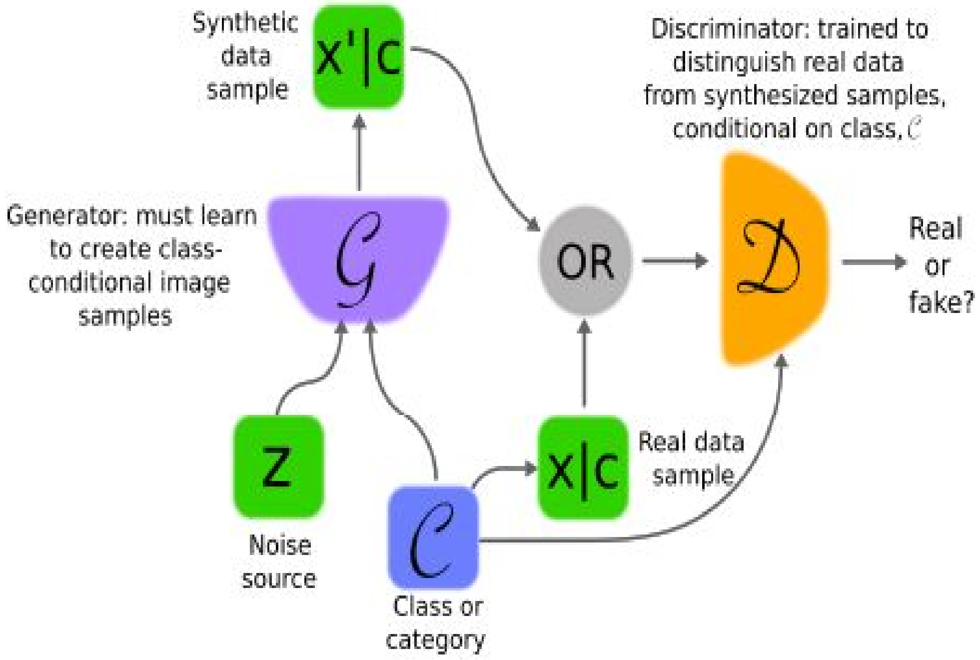
1. 'z' (random noise) is input into the generator
2. Generator outputs content
3. Discriminator classifies the generator output as 'real or 'fake'



Generator (synthesizes output) and discriminator (classifies output as 'real' or 'fake').

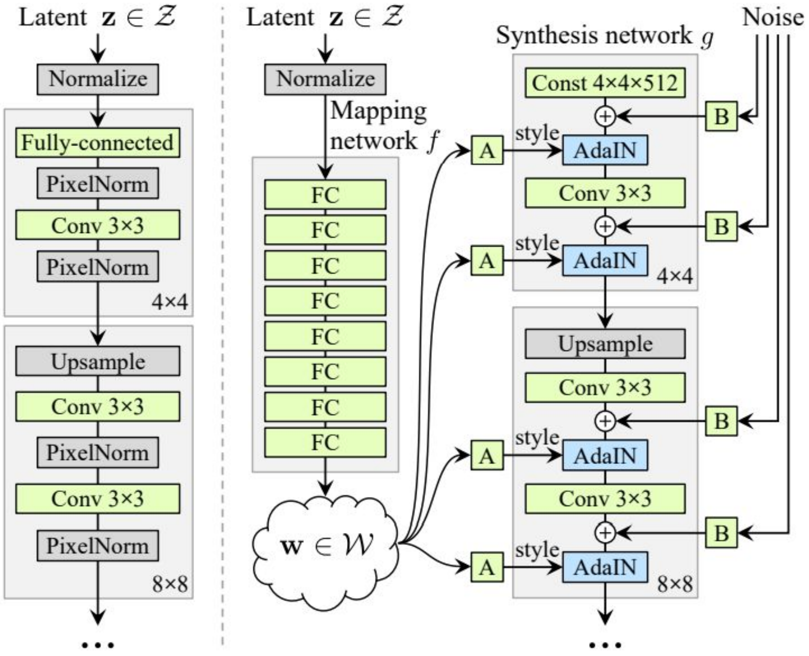
Background - Conditional GAN

- Conditional GAN (cGAN) is a type of GAN
  - Conditional GAN **can generate class-conditional outputs**, as opposed to random outputs of a GAN
  - To obtain a cGAN, a conditioning label can be added to the input of both the generator and the discriminator



Background - StyleGAN

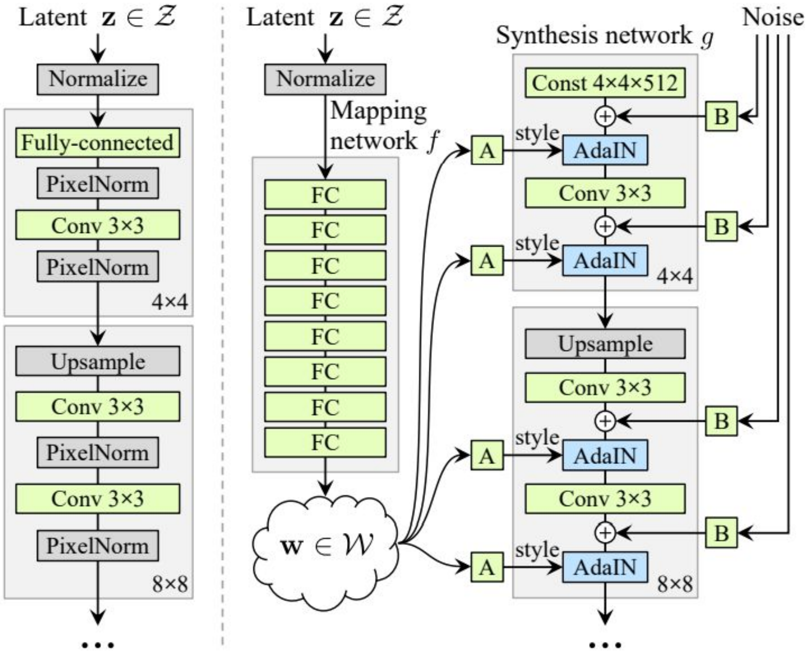
- GAN with redesigned generator architecture
  - Traditional GAN: latent code fed to input layer
  - StyleGAN: **map** the input latent code to an **intermediate latent space** ‘w’
  - StyleGAN **disentangles** image attributes (i.e. hair style, face shape) through mapping styles inside intermediate latent space
  - How is this style mapping done?



(a) Traditional
(b) Style-based generator

Background - StyleGAN

- Learned affine transforms convert the mapped input into styles
    - converted styles are fed into the adaptive instance normalization (AdaIN) operations
- AdaIN: aligns mean and variance of the style features (pixel values) to the target image features, effectively drawing the target image in the new style



(a) Traditional
(b) Style-based generator

Background - StyleGAN

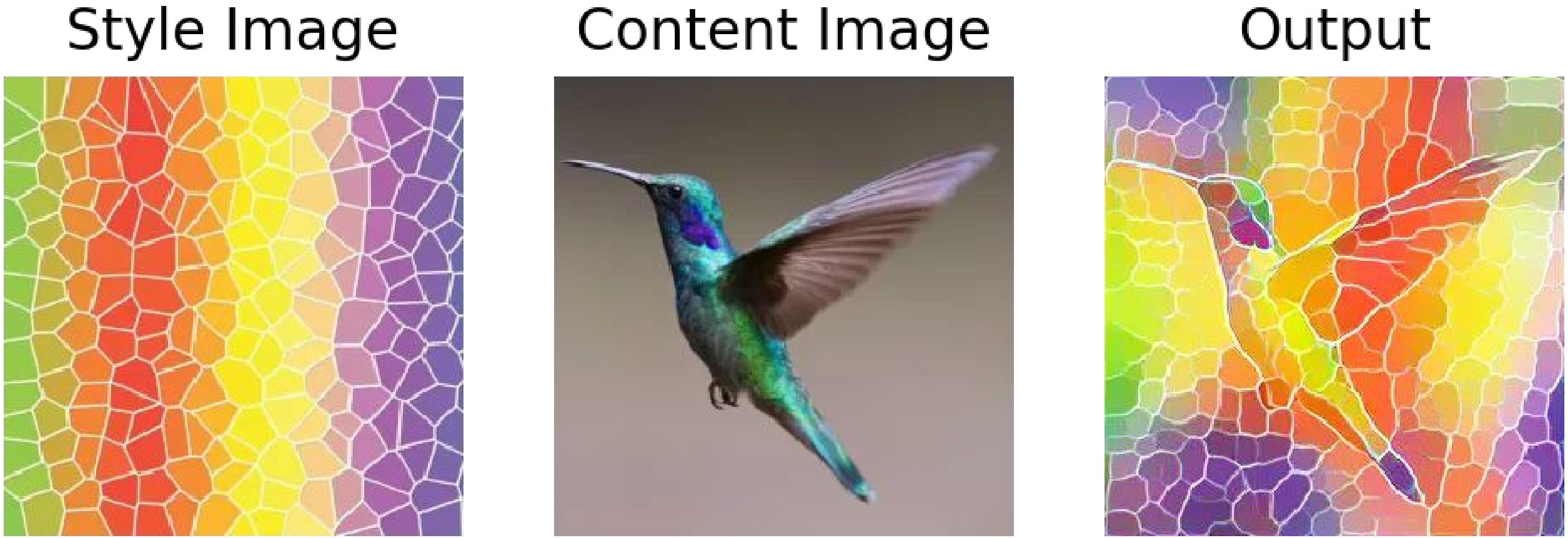
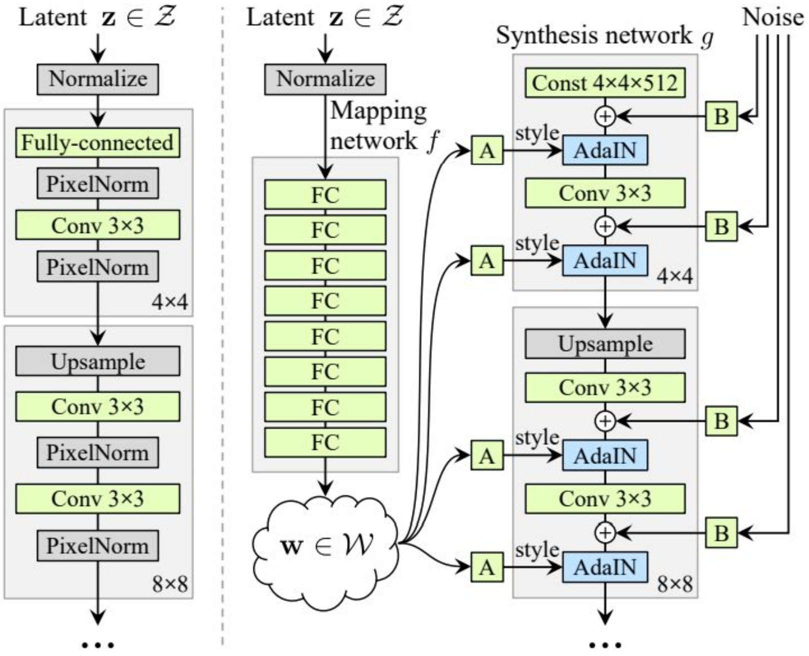


image source: https://medium.com/data-science/an-intuitive-understanding-to-neural-style-transfer-e85fd80394be

Background - StyleGAN

- StyleGAN generator also adds **noise** at each layer
  - noise helps create random-like features such as freckles
- **Progressive growing** (smaller -> larger resolution) until final high resolution output image is reached
- Overall, with these changes to the traditional generator, StyleGAN generates higher quality images than a traditional GAN



(a) Traditional
(b) Style-based generator

Upsample -> progressive growing

Background - StyleGAN2 and StyleGAN2-ADA

- Built on StyleGAN with improvements
    - **StyleGAN2:** fixed blob-like artifact in StyleGAN by removing normalization (AdaIN)
        - replaced AdaIN with demodulation, a weaker version of scaling down output feature maps that resulted in no artifact

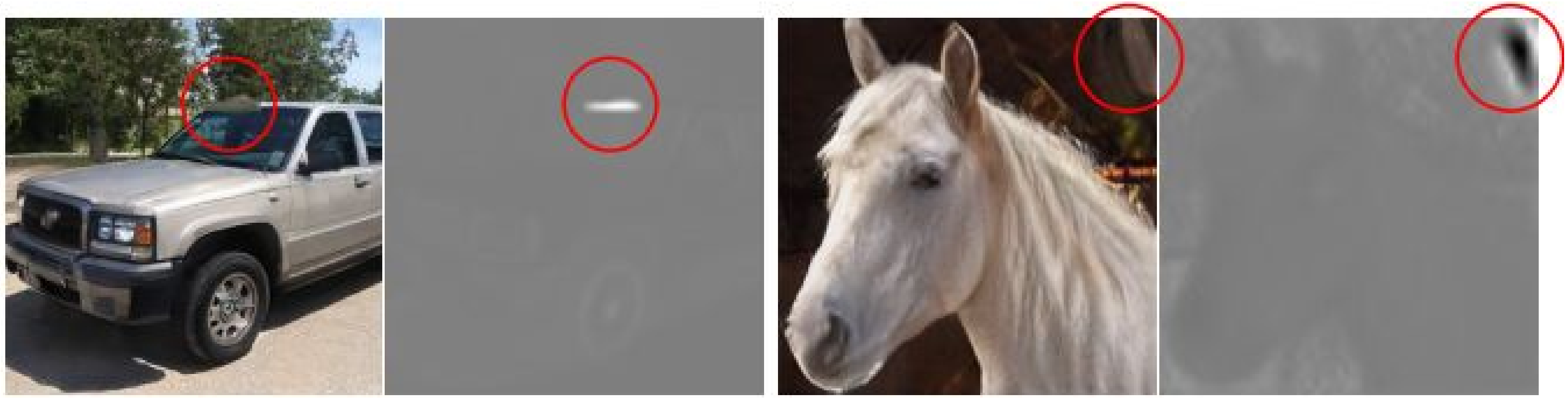


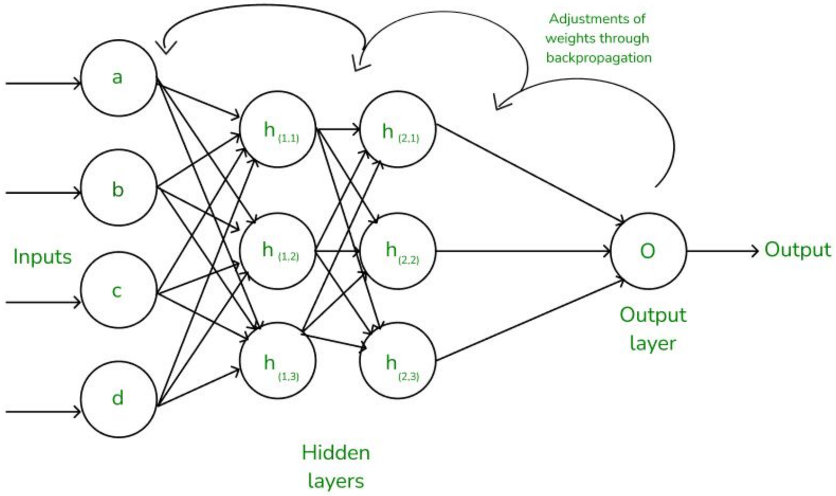
image source: adapted from https://arxiv.org/pdf/1912.04958

Background - StyleGAN2 and StyleGAN2-ADA

- Built on StyleGAN with improvements
    - **StyleGAN2-ADA** allows small training datasets - even just a few thousand training images - to produce good results
        - achieved this by introducing “adaptive discriminator augmentation” (ADA) to dynamically reduce discriminator overfitting

Background - Training a model

- Training a model refers to an iterative process of feeding an input through the model's layers to obtain an output
  - Through **feedback from the loss function, backpropagation**, and repeated iterations, **weights are gradually adjusted** closer toward the 'correct' value



Source: https://www.geeksforgeeks.org/backpropagation-in-neural-network/

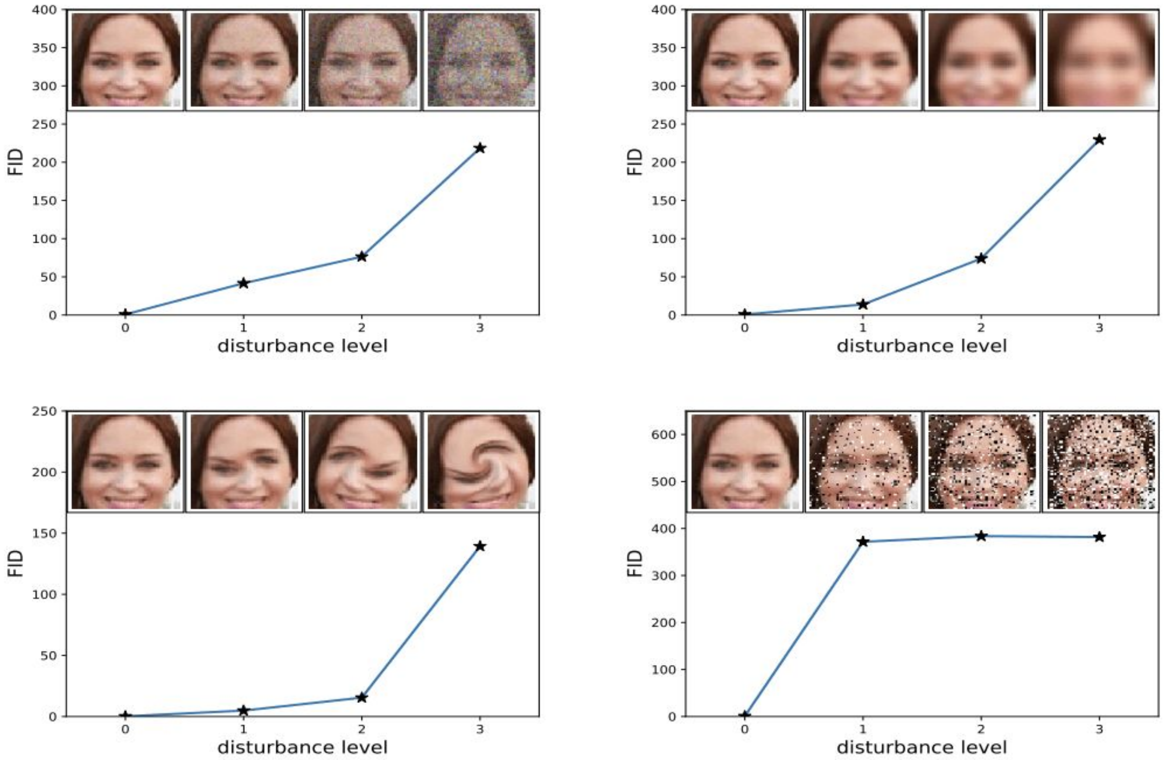
Background - Frechet Inception Distance (FID) Score

- Metric to **measure quality and diversity of images** generated by a GAN
- Calculates “distance between images in pixel space”, comparing the model’s generated images against the true dataset images
- Lower FID score is preferable
    - lower score = more diverse and high quality images
- The FID score is calculated according to the formula below:

$$FID(x,g) = ||u_x - u_g||^2 + Tr\left(\sum_x + \sum_g -2\left(\sum_x \sum_g\right)^{\frac{1}{2}}\right)$$

μ = the mean, ‘Tr’ = the function of summing the main diagonal (top left to bottom right elements) of the matrix, Σ = covariance matrix for feature vector. ‘X’ is the true image and ‘g’ is the model-generated image

Background - Frechet Inception Distance (FID) Score



Correlation between FID score with image quality.
Ideally, FID score of 0 means the GAN's generated images are exactly the same as the dataset's images.

source: https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/

Background - Transfer Learning

- Transfer learning is a method of utilizing weights of a pre-trained model to speed up training of another model
- When training from scratch, a model is initialized with completely random weights
    - Transfer learning **bypasses random weight initialization by using pre-trained weights**
    - The model starts training from a state of partial knowledge, needing less time to train than if it had started from a randomized state of no knowledge

Background - Transfer Learning

- Transfer learning is a broad term, and there are specific methods to apply transfer learning, such as fine-tuning

- **Fine-tuning**
  - Retrains every weight of a pre-trained model, gradually adapting the entire model to the new data
  - Can be used to adapt a model's domain to another domain (i.e. images of cars to people)

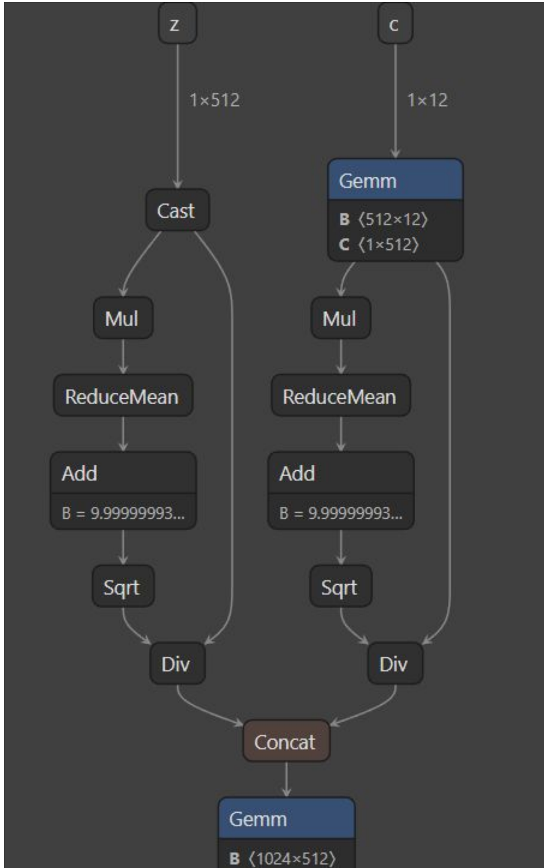
Background - Quantization

- Quantization replaces higher-precision computations (such as 32-bit float) with lower-precision computations (such as 8-bit int)
  - Reduces model size and computation load during inference
    - Downside: model accuracy loss from lower precision weights/computations

- Quantization of GANs is also more difficult and not as thoroughly researched compared to quantization of classifier models

Background - Open Neural Network Exchange (ONNX) Model

- Open Neural Network Exchange (ONNX) is an **interoperability tool** that can represent models from various frameworks, such as PyTorch and TensorFlow, as a common file format



Background - Skybox

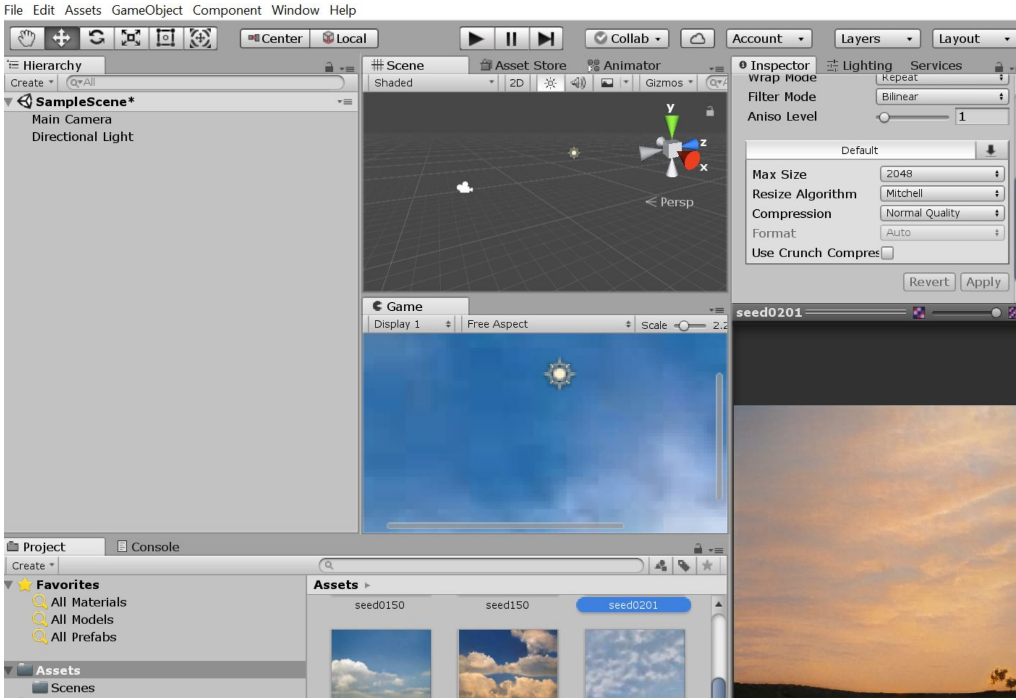
- In 3D graphics, a skybox is a technique used to create the illusion of an encompassing environment
- Making a skybox:
  - Cubemap texture wrapped around the inside of a cube
    - six texture sides must blend at the seams to create a smooth 3D environment look
  - Panorama image wrapped inside a sphere
    - a 360 panorama image naturally wraps smoothly around the inside of a sphere

source: freepik.com

Background - Unity

- Unity is a game engine that allows adding objects and lighting to a scene

- Skyboxes are supported in Unity



Experiments

In this section:

- Fine-tuning a Pre-trained Model
- Unconditional to conditional transfer
  - Hyper-modulation
  - Direct weight transfer
- Quantization

Experiments

Environment:

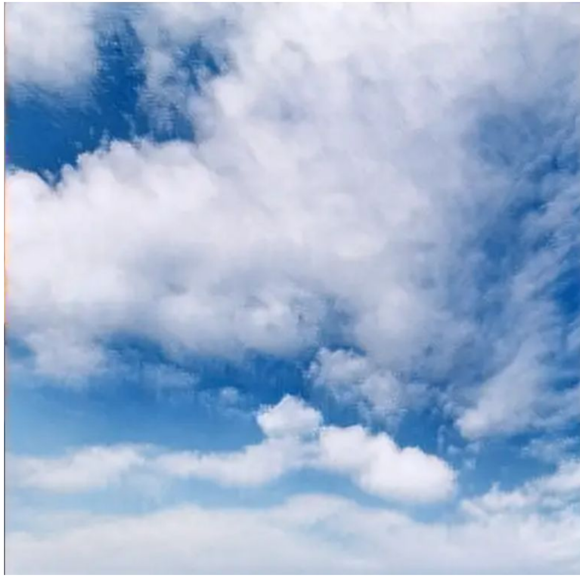
- Most experiments were performed in a conda virtual environment with Python 3.9.20 and PyTorch 1.7.1+cu102
- Hyper-modulation experiment was performed in a conda virtual environment with Python 3.8.8 and PyTorch 1.9.1+cu102.
- All model training was done with two K40m GPUs

Experiments - Fine-tuning a Pre-trained Model

- Goal: Obtain a model that generates sky images, given a pre-trained model of texture images
  - Achieve by fine-tuning the pre-trained texture image model
  - To compare training speed vs fine-tuning, also trained a second model from scratch
- Dataset used for fine-tuning: Cirrus Cumulus Stratus Nimbus (CCSN) dataset of 2543 cloud images sized 512x512 pixels
- Fine-tuned for 100 “kimg”, where “kimg” is defined as “thousands of real images shown to the discriminator”
- The total training time was approximately twenty hours
- Results on the next slides

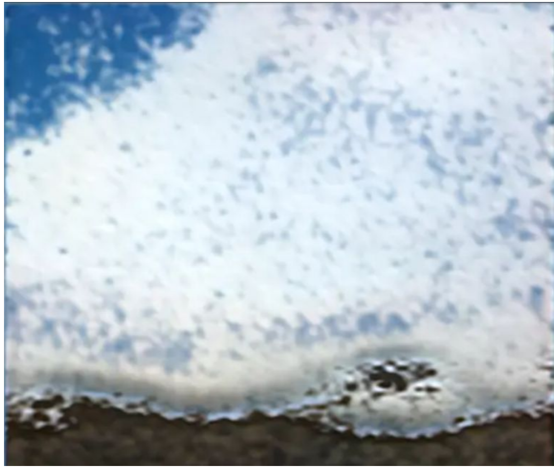
Experiments - Fine-tuning a Pre-trained Model

Fine-tuned
(100 kimg)



Experiments - Fine-tuning a Pre-trained Model

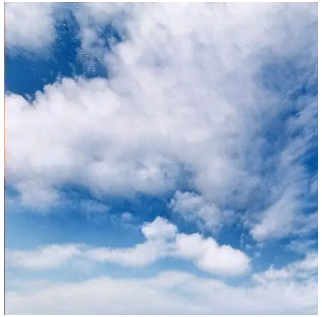
Trained from scratch (100 kimg)



Experiments - Fine-tuning a Pre-trained Model

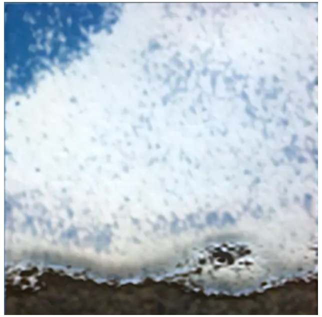
- Results (FID Scores)

100-kimg image samples



FID scores per 20 kimg of fine-tuned GAN

kimg	FID
0	265.07
20	62.21
40	35.25
60	28.23
80	27.16
100	25.55



FID scores per 20 kimg of GAN trained from scratch

kimg	FID
0	331.11
20	395.83
40	250.42
60	214.35
80	173.39
100	198.09

Experiments - Fine-tuning a Pre-trained Model

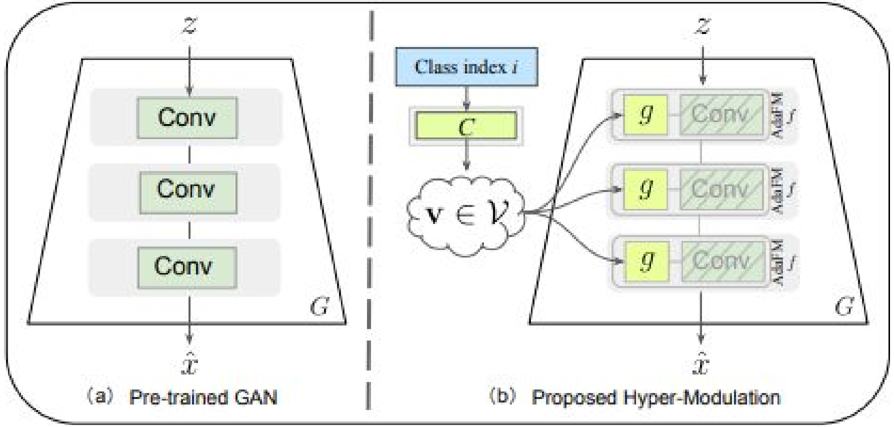
- Overall: fine-tuning is efficient compared to training from scratch
    - Experiment showed that it is viable to use a pre-trained model to help train the final model for generating a skybox

Experiments - Unconditional-to-conditional Transfer

- Why: because most pre-trained GAN models are unconditional. So, transfer from unconditional GAN is especially relevant.
- 1. Hyper-modulation
    - Hyper-modulation implementation was built in StyleGAN
        - could not easily apply to final model (StyleGAN2-ADA)
        - still useful for seeing unconditional-to-conditional transfer in action

Experiments - Unconditional-to-conditional Transfer

- Main idea of hyper-modulation: utilize a hypernetwork (generator that aims to generate parameters for other models) to generate the weights for all classes
  - Hypernetwork takes in a given source model weight along with a class embedding and outputs the desired target weight
  - This process is done in real time during training



- class embedding + source weight fed into the modulator, g, which will produce target weight used for class-specific generation

Experiments - Unconditional-to-conditional Transfer

- We observe the effect of hyper-modulation by training the base unconditional model on the AFHQ dataset (containing 15,000 512x512 images of animal faces divided into 3 classes) for ~38,000 iterations
  - Training duration: 57 hours

Experiments - Unconditional-to-conditional Transfer

- Latent interpolation of classes from 'cat' to 'wild' to 'dog'
- Animal classes are clearly distinguishable, showing success in transferring from unconditional to conditional GAN
  - some artifacts in the lower images; perhaps related to StyleGAN issue
- While effective, this method involves modification of the StyleGAN architecture
  - simpler method may be preferred as rewriting a model to implement unconditional to conditional knowledge transfer may not always be feasible

Experiments - Unconditional-to-conditional Transfer

- 2. Direct weight transfer
  - Motivation: In a paper on GAN transfer, Wang et al. demonstrated that simply initializing the weights of a conditional GAN by “copying the values from the unconditional GAN” is sufficient to improve model training
  - Taking inspiration from this, we conduct an experiment where we transfer the desired weights from the pre-trained model generator into the generator of an untrained conditional model
    - Then, the conditional model will continue training after receiving the new weights (like fine-tuning)

Experiments - Unconditional-to-conditional Transfer

- Steps:
  - Obtain fresh conditional StyleGAN2-ADA model
  - Obtain pre-trained unconditional StyleGAN2-ADA model
  - Extract weights from source pre-trained generator
    - Unconditional model’s state_dict (containing weights and layer mappings) does not exactly match the state_dict of the target conditional generator. To bypass this issue, non-matching keys are excluded when transferring pre-trained weights into filtered_dict.

```
target_dict = unpickled_textures_finetuned['G'].state_dict() # dictionary of target values

# filter out all keys starting with 'mapping'
filtered_dict = {key: val for key, val in target_dict.items() if not key.startswith("mapping")}
```

Experiments - Unconditional-to-conditional Transfer

- Steps:
    - Transfer weights by loading state_dict that contains pre-trained unconditional weights into the conditional generator

```
target_dict = unpickled_textures_finetuned['G'].state_dict() # dictionary of target values

# filter out all keys starting with 'mapping'
filtered_dict = {key: val for key, val in target_dict.items() if not key.startswith("mapping")}

# load the new state dict
unpickled_cond_pkl['G'].load_state_dict(filtered_dict, strict=False)

# load new state dict for 'G_ema' too
unpickled_cond_pkl['G_ema'].load_state_dict(filtered_dict, strict=False)
```

Experiments - Unconditional-to-conditional Transfer

- Steps:
  - Before and after weight transfer:

```
print(unpickled_cond_pkl['G'].state_dict()['synthesis.b4.conv1.weight'])

tensor([[[[ 1.8962e+00, -3.8448e-01, -1.7691e+00],
          [-7.9470e-01, -1.0124e+00, -3.7435e-01],
          [-3.6861e-01, -3.1082e-01, -7.8457e-02]],
```

```
print(unpickled_cond_pkl['G'].state_dict()['synthesis.b4.conv1.weight'])

tensor([[[[ 1.5431,  2.7893, -0.4025],
          [-1.1014, -2.9943,  3.2309],
          [-0.9261, -0.5723,  0.2979]],
```

Experiments - Unconditional-to-conditional Transfer

- Next, save the generator (now containing transferred weights) inside a complete model in a .pkl file
    - Resume training as if fine-tuning a conditional model normally

Create two more models as an experiment:

- We create another model which contains transferred weights from both the pre-trained unconditional generator and the discriminator
- Trained third conditional model from scratch (base case)

Experiments - Unconditional-to-conditional Transfer

- Training for all three models is performed for 100 kimg
- Results (left to right): G only (FID 196.47), G & D (FID 29.01), trained-from-scratch (FID 134.51)

- Best model is obtained from having both G & D weights transferred

Experiments - Unconditional-to-conditional Transfer

- Transferring only generator weights hurts the GAN's training performance, compared to training from scratch
  - Reasonable considering GAN training: ideally, generator and discriminator train at the same rate in order for both to learn well

Experiments - Unconditional-to-conditional Transfer

- Note: in this experiment, there is an untested case of transferring the discriminator weights only
  - Wang et al. (2018) suggested that transferring only discriminator weights produces inferior results, compared to transferring weights of both the generator and the discriminator, which also produced the best results in their research
  - With satisfactory results already obtained from transferring both of the generator and discriminator weights, the discriminator-only transfer case was deemed unnecessary

Experiments - Quantization

- Converted model into ONNX format before applying quantization
  - Used ONNX Neural Compressor tool to quantize the ONNX model using **weight-only quantization** (specifying n_bits = 4 bits) with **round-to-nearest (RTN) algorithm**

```
from onnx_neural_compressor.quantization import matmul_nbits_quantizer

algo_config = matmul_nbits_quantizer.RTNWeightOnlyQuantConfig()
quant = matmul_nbits_quantizer.MatMulNBitsQuantizer(
    onnx_model,
    n_bits=4,
    block_size=32,
    is_symmetric=True,
    algo_config=algo_config,
)
quant.process()
best_model = quant.model
```

Experiments - Quantization

- After quantization, model size shrunk slightly from 120 megabytes to 117 megabytes
- Inference speed was roughly timed on Google Colab using time() function. Over several runs, there seemed to be no significant difference in inference speed between the original and quantized models

116,786 KB

120,261 KB

time: quantized
time2: unquantized

```
time:  2.5210667209998974
time2:  2.5082910640001046
```

```
time:  2.4771116460001394
time2:  2.490175962999956
```

Experiments - Quantization

- It may be noted that the quantization tool's use was largely promoted for Large Language Models (LLMs)
- With this in mind, we tested a pre-trained LLM model "bigbird_Opset16.onnx" obtained from an ONNX model zoo (https://github.com/onnx/models)
  - Original model size: 498,094 kb
  - Quantized model size: 486,563 kb
    - Suggests that, even for LLMs, this tool does not reduce model size significantly with round-to-nearest weight-only quantization

Experiments - Quantization

- With the weight-only RTN quantization not effective for either StyleGAN2-ADA or LLM, we tried another quantization tool: ONNX quantize_dynamic (provided by ONNX Runtime)
- Used same StyleGAN2-ADA model converted to ONNX

```
from onnxruntime.quantization import quantize_dynamic, QuantType

model_fp32 = model_name
model_preprocessed = "preprocessed_onnx_model.onnx"
model_quant = 'my_quantized_onnx_model.onnx'
```

```
# quantize model, trying dynamic
quantized_model = quantize_dynamic(model_fp32, model_quant, weight_type=QuantType.QUInt8)
```

Experiments - Quantization

- Result: model size changed even less, from 120,261 kb to 120,058 kb
- However, noticed a warning stating the model opset does not support node fusions, thus leading to not as optimized performance

```
WARNING:root:The original model opset version is 10, which does not support node fusions. Please update the model to opset >= 11 for better performanc
```

- Issue: StyleGAN2-ADA model conversion fails when using any model opset > 10

```
SymbolicValueError: Unsupported: ONNX export of convolution for kernel of unknown shape.  [Caused by the value 'x.55 defined in (%x.55 : Float(*, *, *, *, strides=[32768, 64, 8, 1], requires_grad=0, device=cpu) = onnx::Reshape(%x.51, %710), scope: torch_utils.persistence.persistent_class.
```

```
# export as onnx file
torch.onnx.export(model=GG,
                  args=(dummy_input,label),
                  f="your_onnx_model_opset_11.onnx",
                  input_names=in_names,
                  output_names=out_names,
                  verbose=True,
                  opset_version=10,
                  export_params=True,
                  do_constant_folding=False,
                  use_external_data_format=False,
                  operator_export_type=torch.onnx.OperatorExportTypes.ONNX)
```

Conclusion: ONNX Runtime quantization may not work on ONNX-converted StyleGAN2-ADA model

conversion code of StyleGAN2-ADA to ONNX; uses opset_version=10

Experiments - Quantization

- When testing ONNX Runtime dynamic quantization on LLM, model size was reduced greatly from 498,094 kb to 125,161 kb
  - Much better result than using RTN weight-only quantization provided by ONNX neural compressor
  - Unfortunately, ONNX-converted StyleGAN2-ADA model cannot make use of the ONNX Runtime quantization
    - We keep the StyleGAN2-ADA model quantized using ONNX neural compressor

125,161 KB

498,094 KB

Quantized LLM model size (top)
Original LLM model size (bottom)

Web Page Deployment

StyleGAN2-ADA Model Skybox Generator

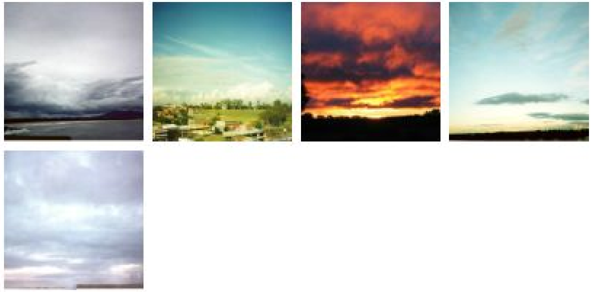
Select a cloud type (i.e. Stratocumulus), then click the button to generate.

Click on the image to download.

1-Stratocumulus ⌄ | Generate Image

Done! Preview the skybox in 3D by clicking the grey icon with four outward-pointing arrows near the bottom of the page.

1-Stratocumulus



2-Altocumulus

Web Page Deployment

- Deployed to a web page through ONNX Runtime Web, a Javascript library that enables ONNX model web application deployment
  - Output data obtained from model inference is accessed through Javascript
  - Class selection for the conditional model is done through a dropdown menu interface that offers a selection of available cloud types
  - To give an idea of what each selection might generate, sample labeled images are displayed on the right side column
  - After a cloud type is selected, the user can click the “generate image” button to start model inference

Web Page Deployment

- The model's generated output consists of numerical data that must be converted into pixel values
  - Through a conversion formula, the output array values are converted into red, green, and blue (RGB) values, which are used to display the resulting image on an HTML Canvas

Web Page Deployment

```
// first, create a new ImageData to contain our pixels
const image_resolution = 512;
var imgData = ctx.createImageData(image_resolution, image_resolution); // width x
height

// get data pointer
const data = imgData.data;

// assign color info
var offsetD = 0;
const factorR = image_resolution*image_resolution;
const factorB = image_resolution*image_resolution*2;
for (var i = 0; i < (image_resolution*image_resolution); i++) {
    data[offsetD]     = (dataC.data[i]*127.5)+128; // green value (0-255)
    data[offsetD + 1] = (dataC.data[i + factorR]*127.5)+128; // blue value (0-255)
    data[offsetD + 2] = (dataC.data[i + factorB]*127.5)+128; // red value (0-255)
    data[offsetD + 3] = 255;
    offsetD += 4;
}

// fill canvas
ctx.putImageData(imgData, 0, 0);
} catch (e) {
    document.write(`failed to inference ONNX model: ${e}.`);
}
```

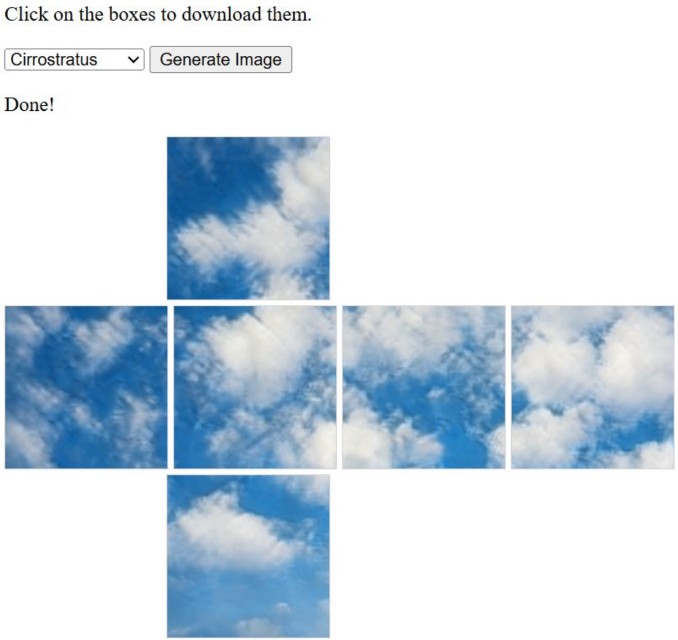
Conversion formula from model output to RGB pixel values

And code to put the color pixels onto the HTML Canvas context (ctx)

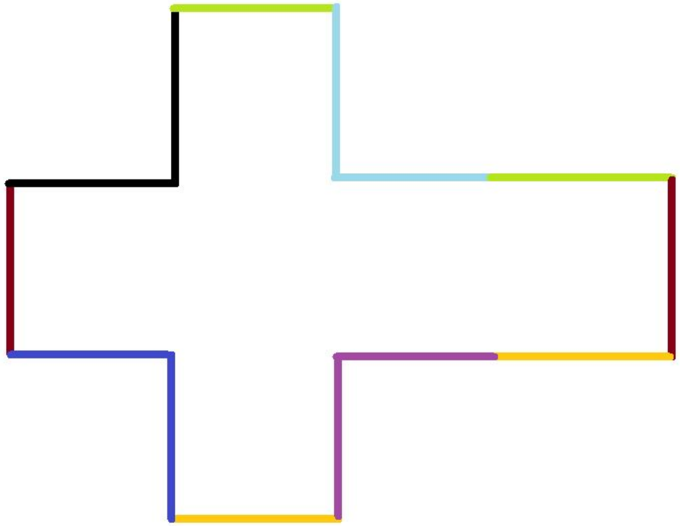
Web Page Deployment

- The displayed image, being a two-dimensional flat image, must be modified before it can be used as a skybox (since a skybox is typically created from either a 6-sided texture or a panoramic image)
- First attempt was with a 6-sided texture:

  - 6-sided texture did not work because the seams must be blended together, ideally with image editing software



Web Page Deployment



Above: the seams that must be blended together

Click on the boxes to download them.

Cirrostratus | Generate Image

Done!

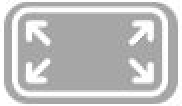
Web Page Deployment

Above: Unity preview of poor quality skybox with manually blended seams using simpl
gradients (to simulate HTML Canvas blending)

Web Page Deployment

- Since obtaining a seamless 6-sided texture from a 2D image did not seem feasible, we chose to use panoramic images for the skybox
- A panoramic image must wrap smoothly from left to right. To achieve this, one half of the image was taken and stitched against the same half but flipped, resulting in an image where the image wraps seamlessly from left to right
- This resulting panorama image can be downloaded by clicking on it.

- A 3D preview of the generated skybox can be viewed directly on the webpage by clicking on the A-Frame icon.



Results

See demo using the website, the website’s 3d preview, and then the skybox in Unity

Conclusion

- We trained a conditional StyleGAN2-ADA model using transferred weights from a pre-trained unconditional generator and discriminator
  - achieved an FID score of 29.01 after 100 kimg of training, outperforming the FID score of 134.51 obtained from the model trained from scratch for the same duration
- Next, we converted the model into ONNX format and quantized the model using weight-only round-to-nearest number (RTN) quantization provided by the ONNX Neural Compressor tool
  - Resulted in file size reduction of 3 megabytes down from 120 megabytes.
  - No noticeable inference speedup when timed using time() function

Conclusion

- Finally, we deployed the final model to a webpage which converts the model’s generated output into a panoramic-like image
  - Image can be downloaded and used to create a skybox
  - Webpage fulfills a use case of choice-based content generation as opposed to the already existing text-based prompt AI skybox generators

Future Work

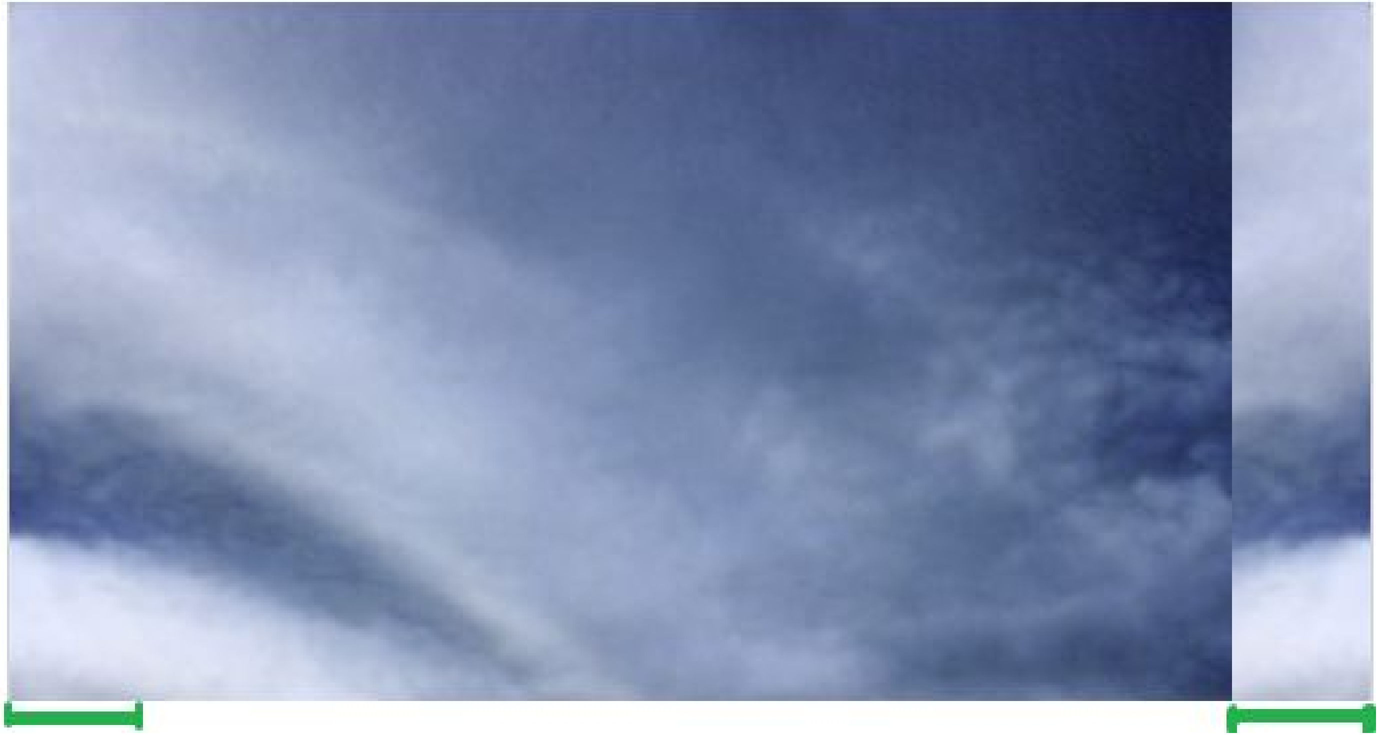
- Quantization
  - measure beyond model size; check resource usage of the quantized model using a tool that analyzes CPU or GPU usage
  - measure inference time using a tool instead of using the convenient but unreliable time() method

Future Work

- Adopt new method of converting 2D image to panorama image
    - current method of taking one image half and stitching it with the flipped image half wastes unique pixels on the right half of the image
    - Ideally, only a smaller stripe of the image is taken from one side, flipped, and blended in with the other side (but this is hard to do in HTML Canvas)

Converted panorama (left) versus original image (right)

Future Work



Ideal converted panorama image (pre-blended)

Future Work

- Combine generator model with an image upscaling model
    - Reduces problem of model's generated images appearing blurry (due to originally blurry dataset images and small 512x512 image resolution)
    - Possible to train a new model on higher resolution images, but training time grows as resolution increases
    - Therefore, instead of attempting to train a model on higher resolution images, can combine current model with a second ready-to-use model that upscales the image, which will produce a higher quality skybox

Future Work

Example upscaled image:

References

Pfau, J., Smeddinck, J. D., & Malaka, R. (2020, November). The case for usable ai: What industry professionals make of academic ai in video games. In *Extended abstracts of the 2020 annual symposium on computer-human interaction in play* (pp. 330-334).

Bengesi, S., El-Sayed, H., Sarker, M. K., Houkpati, Y., Irungu, J., & Oladunni, T. (2024). Advancements in Generative AI: A Comprehensive Review of GANs, GPT, Autoencoders, Diffusion Model, and Transformers. *IEEe Access*.

Sample generated images

8 - stratus