SWIMNGEN - a general progressive dataset augmentation model

Jowan Fromentin supervised by Roderick Mackenzie

***Abstract*—The SWIMNGEN model and a dataset augmentation algorithm with Python code has been created and is available on the SWIMNGEN GitHub repository. The SWIMNGEN model theoretically allows near limitless sampling from the SWIMSEG and SWINSEG datasets as the underlying image distributions have been learnt via SWIMGEN's generative adversarial network (GAN) during training on the Durham NVIDIA NCC network. The datasets have been augmented to give further samples for training sky image segmentation models. Inspection of the SWIMGEN components show that it is well trained and only a small section lead to a poor Frechet inception distance of 268 and inception score of 5. The metrics will improve with further training.**

I. Motivation

A. Photo-voltaic forecast

As we move to de-carbonise our energy, solar power becomes crucial for clean future energy generation. The power generation from solar farms is directly related to the solar irradiance - hence it is important that cloud cover is understood and can be predicted [1]. Accurate sky-image segmentation techniques can classify image pixels as cloud or sky. The implementation of these facilitate accurate forecasts of solar power generation, hence integration with stable energy grids [2].

B. Sky-image segmentation

Originally cloud segmentation and annotation was done by hand, this is still the most attractive annotation as it removes algorithmic biases. However, this is time consuming and inefficient, inspiring newer automated approaches to segmentation.

1) Colour-based: In 2017 Dev presented a supervised approach [3], using an analysis of different combinations of RGB colour components, not needing any manually defined parameters. This was trained on the SWIMSEG [3] and HYTA dataset [4], [5], by defining colour dimensions and performing principal component analysis (PCA), to find the most important for classification. Regression is performed on the principal components to create a potential field that is then thresholded to create the binary cloud map. The threshold was decided by maximising the area under the Receiving Operating Characteristics (ROC) curve. A limitation of this technique is that each pixel is considered individually, not allowing context from neighbouring pixels.

2) Convolutional Neural Networks (CNN): Shelhamer introduced a fully convolutional network approach for semantic segmentation in [6]. In 2019 Dev proposed a convolutional approach to nychthemeron (24 hours) cloud image segmentation [7], CloudSegNet. The architecture is typical of a Convolutional Neural Network (CNN), having an encoder block which convolves and downscales the image into latent space and a decoder block that convolves and upscales the latent space into a cloud 'probability mask'. Latent space is an abstract and multi-dimensional space that that cannot directly be interpreted, but embeds. The probability mask is approached with the method in Section I-B1. A threshold is found for the probability mask by maximising the area under the ROC curve for the training dataset, this is outlined in [3].

In 2021, [8] used machine learning methods to predict the photovoltaic solar energy produced from cloud cover. This compared different machine learning architectures ability to segment sky-images and commented on the relationship between cloud cover and photovoltaic production. The paper compared the colour-based segmentation discussed in [3] and semantic segmentation neural networks. Semantic image segmentation has an automatic advantage over single pixel colour-based algorithms as the context of neighbouring pixels is considered. The semantic networks discussed:

a) Fully convolutional: This is the architecture used by CloudSegNet, with an encoder and decoder block of convolutional layers.

b) U-Net: U-nets were first introduced in [9] for biomedical image segmentation. It is a convolutional architecture with skip connections between the encoder and decoder block, allowing the network to consider both coarse and fine features at different levels of the network. The networks focus is its adaptability to augmented data.

c) DeepLab: DeepLab is a deep learning approach to semantic image segmentation [10]. It utilises atrous convolution [11] to control the resolution that feature responses are computed at in a Deep CNN (DCNN). This enlarges the field of view of filters, increasing filter context. To segment at multiple scales, DeepLab uses an Atrous Spatial Pyramid Pooling technique and uses a Conditional Random Field to improve the localisation of boundaries.

d) Ensemble method: Multiple algorithms are used in ensemble learning to improve the predictive skill of the meta-algorithm beyond that of any constituent algorithm [12]. AdaBoost consists of 'weak' learners, being other traditional classification algorithms. The output of the weak learners are compiled into a weighted sum, this approach is often used for binary classification.

These architectures were tested on the SWIMSEG and HYTA datasets. For image segmentaion, TP, FP, TN and FN represent true posistives, false positives, true negatives and false posistives of the binary pixel classification respectively. Intersection over union is given by (IoU)$=\mathrm{TP}/(\mathrm{TP}+\mathrm{FP}+\mathrm{FN})$, average precision (AP)$=\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$ and average recall (AR)$=\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$. Table I displays the mean Intersection over Union (mIoU), mean AveragePrecision (mAP) and the mean Average Recall (mAR). The AdaBoost architecture was trained on the class labels rather than the cloud probability. We see that all models have have a high mAP. However, many

TABLE I
A TABLE SHOWING THE MEAN INTERSECTION OVER UNION, AVERAGE PRECISION AND AVERAGE RECALL OF DIFFERENT SEGMENTATION ARCHITECTURES [8].

Model	mIoU	mAP	mAR
PLS	0.6467	0.8961	0.6991
FCN	0.5649	0.8974	0.6040
U-Net	0.7626	0.9869	0.7703
DeepLab	0.5335	0.9234	0.5582
AdaBoost (class)	0.6128	0.8494	0.6875

struggle with mAR, limiting their mIoU. The U-Net model performed the best with the highest mIoU.

C. Datasets

One of the main challenges to sky-image segmentation is the lack of large and diverse image sets. 72 open-source ground-based sky-image datasets have been collated in [13], this is the first large-scale attempt to unify image resources for sky-image segmentation and solar prediction. The datasets are subjectively ranked in terms of the different qualities of each dataset, including: quality control, temporal and spatial coverage as well as temporal and image resolution. Each dataset is also annotated with its potential applications: cloud motion prediction, cloud segmentation, cloud classification and solar forecasting. Whilst welcomed by the community, further, augmented datasets can help improve models for all tasks. This could be done with a general algorithm to augment data.

D. Data augmentation

Despite the attempt to collate sky-image datasets it is still important to augment the datasets to avoid overfitting of a segmentation network. The aim of augmentation is to increase the number of unique images in a dataset.

1) Traditional: Traditionally, image dataset augmentation has relied on simple image manipulation techniques. This includes geometric transformations such as, cropping, flipping, and rotating images. Other common techniques include applying a kernel filter to sharpen or blur existing images and photometric or colour space transformations [14]. Colour space transformations can include colour casting where you increase the RGB values by a constant amount or vignetting. These techniques are simple to implement and effectively increase the number of unique images in a dataset. However, these variations from the original dataset are limited.

2) Machine learning: ML techniques including neural style transfer and generative models have been applied to augmentation tasks with great success. Neural style transfer combines a content image and a style reference image and 'blends' them together [15]. This can be applied to sky-image augmentation for example by converting an image from day to night or vice versa. Generative networks are trained on the existing and 'traditionally' augmented data to create artificial or synthetic instances that are plausibly from that dataset. Common generative networks to perform this are Variational Autoencoders (VAEs). VAEs learn the datasets translation to latent space and how to reverse it. But more popular are Generative Adversarial Networks (GANs) [16]. GANs are an adversarial network with a generator and discriminator [17]. The generator attempts to make synthetic examples from a random input to fool the discriminator, and the discriminator gives 'feedback', improving the generator, making the generators future attempts more believable. After sufficient training, the generated samples are of good enough quality to be considered augmented data [18].

II. GENERATIVE ADVERSARIAL NETWORKS

The use of GANs is an increasingly popular and broad subject as different network architectures are explored. The dataset augmentation task is aided by considering and combining different network architectures to create a bespoke model for our task. GANs take a vector of normally distributed random numbers called a latent vector, which is passed through the model to create an output unique to that vector.

A. Augmentation architectures

1) Conditional: A class label was proposed in [19], allowing control of the type of image generated, making a Conditional GAN (cGAN). The class label gives a unique vector for each class with one hot encoding. One hot encoding creates a sparse vector with length equal to the number of classes. Each element refers to a class, only one element can be 'on' at a time and will have a value 1, while the rest have a value 0, hence the name 'one hot'. This vector is then reshaped and concatenated to the image (for a discriminator) or the latent vector (for the generator), giving control over the augmented data that is created. The cGAN can be extended for the discriminator to predict the class of the image rather than take it as an input, this is an Auxiliary Classifier GAN (AC-GAN) [20].

2) Progressive: GANs are useful for dataset augmentation. However, for high resolution photo-realistic synthetic examples, a large training set and a lot of training is required, making the process computationally expensive. [21] proposed a progressive GAN (PGAN) architecture that increases the output resolution of the GAN after a defined number of epochs. Low resolution GANs require less training to achieve suitable results. After the initial low-resolution architecture of the PGAN has been trained, a new higher resolution output layer for the generative network and input layer for the discriminative network are slowly faded in. This process is repeated until a satisfactory resolution is reached. This reduces the training required by breaking the process into steps, teaching each layer of the network about different scale features of the data.

3) Self-attention: Attention was first proposed in Bahdanau's 2014 paper, [22] as a mechanism to improve the image quality of image translation GANs, by allowing the generator to focus on sections of the image. This idea was extended with self-attention in Vaswani's 2017 work, [23]. Self-attention, creates 'long-range dependencies' and gives the full context of the generated image, rather than the local behaviour of convolutional processes.

4) Enhanced super resolution GAN: Enhanced super resolution GANs (ESRGAN) 'increase' the resolution of an input image. The high resolution detail is generated using an adversarial architecture. ESRGANs were first proposed in Wang's 2018 work, [24], outperforming the original super resolution GAN (SRGAN).

Due to the complexity of GAN architectures, the training time and memory requirements increase with output resolution. To reduce the training required to reach high resolutions the output of a lower resolution GAN can be the input to an ESRGAN that is trained on the same dataset, increasing the output resolution.

B. Failure modes

Training a GAN is a non-convex and difficult problem as both the generator and discriminator change. Mode collapse is a failure mechanism where multiple latent variables are mapped to the same dataspace representation, meaning a limited subset of the possible outputs are generated. Instability can cause failure, leaving GANs stuck in a sub optimal parameter state, causing poor quality outputs. These failure mechanisms can be caused by poor model architecture and hyper-parameter selection in training.

C. Training

Both the generator and discriminator are Deep Convolutional Neural Networks (DCNNs).

The goal of a GAN is to 'create' samples that are plausibly from a training set. The generator takes a vector, z from latent space with distribution p_z (normally distributed) and maps it to data space, $G(z;\theta_g)$ or $G(z) \sim p_g$, where p_g is said to be the distribution of the generator. This mapping is defined by the generator parameters, θ_g. The discriminator maps a sample from data space, x to a single scalar, $D(x;\theta_d)$ defined by the discriminator parameters, θ_d. The real samples, x come from an underlying probability distribution, p_{data}.

The discriminator aims to maximise the log-probability that it correctly identifies the real samples, x as real and the generated samples, $G(z)$ as fake:

$$\max_D \log(D(x)) + \log(1 - D(G(z))) \tag{1}$$

The generator aims to minimise the log-probability that the discriminator correctly identifies the generated samples:

$$\min_G \log(1 - D(G(z))) \tag{2}$$

However, this expression can saturate, leaving unsatisfactory gradients for training. An alternative is the equivalent statement of maximising the log-probability of the discriminator incorrectly identifying the generated samples:

$$\max_G \log(D(G(z))) \tag{3}$$

Collectively this can be considered a two-player adversarial zero-sum game satisfying Equation 4:

$$\min_G \max_D \{ V(D,G) = \mathbb{E}_{x \sim p_{data}}[\log(D(x))] + \mathbb{E}_{z \sim p_g}[\log(1 - D(G(z)))] \} \tag{4}$$

The discriminator aims to maximise Equation 4 which represents correct classification and the generator seeks to minimise it, representing fooling the discriminator. This leads the discriminator to be unable to differentiate between p_g and p_{data}:

$$D(x) = \frac{1}{2} \tag{5}$$

Applying a game theoretical approach, training GANs is finding a Nash equilibrium for a two player, non-cooperative game, [25]. It can be seen that this equilibrium (limit) is that $p_g(x) \rightarrow p_{data}(x)$, meaning the generator has successfully mimicked the true distribution.

GANs train by incrementally improving the parameters, θ_D, θ_G. This is achieved by minimising an adversarial loss or cost functions for the generator and discriminator. This is done via the back propagation algorithm, intuitively explained in [26]. There is an analogous set of equations for the back propagation algorithm through CNNs [27]. The model parameters are adjusted with the ADAM optimiser as it helps to avoid non-optimal local minima [28].

The discriminator output is between 0 and 1 which can be interpreted as the probability that an input is real. The discriminator loss function is given as the binary cross entropy between the discriminator output and the true label. The binary cross entropy (BCE) of a probability is equivalent to the negative expected log probability $-\mathbb{E}[\log(x)] = \text{BCE}(x,y)$. Where y are the true labels and BCE is given as:

$$\text{BCE}(x,y) = -\frac{1}{N}\sum_{i=0}^{N} y\log(x) + (1-y)\log(1-x) \tag{6}$$

1) Base GAN: The generator loss, L_G can be given by:

$$L_G = \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))] \tag{7}$$

This is equivalent to BCE with incorrect labels. The discriminator loss, L_D is given by:

$$L_D = -\mathbb{E}_{x \sim p_{data}}[\log(D(x))] - \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))] \tag{8}$$

2) ESRGAN: The ESRGAN uses a relativistic discriminator. A relativistic discriminator modifies the standard discriminator architecture to include an additional input alongside the initial discriminator output. This means it evaluates the probability that a real image, x_r is relatively more realistic than a fake image, x_f.

$$D_{Ra}(x_r, x_f) = \sigma(C(x_r) - \mathbb{E}_{x_f}[C(x_f)]) \tag{9}$$

Where σ is the softmax function, given by:

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{x_j \in \boldsymbol{x}} \exp(x_j)} \tag{10}$$

C is the output of the discriminator and $\mathbb{E}_{x_f}$ finds the mean with respect to the minibatch x_f. Fake samples are the output of the generator for an input image, $x_f = G(x_i)$.

The discriminator loss, L_D is given by:

$$L_D = -\mathbb{E}_{x_r}[\log(D_{Ra}(x_r, G(x_i)))] - \mathbb{E}_{G(x_i)}[\log(1 - D_{Ra}(G(x_i), x_r))] \quad (11)$$

The respective competing component of the generators loss function, L_G^{Ra} is given by:

$$L_G^{Ra} = -\mathbb{E}_{x_r}[\log(1 - D_{Ra}(x_r, G(x_i)))] - \mathbb{E}_{G(x_i)}[\log(D_{Ra}(G(x_i), x_r))] \quad (12)$$

L_{pixel} is the mean squared pixel-wise error - content loss:

$$L_{\text{pixel}} = \mathbb{E}_{x_i}||G(x_i) - y||_2^2 \quad (13)$$

for the high resolution, ground truth, y. The perception loss, L_{percep} is given by:

$$L_{\text{percep}} = \mathbb{E}_{x_i}||\text{VGG}(G(x_i)) - \text{VGG}(y)||_2^2 \quad (14)$$

Where VGG is a pre-trained DCNN model with its output layer removed, designed for classification, [29]. The perception loss aims to minimise the distance between VGG features before activation, for the ground truth and generated samples, providing stronger supervision compared to the SRGAN. The complete generator loss function, L_G is given as:

$$L_G = L_{\text{percep}} + \lambda L_G^{Ra} + \eta L_{\text{pixel}} \quad (15)$$

λ and η are hyper-parameter scaling factors.

III. Methodology

A. Dataset

The SWIMSEG and SWINSEG datasets are being augmented in this study. This contains 1013 daytime and 115 night-time 600×600 pixel sky-images with associated cloud maps respectively. The collective labelled datasets will be referred to as SWIMNSEG.

The initial SWIMNSEG datasets were manually augmented before training the GAN to increase the sample size. Care must be taken when augmenting data so that the properties of the data are maintained and that no artifacts are introduced. The dataset was increased fourfold by rotating each image and its map by each 90° iteration. It was decided that augmentations such as shearing would create properties that are not reflective of real sky-images and that should do not want reflected in the final dataset.

B. GAN architecture

The SWIMNSEG dataset will be augmented by training a custom GAN architecture, SWIMNGEN . SWIMNGEN is implemented in python with tensorflow, using tensorflow layers. SWIMNGEN comprises of two separate generative networks. A conditional progressive self-attentive GAN, BaseGAN 75, will take in a 128 length latent vector and create images with a low resolution of 75×75. The output of BaseGAN is then fed into three ESRGAN upscaling GANs, with a $\times 2$ scaling factor. The output from the upscaling GANs will be a 600×600 high resolution output image with cloud mask.

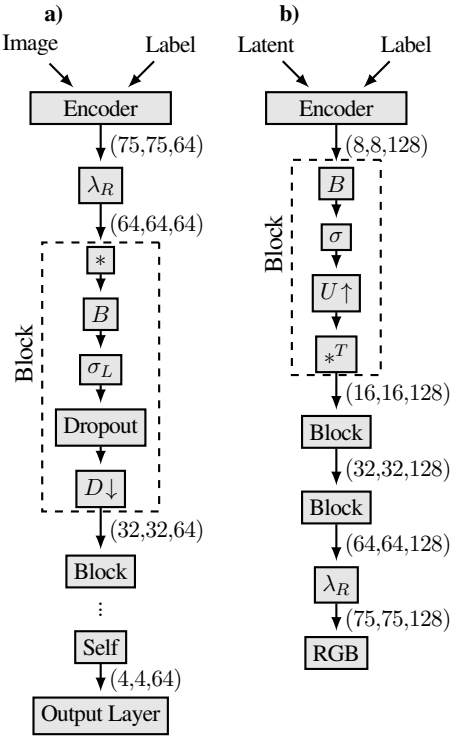

Fig. 1. General architecture of the a) discriminator and b) generator with their respective general block architecture. The layers are abbreviated: Convolution($*$), Transposed convolution($*^T$), Batch normalisation (B), ReLU(σ), LeakyReLU(σ_L), self attention(Self), downsampling($D\downarrow$), upsampling($U\uparrow$), Resizing layer(λ_R) and Dropout(`tensorflow.keras.layers.Dropout`)

The performance of SWIMNGEN with a BaseGAN 75 is compared with BaseGAN 32, which produces super low-resolution images of 32×32 before a custom ESRGAN upscales from $32 \times 32 \rightarrow 75 \times 75$ and is fed into the same upscaler. The BaseGAN and the upscaling GANs are trained separately to reduce training time and because both architectures have different loss mechanisms.

C. BaseGAN

BaseGAN will create novel, low-resolution sky-images. The BaseGAN model architecture is a combination of the techniques outlined in sections II-A1-II-A3.

Progressive GANs increase the output resolution after sufficient training by adding simple 'building' blocks. The architecture for the discriminative and generative network is shown in Figure 1. To reduce the training required to generate high-resolution images, BaseGAN is trained with a progressive algorithm using the growth and prune functions:

1) Growth: When the low resolution GAN has trained, a new block is added to increase the output resolution. The new blocks are faded in, in parallel to the upscaled old output. The parameter α is varied linearly with the training epoch.

$$\alpha = \frac{\text{epoch}}{\text{fade-in limit} - 1} \quad (16)$$

It is initialised with $\alpha = 0$, and is increased until $\alpha = 1$ when the epoch reaches the predefined fade-in limit.

When α is 0, the architecture is unchanged from the initial old model. Fading the new block avoids a step change and

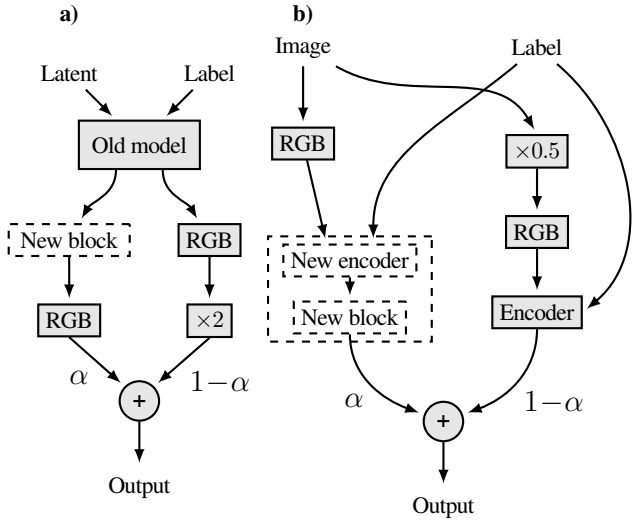

Fig. 2. Block diagram illustrating how the progressive architecture fades in blocks for the a) generator and b) discriminator

allows the new block to learn whilst it is being introduced. The one hot encoded class labels are learnt as the blocks are faded in. For the generator a new block is added after the old model, for the discriminator a new block must be added before the old model as shown in Figure 2.

2) Prune: When the new block has been faded in and $\alpha = 1$, the model is 'pruned', removing the 'old' branch.

D. Upscale GANs

The upscale GANs (ESRGANs) use a generative network to increase the resolution of the generated images. The ESRGAN consists of a generator and discriminator. The generator heavily utilises skip connections, as it is built from Residual-in-residual dense blocks (RRDB). Skip branches help avoid vanishing gradients which slow learning as described in [30]. Batch normalisation is not used in the generator as it is prone to creating artifacts [24]. The discriminator is a sequential convolutional model. The architectures are displayed in Figure 3, the model layers are abbreviated and their definitions are given in the figure caption.

E. Model layers

The SWIMNGEN model is built with layers from the `tensorflow.keras.layers` module, abbreviated to `tf`. Each layer performs an operation to the latent input tensors or 'feature maps' to get the output. The layers and their abbreviations for architecture visualisation are given:

1) ReLU & LeakyReLU, σ, σ_L: Activation functions are used in NNs and CNNs to introduce non-linearities, this allows them to 'learn' the complex, non-linear behaviours within the data. Traditionally the sigmoid function was used. However, due to the vanishing gradient which hindered training, the Rectified Linear Unit (ReLU) and its leaky counterpart (Leaky ReLU) are commonly used as they avoid this. The ReLU activation function, σ is given by:

$$\text{ReLU}(x) = \max(x, 0) \tag{17}$$

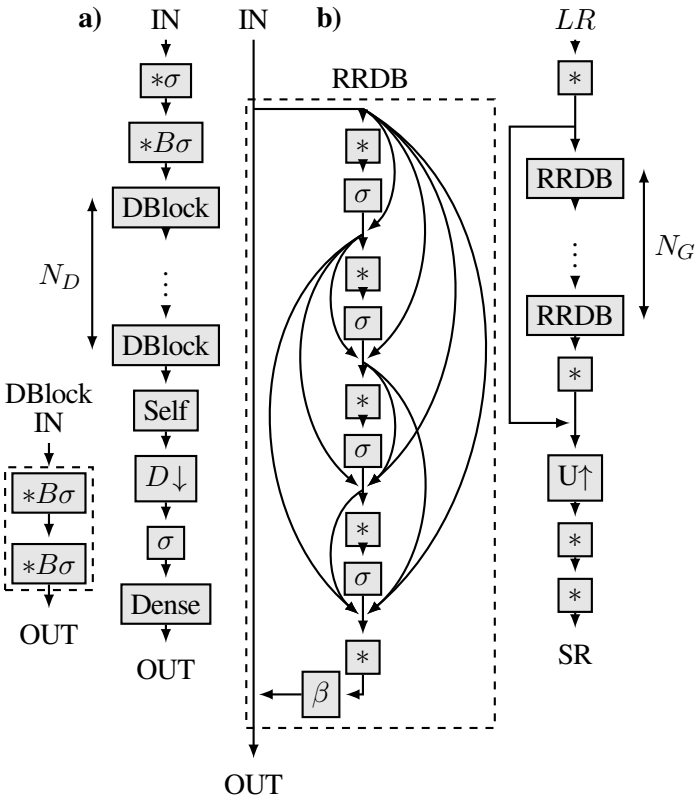

Fig. 3. Architecture of the ESRGAN. a) The discriminator architecture. b) The generator architecture, with skip connections. The layers are abbreviated: Convolution($*$), Batch normalisation (B), ReLU(σ), self attention(self), downsampling($D\downarrow$), scaling factor(β), upsampling($U\uparrow$), `tensorflow.keras.layers.Dense`(Dense).

Leaky ReLU, σ_L is given as:

$$\text{LeakyReLU}(x; a) = \max(x, ax) \tag{18}$$

where $a = 0.3$. This is implemented using `tf`'s, `ReLU` and `LeakyReLU` respectively.

2) Convolution, $*$: The convolution layers apply the following operation on input feature maps, $\boldsymbol{x}$:

$$\boldsymbol{y}_i = \sum_{\lambda}^{f^{l-1}} k_i * \boldsymbol{x}_\lambda + b_i \tag{19}$$

Where y_i and k_i is the ith output feature map and convolutional kernel respectively, f^{l-1} is the number of channels in the input feature maps, b is the bias, and $*$ is the 2 dimensional convolution operator. Convolution is applied between the kernels and input activations using the `tf` layers:

- `Conv2D` is used for BaseGAN discriminator and for the upscale GAN.
- `Conv2DTranspose` is used for the BaseGAN generator.

The output is padded with zeros to keep input and output dimensions consistent.

3) Encoder: Both the generator and discriminator of BaseGAN use an encoder block to format the input images and labels. The generator encoder concatenates the one hot labels and the latent vector before passing it through a dense layer (`tensorflow.keras.Layers.Dense`) and reshaping the output. The discriminator encoder passes the one hot label

through a dense layer and reshapes it to concatenate it to the input image.

4) Discriminator output layer: The output layer of the BaseGAN discriminator has an input shape, $(4,4,64)$ from the self-attention layer and has an output shape, (1). The output layer applies two convolution layers before flattening and summing the activations.

5) RGB layer: The RGB layer maps the feature maps to RGB values. It is a convolution layer with 3 $(1,1)$ kernels. This acts as a weighted sum of the latent activations for each channel. The tanh output activation is scaled using a `Lambda` layer, $\frac{1}{2}+\frac{1}{2}\tanh(x)\in[0,1]$ for $x\in\mathbb{R}$. This bounds the GAN output RGB output between $[0,1]$.

6) BaseGAN general block: The BaseGAN generator and discriminator blocks are displayed in Figure 1. The discriminator block contains a convolution layer, batch normalisation, Leaky ReLU, dropout and average pooling layers. [31] suggested that Leaky ReLU is most suitable for the discriminator.

The generator block comprises a batch normalisation, ReLU, up sampling and a transposed convolution layers.

The layers in the BaseGAN blocks are executed using:

- `Dropout`: Dropout is used to avoid overfitting in NNs and CNNs. A random fraction of the connections in a dropout layer will not 'conduct' at each instance during training. This requires the model to learn 'robust' features that do not depend on individual nodes. The dropout 'rate' was set to 0.4 in the discriminator.
- `AveragePooling`, $D\downarrow$: Average pooling reduces the dimension of the CNN latent activation tensors between discriminator blocks by splitting the tensor into $2\times2\times c$ sections and taking the mean in the first two dimensions. Max pooling (`MaxPooling`) is another common pooling method, where the pooled sections take the maximum activation in the region.
- `BatchNormalization`, B: Batch normalization (BN) increases the stability of training networks, normalising activations across feature maps in CNNs over a minibatch. BN uses Z-score normalisation, so the minibatch activations have a mean, μ_B of 0 and a standard deviation, σ_B of 1.

$$\mu_b=\frac{1}{m}\sum_{i=1}^{m}x_i \tag{20}$$

$$\sigma_B^2=\frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_B)^2 \tag{21}$$

Each dimension is normalised for the batch, $x=\left(x^{(1)},...,x^{(d)}\right)$

$$\hat{x}_i^{(k)}=\frac{x_i^{(k)}-\mu_B^{(k)}}{\sqrt{(\sigma_B^{(k)})^2+\epsilon}} \tag{22}$$

ϵ adds numerical stability. With hyper parameters, γ and β, the output is given by:

$$\begin{aligned} y_i^{(k)}&=\gamma^{(k)}\hat{x}_i^{(k)}+\beta^{(k)}\\ y^{(k)}&=\mathrm{BN}\left(x^{(k)};\gamma^{(k)},\beta^{(k)}\right)\end{aligned} \tag{23}$$

BN speeds up training and allows a wider range of learning rates, the reason is disputed. The first paper proposing BN, [32] claimed it reduced internal covariate shift. However, this was debunked in [33], which proposed that it helps training by reparametrizing the optimisation in terms of γ and β, making the problem more stable and smoother.

- `Upsampling`, $U\uparrow$: Up sampling is used to double the resolution of latent tensors in the generator blocks before the padded transposed convolution.

7) Weighted sum: The WeightedSum function is implemented with a custom layer with a variable hyperparameter, α. The weighted sum is given by:

$$\begin{aligned} y&=\mathrm{WeightedSum}(x_1,x_2,\alpha)\\ &=(1-\alpha)x_1+\alpha x_2\end{aligned} \tag{24}$$

8) Self-attention: Self-attention creates long range dependencies between features and is implemented with a custom self-attention layer. The process outlined and visualised in Appendix A and also in Zhang's, [27] . For input feature maps, $\boldsymbol{x}$, the output of the self-attention layer, $\boldsymbol{y}$ is given by:

$$\boldsymbol{y_i}=\gamma\boldsymbol{o_i}+\boldsymbol{x_i} \tag{25}$$

Where $\boldsymbol{o}$ is the self-attention feature maps and γ is a learnable scaling factor.

F. BaseGAN Algorithms

1) Architecture algorithm: The batch size is adjusted with image resolution to adhere to memory constraints, using the `epoch_batch_list` as a lookup table.

2) Total train: The process for training and growing BaseGAN is given in Algorithm 1.

3) Train function: The function `GAN_train` used in Algorithm 1 applies an optimiser to update the model parameters according to the gradients and losses and is given in Appendix B.

G. Upscale GAN algorithms

1) Total train: Due to the static size of the ESRGAN, its training algorithm is much simpler than the BaseGAN . The training algorithm for the upscale GANS are shown in Algorithm 2. The `PreTrain` function is similar to the `ESRGAN_train` function, without the discriminator and only considering the pixel loss.

2) Train function: The `ESRGAN_train` function updates model parameters and is given in Appendix B.

IV. Execution

A. Durham NVIDIA NCC

The models were trained on a GPU on the Durham NVIDIA CUDA Center (NCC) network. CUDA allows parallel computing, speeding up tensorflow applications. The python files were called using shell files on the undergraduate GPU partition. There is a 28GB RAM memory limit for each program using a GPU.

Algorithm 1: BaseGAN training algorithm

input : R_{init},R_{final}, `epoch_batch_list`

Initialise and compile the model
$\{G,\theta_G\},\{D,\theta_D\} \leftarrow$ `architecture`(R_{init})
Calculate resolution increases, $R\uparrow$
$R\uparrow \leftarrow \text{ceil}\left(\log_2 \frac{R_{\text{final}}}{R_{\text{init}}}\right)$
Set the batch
size, B_s and train and fade in epoch limits, E_T and E_F
E_T,E_F,$B_s \leftarrow$ `epoch_batch_list`(R_{init})
Retrive the dataset, $\boldsymbol{X}$
$\boldsymbol{X} \leftarrow$ `dataset`(R_{init},B_s)
for R_i **in** $R\uparrow$ **do**
  $\theta_G,\theta_D \leftarrow$ `GAN_train`$(\boldsymbol{X},E_T,$Fade_in=**`False`**$;\theta_G,\theta_D)$
  if $R_i < R\uparrow$ **then**
    Grow the GAN
    $\theta_G,\theta_D \leftarrow$ `grow` (θ_G,θ_D)
    set the resolution, R
    $R \leftarrow R_{\text{init}} \times 2^{R_i}$
  else
    Final GAN growth
    $\theta_G,\theta_D \leftarrow$ `final_growth`$(R_{\text{final}};\theta_G,\theta_D)$
    Final resolution
    $R \leftarrow R_{\text{final}}$
  end
  Set the batch and epoch size
  E_T,E_F,$B_s \leftarrow$ `epoch_batch_list`(R)
  Retrieve higher resolution dataset, $\boldsymbol{X}$
  $\boldsymbol{X} \leftarrow$ `dataset`(R,B_s)
  $\theta_G,\theta_D \leftarrow$ `GAN_train`$(\boldsymbol{X},E_F,$Fade_in=**`True`**$;\theta_G,\theta_D)$
  Prune the GAN
  $\theta_G,\theta_D \leftarrow$ `prune`(θ_G,θ_D)
end
$\theta_G,\theta_D \leftarrow$ `GAN_train`$(\boldsymbol{X},E_T,$Fade_in=**`False`**$;\theta_G,\theta_D)$
return $G(\theta_G)$

Algorithm 2: ESRGAN algorithm

input : $\boldsymbol{X}:\{\boldsymbol{x}_{LR},\boldsymbol{x}_{HR}\}$

Load initial parameters from config file
$\theta_G,\theta_D \leftarrow$ `config()`
Initialise and compile the generator
$G \leftarrow$ `generator`(θ_G)
Pre-train the generator with pixel loss
$\theta_G \leftarrow$ `PreTrain`$(\boldsymbol{x}_{LR},\boldsymbol{x}_{HR},\theta_G)$
Initialise and compile the discriminator
$D \leftarrow$ `discriminator`(θ_D)
Train the generator with all losses
$\theta_G,\theta_D \leftarrow$ `ESRGAN_train`$(\boldsymbol{x}_{HR},\boldsymbol{x}_{LR},\theta_G,\theta_D)$
return $G(\theta_G)$

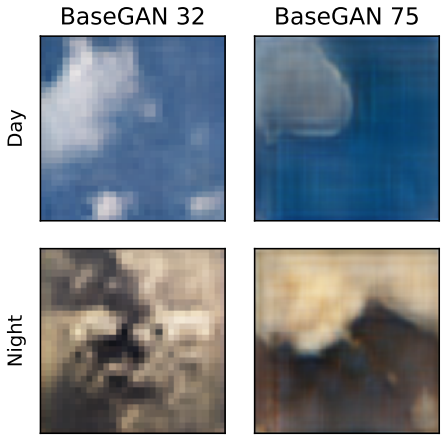

Fig. 4. Day and night samples from BaseGAN 32 and BaseGAN 75.

B. Model parameters

The hyper-parameters for SWIMNGEN during training are given.

1) Base GAN: The base GAN used a training rate of 3×10^{-4} with the ADAM optimiser for both the generator and discriminator. For the `GAN_train` function no noise was added at the input of the discriminator ($\theta_N = 0$) and no discriminator update threshold was used ($U_T = 0$). The base GAN started with a 32×32 resolution, R_{init} and performed Algorithm 1 to the 75×75 final resolution, R_{final}. The `epoch_batch_list` is given in Appendix B.

C. SWIMSEG_PLUS GitHub

The trained SWIMNGEN models with both BaseGAN 32 and BaseGAN 75 are available on the SWIMSEG_PLUS GitHub repository along with a sample of their respective augmented datasets. The SWIMNGEN repository also contains the code to sample images from a model, and code to train a new model.

1) Upscale GANs: The ESRGANs each had 16 generator RRDB ($N_G = 16$), 4 discriminator blocks ($N_D = 4$) and a LeakyReLU parameter, $a = 0.2$. The learning rate was 1×10^{-4} for the `PreTrain` function and was 5×10^{-5} for both generator and discriminator in the `ESRGAN_train` function.

V. Evaluation

A. Visual inspection

1) BaseGAN : Sample outputs of the base GANs are visualised in Figure 4. We see that BaseGAN 32 creates convincing low resolution samples. Whereas, BaseGAN 75 struggles to generate images that are not clearly synthetic to the human eye.

2) t-distributed stochastic neighbour embedding: t-distributed Stochastic Neighbour Embedding (t-SNE) plots, introduced in Van der Maaten and Hinton's 2008 work, [34] allow us to visualize high dimensional data in 2d. t-SNE creates joint probabilities between outputs and minimises the KL-divergence, attempting to find a 2D representation that best preserves the relative distance between high dimensional samples. t-SNE is a dependent on a perplexity parameter that is

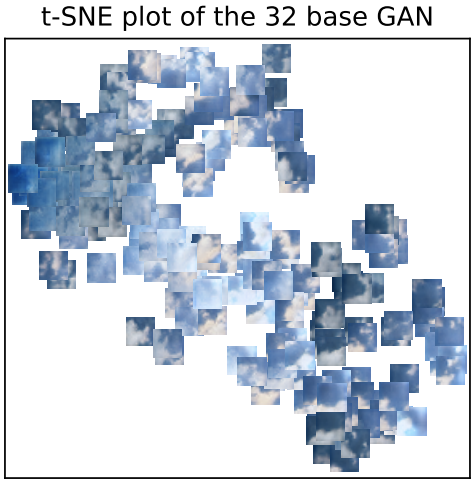

Fig. 5. t-SNE plot for the 32 base GAN with a perplexity 10.

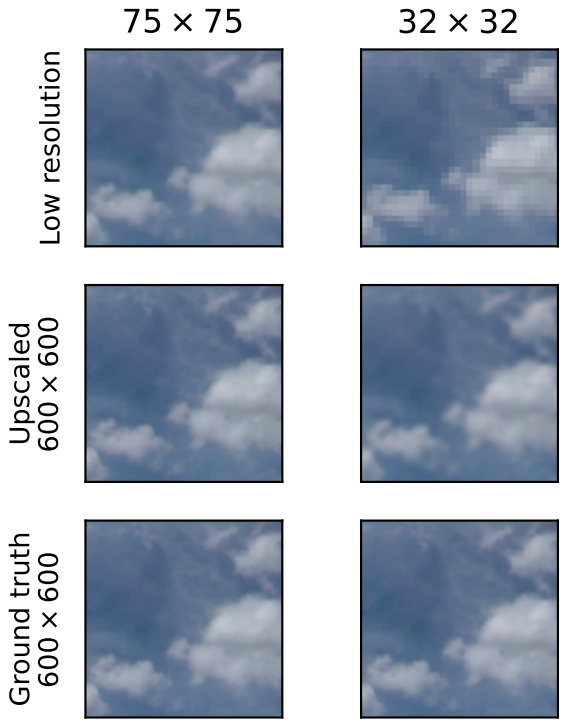

Fig. 6. The ESRGAN upscaling visualised on a 75×75 and 32×32 resolution image from SWIMSEG.

a trade of between preserving local and global structure in the 2d plot. A t-SNE plot of the 32 base GAN day class is displayed in Figure 5. t-SNE plots for the 32 base GAN the 75 base GAN are give in Appendix C. The t-SNE plot preserves latent space representations [34], meaning neighbouring points in the plot will have close latent input vectors. This allows us to see how the BaseGAN has learnt the ground truth dataset and can display the model's sample diversity. We see a wide range of sky scenes, implying that cloudGEN has learnt the distribution of the dataset. No inferences can be made between t-SNE plots. In Figure 5 we see that images that share features, such as image brightness and cloud position are clustered together. This implies that images with a certain type of cloud on the left for example are all sampled from a similar section of latent space.

3) Upscaling: The performance of the upscaler on a SWIMSEG sample is shown in Figure 6. This compares upsampling from a 32×32 and 75×75 resolution image. The upscaler does a good job at upscaling the 75×75 resolution image as it closely matches the ground truth with lots of detail. There is a clear reduction in image quality for the 32×32 image. This arrises from the extra upscaling layer to convert it to 75×75. This should be trained further to achieve a high quality output.

4) cloudGEN samples: Output samples for cloudGEN with both BaseGAN 32 and BaseGAN 75 are shown in Appendix D. With close inspection the samples are visibly synthetic. This is due to the unsatisfactory scaling between 32×32 and 75×75 resolution images for BaseGAN 32, and that BaseGAN 75 is yet to adequately model the SWIMNSEG dataset.

B. Metrics

There is no single objective method of evaluating the output quality of GANs and this is an important area of research [35]. The Frechet inception distance (FID) and inception score (IS) are some of the most common methods for creating evaluation metrics for the quality of GAN outputs [36]. For the evaluation of GANs, it is common to use a pre-trained network to act as a mapping from image space to either latent or label space. The Inception v3 network [37] is generally used as it has been trained on the diverse ImageNet dataset. The Inception v3 network takes images with 299×299 resolution, meaning the images must be resized before its quality can be assessed with Inception v3, limiting its use for high resolution images.

C. Inception score

The inception score (IS) of a generator G is given by:

$$\text{IS}(G) = \exp(\mathbb{E}_{\boldsymbol{x} \sim p_G} D_{KL}(p(y|\boldsymbol{x})||p(y))) \tag{26}$$

Where p_G is the distribution of the generator, $D_{KL}(p||q)$ is the KL-divergence between distributions p and q, $p(y|\boldsymbol{x})$ is the conditional class label distribution, mapped by the Inception v3 model. The marginal class label distribution, $p(y)$ is given by $p(y) = \int_{\boldsymbol{x}} p(y|x) p_{gen}(x) dx$. The inception score is limited between: IS $\in [1, N]$, where N is the number of classes in the classification model. Better generators will have a higher inception score.

However, the usefulness of IS is limited for our dataset, as there is no cloud class in the ImageNet database that Inception v3 was trained on.

D. Frechet inception distance

FID is preferential over IS as it compares the distribution of generated images with that of real images. The Frechet inception distance between two distributions, d_F is given by:

$$d_F(\mathcal{N}(\mu,\Sigma),\mathcal{N}(\mu',\Sigma'))^2 = ||\mu - \mu'||_2^2 + \text{tr}\left(\Sigma + \Sigma' - 2\left(\Sigma^{\frac{1}{2}} \cdot \Sigma' \cdot \Sigma^{\frac{1}{2}}\right)^{\frac{1}{2}}\right) \tag{27}$$

The multi-dimensional gaussians, $\mathcal{N}(\mu,\Sigma)$ and $\mathcal{N}(\mu',\Sigma')$ are found for an augmented dataset $\boldsymbol{X}$ and ground truth dataset

TABLE II
SUMMARY OF EVALUATION METRICS TO COMPARE THE 32 AND 75 BASE GAN, THE BEST RESULTS ARE BOLD.

	32 base		75 base	
	Mean	Std	Mean	Std
IS	**4.9659**	2.28e-4	4.9561	2.94e-4
FID	**267.75**	7.699	323.73	7.818

$\boldsymbol{X'}$ respectively. The mean and standard deviation parameters are extracted when the gaussian distributions are fitted to $f(\boldsymbol{X})$ and $f(\boldsymbol{X'})$. f is the Inception v3 model without the classification layer, mapping the datasets to latent space.

E. CloudSegNet

A sky-image segmentation model with the CloudSegNet's convolutional architecture has been created and trained on the augmented datasets. An indicator of the augmented datset's quality is determined by the segmentation model's performance segmenting the original SWIMNSEG dataset. The datasets are resized to a 300×300 resolution before training, as per the CloudSegNet paper, [7].

F. Evaluation metrics

For both the 32 and 75 resolution BaseGAN the mean values of inception score and Frechet inception distance are displayed in Table II, the best results are given in bold. The metric evaluation will give a different result each time, as it is evaluated with a random latent input vector. This is equivalent to us sampling from the generator distribution p_g, with the central limit theorem (CLT) repeating this evaluation will tend to the 'true' metrics. The IS was calculated for 10,000 cloudGEN samples 100 times and the FID was calculated for 20 samples (due to memory constraints) 600 times.

In Figure 7 we see that both base GANs IS distributions are narrow with low standard deviations, given in Table II. BaseGAN 32 has a higher mean than the BaseGAN 75, implying a better performance under this metric. However, this is only a slight increase in metric performance of 0.0098 between the means. Both IS metrics would be considered bad as the upper limit on the IS score is 1000 from the number of classes in the ImageNet dataset.

BaseGAN 32 also outperforms BaseGAN 75 in terms of FID. However, BaseGAN 32's mean FID score of 267.75 is considered poor. FID scores of modern generative networks achieve are an order of magnitude lower, in the tens [38].

The poor metric results for both IS and FID indicate room for improvement for cloudGEN and highlight the limitations of the commonly used Inception v3 network in our case, due to the lack of a cloud class label.

The performance of the 32 and 75 base GAN cloudGENs on training a segmentation architecture are displayed in Table III. Segmentation models were first trained on an augmented dataset, and the segmentation threshold was set by maximizing the area under the ROC curve. The model's performance on the augmented dataset is shown in the Table III, displaying

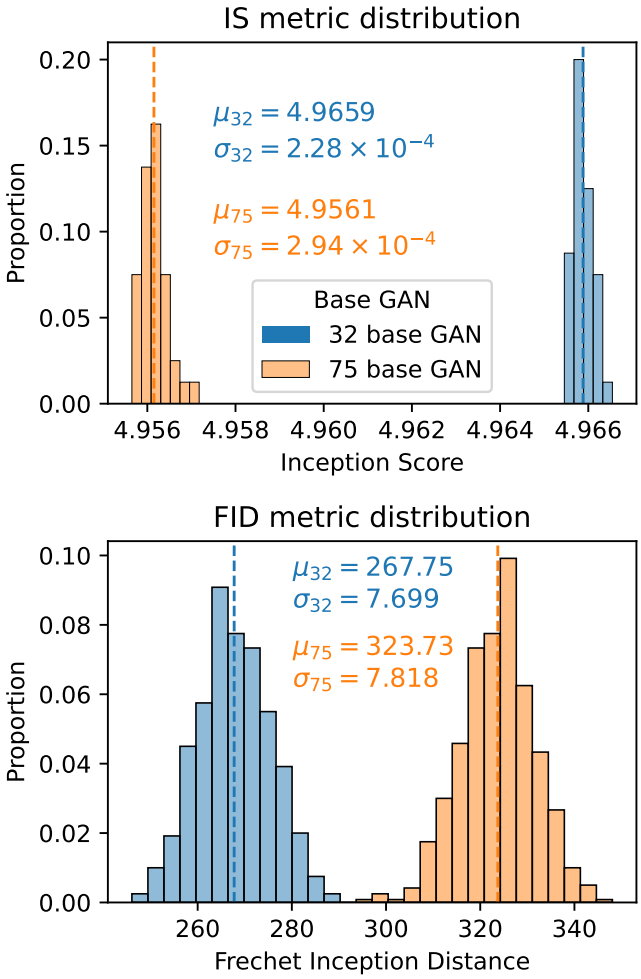

Fig. 7. Evaluation metrics, IS and FID for both the 32 and 75 base GAN with the metric distribution mean and standard deviations annotated.

TABLE III
RESULTS FROM A SEGMENTATION ARCHITECTURE TRAINED ON THE AUGMENTED DATASET.

		F1 score	Accuracy	Precision	Recall
32 base GAN	Augmented	0.93363	0.91018	0.95124	0.91666
	SWIMNSEG	0.66912	0.64985	0.68838	0.65092
75 base GAN	Augmented	0.94792	0.94130	0.95928	0.93682
	SWIMNSEG	0.75449	0.67473	0.64000	0.91886

that it was successfully trained. The segmentation model's performance was evaluated by predicting the cloud maps of the original SWIMNSEG datasets. The performance metrics for SWIMNSEG segmentation are poor relative to the augmented dataset. This indicates that the model did not fully learn the relationship between RGB sky images and their respective cloud mask when trained solely on the augmented dataset. However, it is clear that the augmented datasets have transferred useful information to the segmentation model and that, in tandem with ground truth data this technique can be used to further train models.

VI. CONCLUSION

A progressive algorithm for training a generative adversarial network to augment the SWIMSEG and SWINSEG datasets

was proposed and is available on the SWIMNGEN_PLUS GitHub. The augmented dataset sampled from SWIMNGEN can be used to improve the ability of sky-image segmentation models by increasing the training dataset size. The model requires further training but shows promising signs as both the 32×32 GAN and the $75 \rightarrow 600$ upscaler works well under visual inspection. The base GANs have learnt many modes of the ground truth dataset as a t-SNE plot shows the high sample diversity. Sample output evaluation metrics are poor, however this does not reflect the reality as only further training of a couple of sections are essential before we can expect good results.

The constituent parts of the cloudGEN model should be further trained to increase the quality of the output samples. This would be reflected in the output metrics. Focus should be on generating a high quality 75×75 image, either by training the BaseGAN 75 or by training the $32 \times 32 \rightarrow 75 \times 75$ upscaler due to the convincing ability of the BaseGAN 32 and the $75 \times 75 \rightarrow 600 \times 600$ upscaler.

A custom version of the inception network could be fine tuned by introducing a cloud class so that the evaluation metrics can better represent the sample quality. This process can be extended to augment further datasets, using code on the SWIMSEG_PLUS GitHub. Once properly trained on SWIMNSEG, applying transfer learning would rapidly speed up training for augmenting similar datasets such as HYTA.

REFERENCES

[1] Y. Nie, A. S. Zamzam, and A. Brandt, "Resampling and data augmentation for short-term PV output prediction based on an imbalanced sky images dataset using convolutional neural networks," *Solar Energy*, vol. 224, no. May, pp. 341–354, 2021.

[2] D. Yang, W. Li, G. M. Yagli, and D. Srinivasan, "Operational solar forecasting for grid integration: Standards, challenges, and outlook," *Solar Energy*, vol. 224, no. June, pp. 930–937, 2021.

[3] S. Dev, S. Member, Y. H. Lee, and S. Member, "Color-Based Segmentation of Sky / Cloud Images From Ground-Based Cameras," vol. 10, no. 1, pp. 231–242, 2017.

[4] Q. Li, W. Lu, and J. Yang, "A hybrid thresholding algorithm for cloud detection on ground-based color images," *Journal of atmospheric and oceanic technology*, vol. 28, no. 10, pp. 1286–1296, 2011.

[5] S. Dev, Y. H. Lee, and S. Winkler, "Multi-level semantic labeling of sky/cloud images," in *2015 IEEE International Conference on Image Processing (ICIP)*, pp. 636–640, IEEE, 2015.

[6] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, 2017.

[7] S. Dev, A. Nautiyal, Y. H. Lee, and S. Winkler, "CloudSegNet: A Deep Network for Nychthemeron Cloud Image Segmentation," *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 12, pp. 1814–1818, 2019.

[8] S. Park, Y. Kim, N. J. Ferrier, S. M. Collis, R. Sankaran, and P. H. Beckman, "Article prediction of solar irradiance and photovoltaic solar energy product based on cloud coverage estimation using machine learning methods," *Atmosphere*, vol. 12, no. 3, 2021.

[9] W. Weng and X. Zhu, "INet: Convolutional Networks for Biomedical Image Segmentation," *IEEE Access*, vol. 9, pp. 16591–16603, 2021.

[10] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[11] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," 2017.

[12] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[13] Y. Nie, X. Li, Q. Paletta, M. Aragon, A. Scott, and A. Brandt, *Open-Source Ground-based Sky Image Datasets for Very Short-term Solar Forecasting, Cloud Analysis and Modeling: A Comprehensive Survey*. No. November, 2022.

[14] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep Image: Scaling up Image Recognition," 2015.

[15] Y. Li, N. Wang, J. Liu, and X. Hou, "Demystifying neural style transfer," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 0, pp. 2230–2236, 2017.

[16] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 1, 2019.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, 2020.

[18] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification," *Neurocomputing*, vol. 321, pp. 321–331, 2018.

[19] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," pp. 1–7, 2014.

[20] J. Brownlee, "Generative Adversarial Networks with Python," *Machine Learning Mastery*, pp. 1–654, 2019.

[21] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–26, 2018.

[22] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[24] X. Wang, K. Yu, S. Wu, J. Gu, and Y. Liu, "ESRGAN : Enhanced Super-Resolution Generative Adversarial Networks," pp. 1–16.

[25] F. Farnia and A. Ozdaglar, "Do GANs always have Nash equilibria?," *37th International Conference on Machine Learning, ICML 2020*, vol. PartF168147-4, pp. 3010–3020, 2020.

[26] M. A. Nielsen, "Neural networks and deep learning," 2018.

[27] Z. Zhang, "Derivation of Backpropagation in Convolutional Neural Network (CNN)," *University of Tennessee, Knoxville, TN*, pp. 1–7, 2016.

[28] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd International Conference on Learning Representations, ICLR 2015*, pp. 1–14, 2015.

[30] N. Adaloglou, "Intuitive explanation of skip connections in deep learning," *https://theaisummer.com/*, 2020.

[31] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–16, 2016.

[32] S. Ioffe, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015.

[33] E. Sari, M. Belbahri, and V. P. Nia, "How Does Batch Normalization Help Binary Training?," no. NeurIPS, 2019.

[34] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE Laurens," *Journal of Machine Learning Research*, 2008.

[35] B. B. Moser, F. Raue, S. Frolov, S. Palacio, J. Hees, and A. Dengel, "Hitchhiker's Guide to Super-Resolution: Introduction and Recent Advances," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. X, 2023.

[36] E. Betzalel, C. Penso, A. Navon, and E. Fetaya, "A Study on the Evaluation of Generative Models," no. ii, 2022.

[37] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 2818–2826, 2016.

[38] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, and S. Gelly, "Are Gans created equal? A large-scale study," *Advances in Neural Information Processing Systems*, vol. 2018-December, no. NeurIPS, pp. 700–709, 2018.

APPENDIX A
SELF-ATTENTION

Self-attention applies three separate 1×1 convolutions with C filters each to the latent feature maps, $\boldsymbol{x}$, with C channels, creating intermediate feature maps: $f(x)$, $g(x)$ and $h(x)$ with $\bar{C}$ channels. $\bar{C}$ is given by, $\bar{C}=\text{floor}\left(\frac{C}{8}\right)$. The softmax equation normalises the input activations, and can be thought of creating a probability distribution, β. $\beta_{i,j}$ is a measure of how much the model regards the i^{th} location whilst synthesising the j^{th} region and is found using the softmax function:

$$\beta_{j,i}=\text{softmax}(s_{ij})=\frac{\exp(s_{ij})}{\sum_{i=1}^{N}\exp(s_{ij})} \tag{28}$$

where

$$s_{ij}=\boldsymbol{f}(\boldsymbol{x}_i)^T\boldsymbol{g}(\boldsymbol{x}_j) \tag{29}$$

From this the self-attention feature maps, $\boldsymbol{o}$ are given by:

$$\boldsymbol{o_j}=\boldsymbol{v}\left(\sum_{i=1}^{N}\beta_{j,i}\boldsymbol{h}(\boldsymbol{x_i})\right) \tag{30}$$

Where $\boldsymbol{v}$ is a 1×1 convolution, mapping the $\bar{C}$ channels to the C output channels. The output of the self-attention layer, $\boldsymbol{y}$ is given by:

$$\boldsymbol{y_i}=\gamma\boldsymbol{o_i}+\boldsymbol{x_i} \tag{31}$$

Where γ is a learnable scaling factor. The self-attention process is visualised in Figure 8.

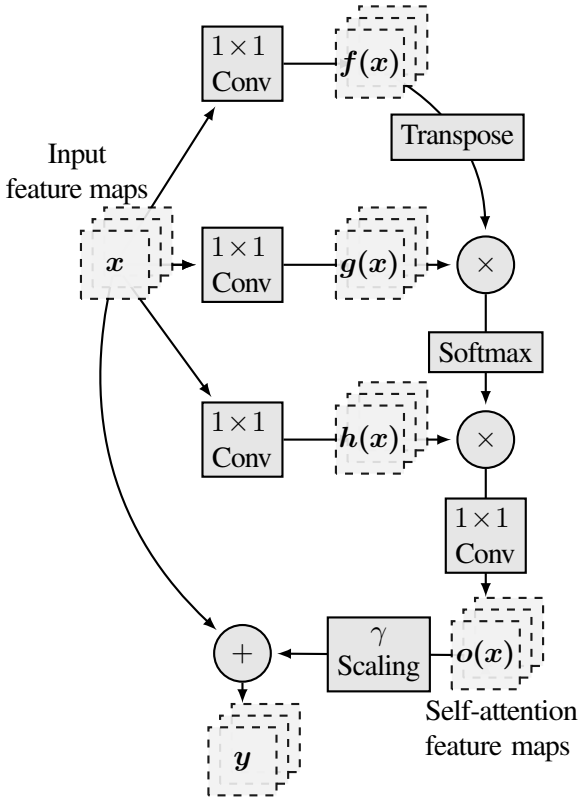

Fig. 8. Self attention process for input feature maps, $\boldsymbol{x}$

TABLE IV
THE `epoch_batch_list` LOOKUP TABLE FOR BASE GAN TRAINING

Resolution	32	64	75
Training epoch limit	8000	10000	-
Fade in epoch limit	-	3000	4000
Batch size	16	16	16

APPENDIX B
GAN TRAINING ALGORITHMS

The `GAN_train` and `ESRGAN_train` functions executed using Algorithms 3 and 4 respectively. The look up table `epoch_batch_list` used in `GAN_train` is given in Table IV.

APPENDIX C
T-SNE PLOTS

T-SNE plots for BaseGAN 32 are shown in Figures 9 and 10.T-SNE plots for BaseGAN 75 are shown in Figures 11 and 12. These plots show that the model covers many dataset modes, even rare cases in the SWIMNSEG dataset.

APPENDIX D
CLOUDGEN OUTPUTS

600×600 resolution output samples for cloudGEN with both BaseGAN 32 and BaseGAN 75 are given in Figures 13, 14, 15 and 16.

Algorithm 3: `GAN_train`

input : E_L, $\theta_N = 0$, $U_T = 0$, Fade_in, $\boldsymbol{X}$; θ_D,θ_G
output Updated GAN parameters, θ_D,θ_G
:

for E_i **in** E_L **do**
  Shuffle Dataset
  $\mathbf{X} \leftarrow$ `Shuffle`($\mathbf{X}$)
  for Labelled image batches, $\boldsymbol{x_R}$, $\boldsymbol{l}$ **in X do**
    Convert labels to one hot format
    $\boldsymbol{l} \leftarrow$ `OneHot`($\boldsymbol{l}$)
    Train the discriminator
    Sample noise
    $\boldsymbol{z} \sim p_z$
    while Saving gradients **with** `tf.GradientTape()` **as** `Dis_tape` **do**
      Create fake samples
      $\boldsymbol{x_F} \leftarrow G(\boldsymbol{z},\boldsymbol{l};\theta_G)$
      Get noisy discriminator output
      $\boldsymbol{y_R} \leftarrow D(\texttt{AddNoise}(\boldsymbol{x_R},\theta_N),\boldsymbol{l};\theta_D)$
      $\boldsymbol{y_F} \leftarrow D(\texttt{AddNoise}(\boldsymbol{x_F},\theta_N),\boldsymbol{l};\theta_D)$
      Get discriminator loss
      $l_D \leftarrow \texttt{DisLoss}(\boldsymbol{y_R},\boldsymbol{y_F})$
    end
    Get the gradients
    $\nabla C_D \leftarrow \texttt{Dis\_tape}(l_D,\theta_D)$
    Update parameters
    if $l_D > U_T$ **then**
      $\theta_D \leftarrow \texttt{Adam}(\nabla C_D,\theta_D)$
    end
    Train the generator
    Sample noise
    $\boldsymbol{z} \sim p_z$
    while Saving gradients **with** `tf.GradientTape()` **as** `Gen_tape` **do**
      Create fake samples
      $\boldsymbol{x_F} \leftarrow G(\boldsymbol{z},\boldsymbol{l};\theta_G)$
      Get noisy discriminator output
      $\boldsymbol{y_F} \leftarrow D(\texttt{AddNoise}(\boldsymbol{x_F},\theta_N),\boldsymbol{l};\theta_D)$
      Get loss;
      $l_G \leftarrow \texttt{GenLoss}(\boldsymbol{y_F})$
    end
    Get the gradient
    $\nabla C_G \leftarrow \texttt{Gen\_tape}(l_G,\theta_G)$
    Update parameters
    $\theta_G \leftarrow \texttt{Adam}(\nabla C_G,\theta_G)$
  end
  if Fade_in == **`True`** **then**
    $\alpha \leftarrow \texttt{UpdateFadein}(E_i,E_L)$
  end
end
return θ_D,θ_G

Algorithm 4: `ESRGAN_train`

input : E_L, $\boldsymbol{X}$; θ_D,θ_G
output updated ESRGAN parameters, θ_D,θ_G
:

for E **in** E_L **do**
  Shuffle Dataset
  $\mathbf{X} \leftarrow \texttt{Shuffle}(\mathbf{X})$
  for Paired image batches, $\boldsymbol{x}_{LR}$, $\boldsymbol{x}_{HR}$ **in X do**
    Train the discriminator
    while Saving gradients **with** `tf.GradientTape()` **as** `Dis_tape` **do**
      Create fake samples
      $\boldsymbol{x}_{SR} \leftarrow G(\boldsymbol{x}_{LR};\theta_G)$
      Get the raw discriminator predictions
      $\boldsymbol{P}_{SR,Raw} \leftarrow D(\boldsymbol{x}_{SR};\theta_D)$
      $\boldsymbol{P}_{HR,Raw} \leftarrow D(\boldsymbol{x}_{HR};\theta_D)$
      Get the relative predictions
      $\boldsymbol{P}_{SR} \leftarrow \sigma(\boldsymbol{P}_{SR,Raw} - \texttt{mean}(\boldsymbol{P}_{HR,Raw}))$
      $\boldsymbol{P}_{HR} \leftarrow \sigma(\boldsymbol{P}_{HR,Raw} - \texttt{mean}(\boldsymbol{P}_{SR,Raw}))$
      Get discriminator loss
      $l_D \leftarrow \texttt{BCE}(\boldsymbol{P}_{SR},\boldsymbol{P}_{HR},\text{labels})$
    end
    Get the gradients
    $\nabla C_D \leftarrow \texttt{Dis\_tape}(l_D,\theta_D)$
    Update parameters
    $\theta_D \leftarrow \texttt{Adam}(\nabla C_D,\theta_D)$
    Train the generator
    while Saving gradients **with** `tf.GradientTape()` **as** `Gen_tape` **do**
      Create fake samples
      $\boldsymbol{x}_{SR} \leftarrow G(\boldsymbol{x}_{LR},\boldsymbol{l};\theta_G)$
      Get the raw discriminator predictions
      $\boldsymbol{P}_{SR,Raw} \leftarrow D(\boldsymbol{x}_{SR};\theta_D)$
      $\boldsymbol{P}_{HR,Raw} \leftarrow D(\boldsymbol{x}_{HR};\theta_D)$
      Get the relative prediction
      $\boldsymbol{P}_{SR} \leftarrow \sigma(\boldsymbol{P}_{SR,Raw} - \texttt{mean}(\boldsymbol{P}_{HR,Raw}))$
      Get generator loss
      $l_G \leftarrow \texttt{BCE}(\boldsymbol{P}_{SR},\text{misleading labels})$
      Compute the pixel loss
      $l_{\text{pixel}} \leftarrow \texttt{MSE}(\boldsymbol{x}_{SR},\boldsymbol{x}_{HR})$
      Format images and compute the perception loss
      $V_{SR} \leftarrow \texttt{pre\_VGG}(\boldsymbol{x}_{SR})$
      $V_{HR} \leftarrow \texttt{pre\_VGG}(\boldsymbol{x}_{HR})$
      $l_{\text{percep}} \leftarrow \texttt{MSE}(V_{HR},V_{SR})$
      Calculate the total loss
      $l_{\text{total}} \leftarrow 5 \times 10^{-3} l_G + l_{\text{percep}} + 10^{-2} l_{\text{pixel}}$
    end
    Get the gradient
    $\nabla C_G \leftarrow \texttt{Gen\_tape}(l_{\text{total}},\theta_G)$
    Update parameters
    $\theta_G \leftarrow \texttt{Adam}(\nabla C_G,\theta_G)$
  end
end
return θ_D,θ_G

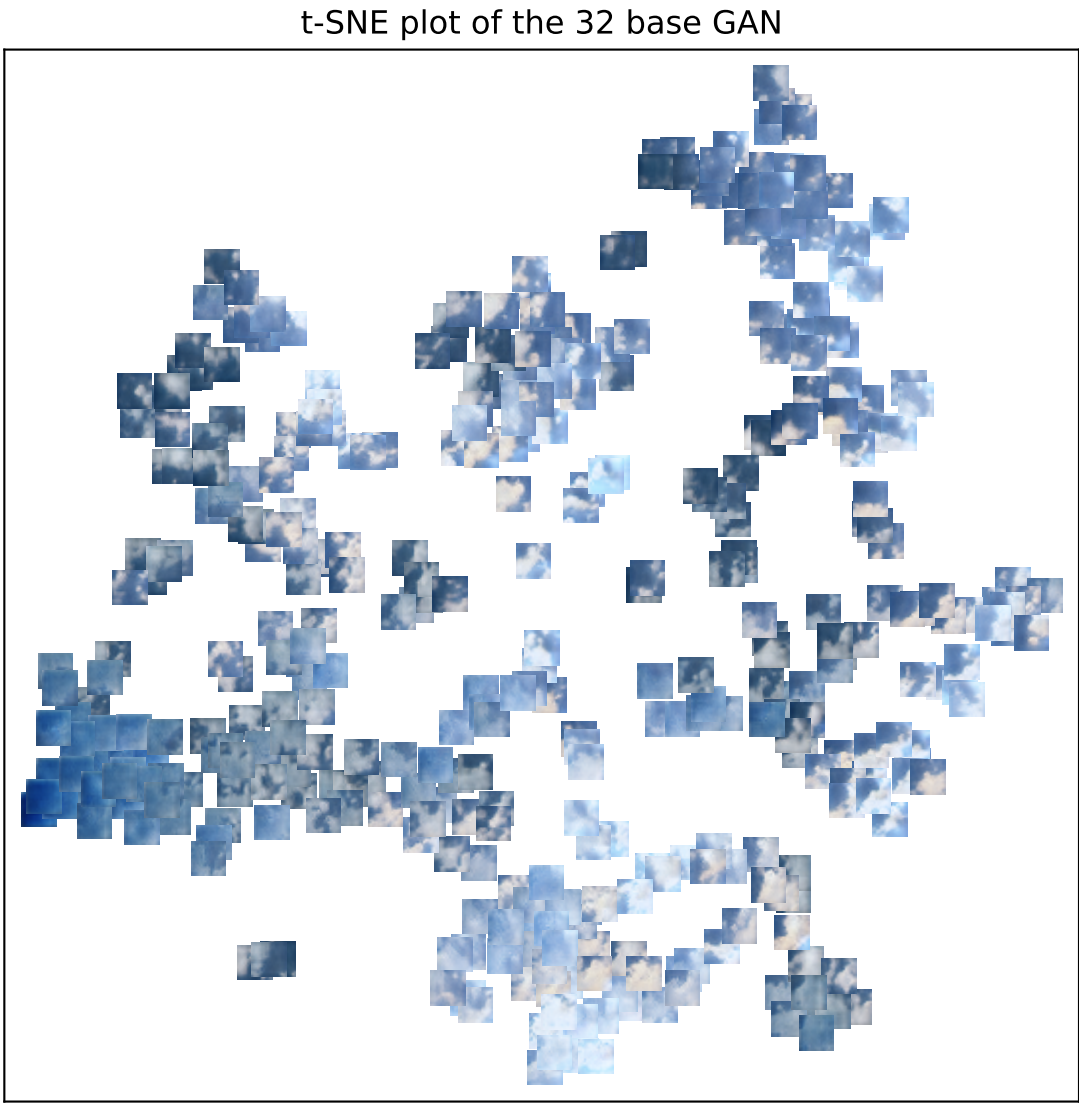

Fig. 9. t-SNE plot for BaseGAN 32's day class with perplexity 10.

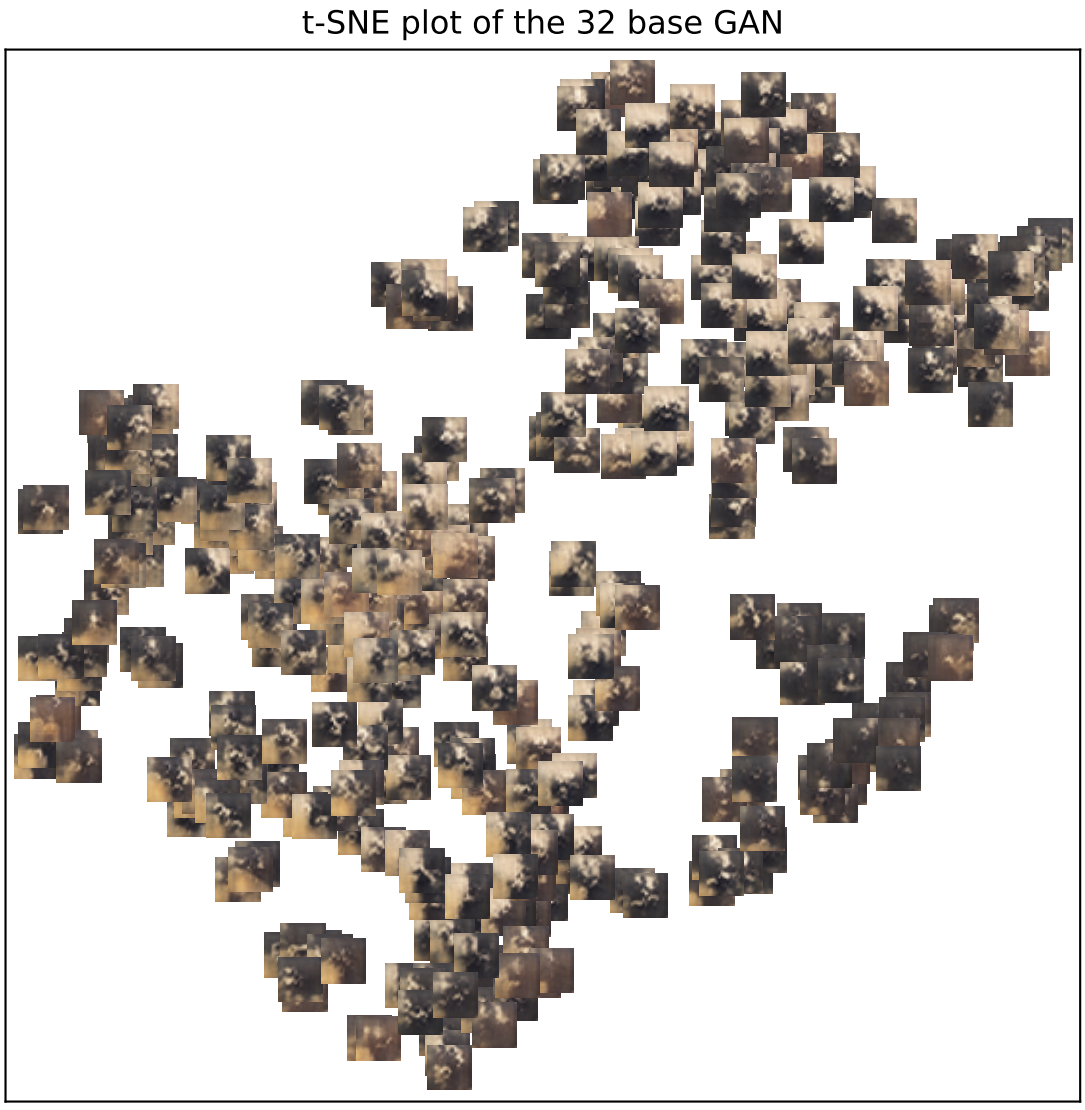

Fig. 10. t-SNE plot for BaseGAN 32's night class with perplexity 10.

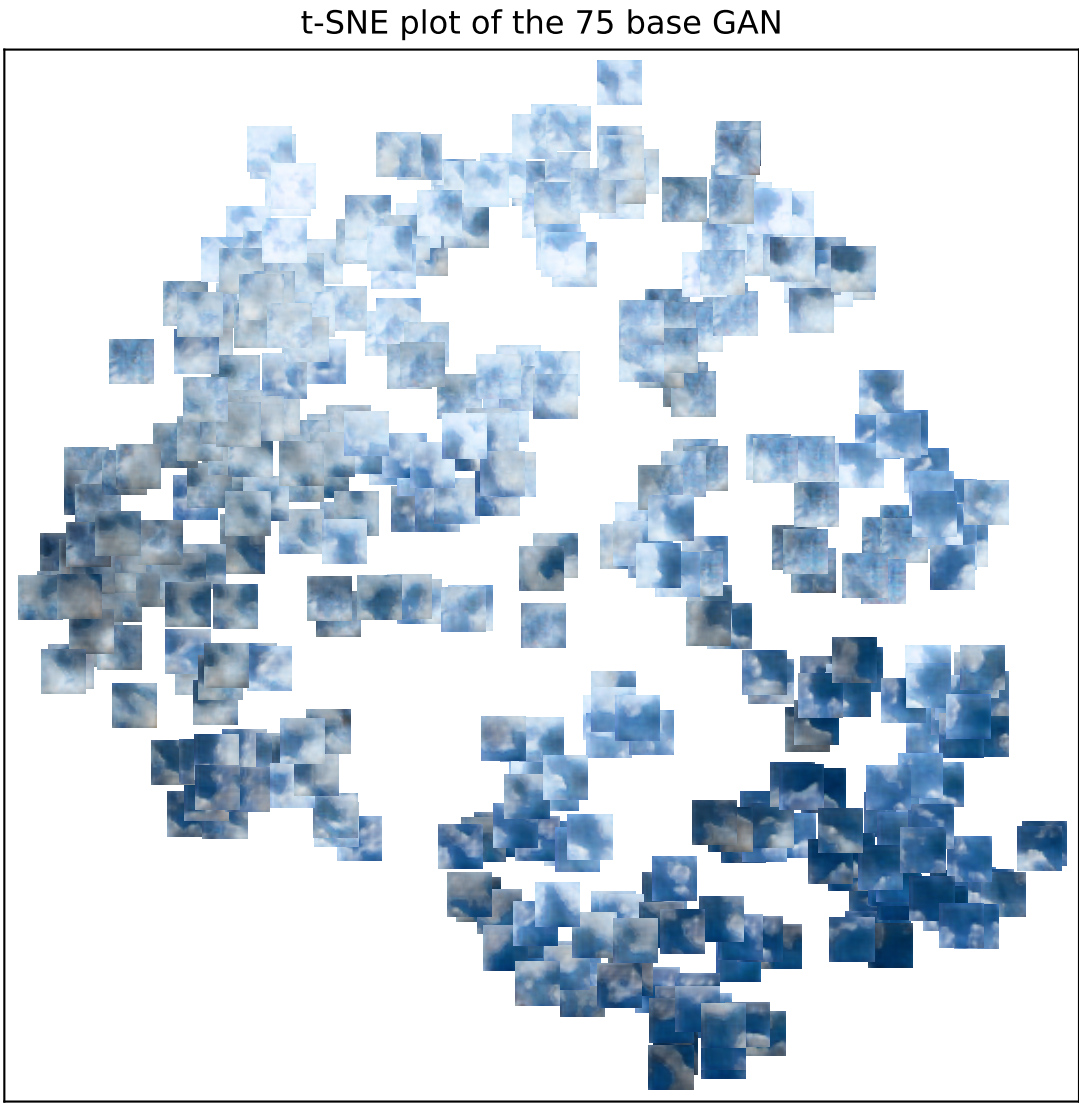

Fig. 11. t-SNE plot for BaseGAN 75's day class with perplexity 10.

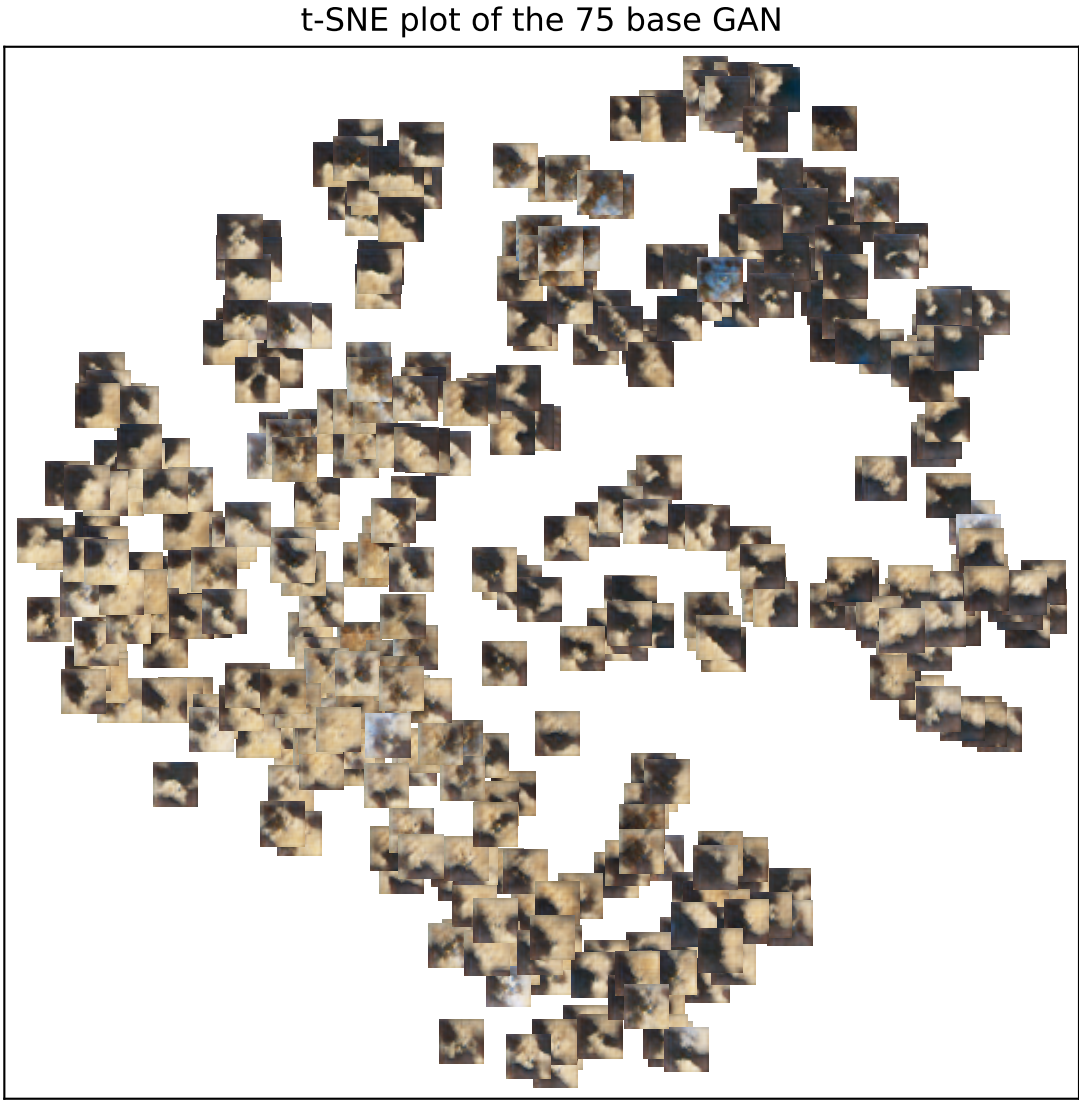

Fig. 12. t-SNE plot for BaseGAN 75's night class with perplexity 10.

Fig. 13. Output images for cloudGEN with BaseGAN 32 for the day class.

Fig. 14. Output images for cloudGEN with BaseGAN 32 for the night class.

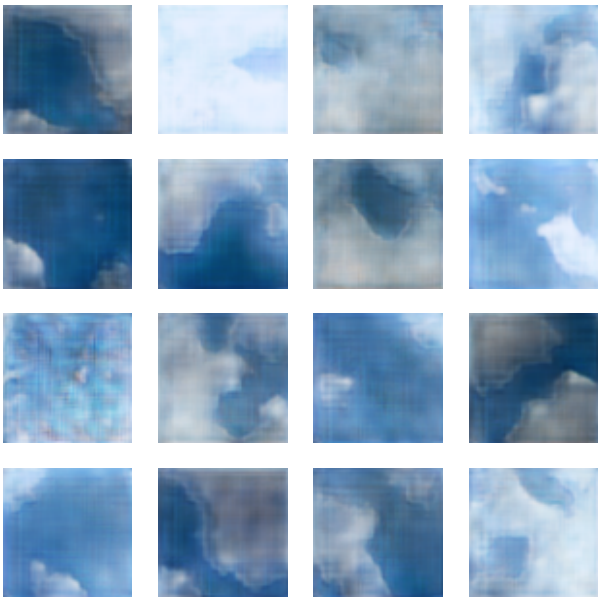

Fig. 15. Output images for cloudGEN with BaseGAN 75 for the day class.

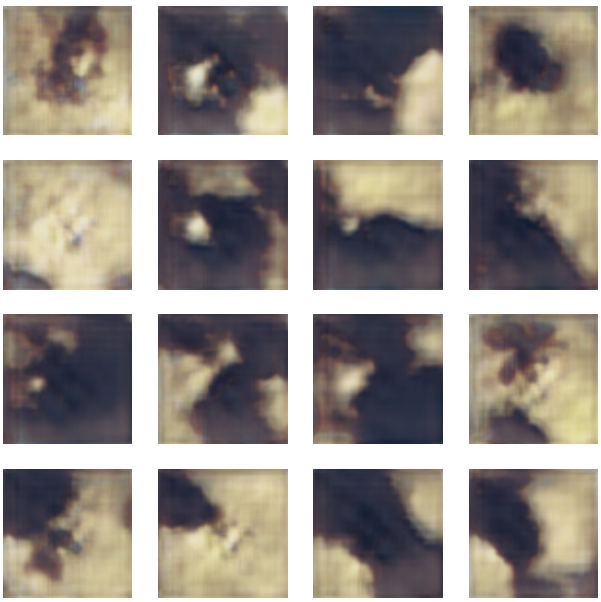

Fig. 16. Output images for cloudGEN with BaseGAN 75 for the night class.