Kick: Shift-N-Overlap Cascades of Transposed Convolutional Layer for Better Autoencoding Reconstruction on Remote Sensing Imagery

SEUNGKYUN HONG, (Member, IEEE), AND SA-KWANG SONG
Department of Data and HPC Science, Korea University of Science and Technology (UST), Daejeon 34113, South Korea
Research Data Sharing Center, Korea Institute of Science and Technology Information (KISTI), Daejeon 34141, South Korea
Corresponding author: Sa-Kwang Song (esmallj@kisti.re.kr)

This work was supported in part by the Korea Institute of Science and Technology Information (KISTI) through the Project “Establishing a System for Sharing and Disseminating Research Data” under Grant K-20-L01-C04, and in part by the Korea National Supercomputing Center with supercomputing resources under Grant KSC-2019-CRE-0257.

ABSTRACT A convolutional autoencoder is an essential deep neural model architecture for understanding and predicting large-scale and widespread multi-dimensional information such as remote sensing imagery. To training a convolutional autoencoder, an automatic image reconstruction from input data and evaluation is repeatedly performed to achieve optimal reconstruction performance. Checkerboard artifacts, which are frequently produced on output images and lead to degraded image quality, are a significant issue during image reconstruction using a convolutional autoencoder. To remedy this coarse visual saliency issue during model training, we propose the ‘Kick’ deconvolutional layer - a cascaded transposed convolutional layer with pixel shifting and overlapping for checkerboard pattern smoothing. By using pixel-shifted identity convolutional layers, we improved image reconstruction performance using fewer trainable decoder parameters than previously suggested models without losing reconstruction capability. Moreover, our proposed layer can be used with any type of convolutional autoencoder, including typical convolutional autoencoders and adversarial autoencoders. To evaluate an image reconstruction performance of our suggested deconvolutional layer, we used a dataset containing 12 years of geostationary satellite observation data of East Asia.

INDEX TERMS Deconvolutional layer, convolutional autoencoder, adversarial autoencoder, image reconstruction, checkerboard artifact, satellite observation.

I. INTRODUCTION Predicting future weather events has been a challenge for human beings since ancient times. Extreme weather events such as heavy rain and typhoons, which strike Eastern Asian several times every year, are difficult to predict and contribute to tremendous financial and social losses. Therefore, better prediction strategies may help minimize some of these losses. To understand intricate weather patterns and investigate potential risk management strategies, numerous meteorological organizations gather and extract digitized weather data from earth simulations called numerical weather prediction (NWP) models [1]–[3] with continuous observation data assimilation.

A significant benefit of using the NWP model is the capability of predicting short-term (within few days) weather prediction in a few hours of computation. At the same time, those digitized weather data are challenging to interpret or differentiate because each data instance consists of multiple sparse and simultaneous weather events. Furthermore, an operation period of NWP model simulation required a comparatively long time than a period of each remote sensing weather is observed. Therefore, if any weather model can extract useful digitized weather data mainly from remote observation data, it could understand current weather status and predict upcoming weather phenomena.

To understand the large-scale unlabeled remote observation images, an unsupervised machine learning algorithm could be used to discover hidden information by the model itself. An autoencoder is one of the well-known neural network techniques used for discovering features from unlabeled...
data. It is widely used to produce data-driven encoding models by matching input and output data with the same data [4], [5]. An autoencoder is comprised of two major parts - the encoder and the decoder. The encoder compresses a representation of input data as a latent vector, while the decoder reconstructs expected original input data from the latent vector. Accordingly, an autoencoder model can be regarded as an identity function that learns a minimized representation of weather observation and simulation data in an unsupervised manner. ExtremeWeather [6] is one example of a semi-supervised climate autoencoder model used for learning representations of extreme weather events. It utilizes convolutional neural networks to improve spatial feature extraction for both the encoder and the decoder. This model was successfully deployed in a high-performance computing environment where it boosted time-consuming tasks within one hour [7].

Meanwhile, as a decoder essentially reconstructs expected original input data to understand and reveal important associative filters [8], [9], it consists of a cascaded decode unit called the deconvolutional layer [10], [11]. Every single deconvolutional layer generates more high-dimensional output (fan-out) from an input tensor (fan-in) with a lack of information, as the total amount of output is larger than the input compared to the typical convolutional operation of large input (fan-in) and small output (fan-out) manner [12]–[15]. To overcome information shortage while reconstruction procedure, transposed convolution, a type of deconvolution strategy, is utilized to produce larger dimensions of output by manipulating strides of deconvolutional operation and dimension of deconvolutional filter size. In other words, the optimal size of the deconvolutional filter and striding step must be addressed to achieve better decoding of the latent vector.

Previous studies such as Inception-v2 [16], ResNet-101 [17], and Inception-v4 [18] concerning the design of convolutional filters already proved that utilization of $1 \times 1$ (identity convolution) and $3 \times 3$ (regular convolution) size of filter could achieve a considerable performance of feature extraction as well as a significant reduction of a number of trainable parameters in the overall model. Especially, an identity convolution is a kind of convolutional operation which produces a filtered output with the same dimension of input tensor achieved by $1 \times 1$ convolutional filter size and stride 1. By using a $1 \times 1$ convolutional operation, a neural network model can easily produce various outputs from any input tensor with minimum computational costs compared to using regular convolution such as $3 \times 3$ and $5 \times 5$. Still, identity convolution is not suitable for core deconvolutional operation compared to regular convolution due to its characteristics of identical dimensions of input and output, which have less similarity with small input and large output. Therefore, a design of deconvolutional operation with small input (fan-in) and large output (fan-out) could be utilized to decode an encoded latent vector and is easily configurable by reusing successful regular convolutional operation into a transposed form which produces enlarged filtered feature maps with stepped striding.

As a result of choosing an optimal deconvolutional filter size like $3 \times 3$, discussion on selecting proper striding step of the deconvolutional filter must be followed to determine a dimension of output tensor, which affects output pattern as well as computation cost. An exhaustive study on a deconvolutional operation provides an interactive web page [19] for visualizing simple results of how a deconvolutional operation is made with configurable filter size and stride step. Three ways are possible to produce checkerboard patterns on output tensor during our optimal $3 \times 3$ deconvolutional operations with various stride steps denoted as Figure 1, and below is a list of how checkerboard patterns could be made.

- Visual checkerboard artifacts (stride < kernel):
  - 2 strides (overlaps on output: 1)
- Visual checkerboard artifacts (stride > kernel):
  - 4, 5, 6 strides (gaps on output: 1, 2, 3)
- Contexture checkerboard artifacts:
  - 2, 3, 4, 5, 6 strides (skipping on input: 1, 2, 3, 4, 5)

According to the visualization of deconvolutional (deconv) progress and results as Figure 1 and Figure 2,
deconvolutional operations with strides over 2 produces checkerboard patterns on an output tensor. In the case of deconv operation with 1 step, any checkerboard patterns may difficult to occur. However, an output dimension is almost similar to the input tensor. Due to the issue of output dimension on deconv, stride steps over one is suggested. With two steps of striding, deconv operations occasionally produce an overlapped, inner checkerboard pattern on output tensor, which is known as checkerboard artifacts. Also, information on input tensor may be scattered (information skipping) while blank space is attached around the target input pixel per every deconv operation as the stride size of deconv operation grows, which can be regarded as contexture checkerboard artifacts. In other words, large stride steps on deconvolution are not proper due to a repeated blank space attachment during the deconv operations, especially on latent encoded information. Hence, two steps of kernel striding may be proper than any other large steps since it occurs the minimum contextual checkerboard artifacts. Nevertheless, these checkerboard artifacts on the final output images after the reconstruction process using the convolutional autoencoder are inevitable, which leads to a deconv operation to produce uneven data concentration or scattering on output tensor during each deconvolution process. Furthermore, an autoencoding reconstruction of delicate images such as remote sensing imagery providing highly detailed visual information about ongoing weather observation could be easily degraded due to the repeatedly produced checkerboard patterns on final reconstruction output.

To reduce inevitable checkerboard patterns on deconvolution operations, recent studies such as explicit nearest-neighbor resizing (NNr) [19] at deconvolution step and Pixel Deconvolutional Networks (PixelTCL) have been previously proposed [20] as advanced deconvolution strategies. An NNr deconvolution performs upsampling prediction by first resizing in a bicubic or bilinear manner. Resizing is then followed by additional convolutional operation for nonlinearity. A benefit of NNr is a pixel-wise mashup with minimum tensor operations. However, the performance of NNr relies heavily on input feature maps, which are the results of each previous convolutional layer. Unlike NNr convolution, PixelTCL performs deconvolution operation in a full convolutional manner, without any deconvolutional operations such as transposed convolution. Instead, PixelTCL utilizes a sparse convolutional operation for upsampling, followed by a jointly filtered convolutional operation. A benefit of PixelTCL is the local attention that attracts for its ability to build detailed mask maps. However, it merely produces chunks of several monotonous mask maps.

To solve these problems, we introduce a Kick deconvolutional layer - a cascaded transposed convolutional layer with a residual connection that minimizes checkerboard artifacts both in final and intermediate reconstructions. The name of the proposed deconvolutional layer ‘Kick’ is motivated by the motion picture ’Inception’, which depicts both cascaded dream layers and a method of recall processes in a very deep complex world. The remainder of this paper is organized as follows. In the following section, we discuss the autoencoder architecture and checkerboard artifacts and present our proposed Kick deconvolutional layer for better autoencoding reconstructions. In the third and fourth sections, we compare the performance of a conventional convolutional decoder and the proposed decoder using kick deconvolutional layers. We present our conclusions in the final section.

II. MATERIALS AND METHODS
A. DATASET DESCRIPTION
To learn large-scale weather changes and their characteristics by using autoencoders, we use a homogeneous remote observation dataset from three discrete satellite programs. Each satellite program consisted of four discrete observations, including two infrared (IR), one shortwave infrared (SWIR), and one water-vapor (WV) images. We organized the 12-year data of remote observations from July 2005 to December 2017 into a single dataset (307,808 scenes). Observations are detailed further in Table 1.

B. IMAGERY REPROJECTION AND PREPROCESSING
An original observation was made regarding geostationary projection wherein each satellite program used discrete longitude with a heterogeneous longitude operation [21]–[23]...
Finally, we split the dataset into three parts to build separate training, validation, and test datasets. Since each observation is highly related between a forward and backward observation, conventional k-fold preparation was not made to avoid overfitting. Instead, we segregated the validation and test datasets from the training dataset to make sure observation data that are already utilized to learn its inner features in the model should not be regarded as a validation target nor test measure target. Moreover, a validation dataset is utilized for checking whether a trained model is being overfitted or not per every one epoch. A test dataset is then utilized for literally evaluating model performance similar to the validation dataset. However, elements of the dataset are entirely far from the validation dataset to validate whether the trained model can describe better reconstruction results with generalized parameters per every epoch. As a result of explicit segregation, the training set contains data from July 2005 to December 2013 (189,490 scenes), the validation set from 2014 and 2016 (60,585 scenes), and the test set from 2015 to 2017 (57,733 scenes). Further, we compressed each dataset by the gzip algorithm with GNU Parallel [26] for input pipelines and storage optimization.

C. KICK: CASCADED TRANSPOSED CONVOLUTIONAL LAYER

Our proposed deconvolutional layer, Kick, is a unified deconvolutional layer consists of heterogeneous convolutional layers. A kick layer is comprised of a single \( 3 \times 3 \) transposed convolutional layer, three \( 1 \times 1 \) identity convolutional layers, a single \( 3 \times 3 \) convolutional layer, and an exponential linear unit (ELU) for residual activation.

\[
F_{in} = C_{in} \times H \times W
\]

\[
F_1 = F_{in} \odot \text{Conv}2D_{k3\times3,s2\times2}
\]

\[
F_{cascade} = F_1 \odot 3 \cdot \text{Conv}2D_{k1\times1,s1\times1}
\]

\[
F_{2a}, F_{2b}, F_{2c} = \text{Shift}_{-1,1,1}(F_{cascade})
\]

\[
F_3 = F_1 \odot F_{2a} \odot F_{2b} \odot F_{2c}
\]

\[
F_4 = F_3 \odot \text{Conv}2D_{k3\times3,s1\times1}
\]

\[
F_{out} = \text{ELU}(F_4), \quad C_{out} \times 2H \times 2W
\]

Figure 4 describes an overview how a single kick layer performs complex feature extraction from an input with initial number of tensor depth \((C_{in})\) (Eq. 1). (Each operation equation can be found below.) A transposed convolutional layer of

| Space Agency | Satellite Programme | Imaging Instrument | Central Longitude | Observation Area | Observation Duration |
|--------------|--------------------|--------------------|-------------------|-----------------|---------------------|
| JMA [21]     | MTSAT-1R           | JAMI               | 140° E            | Full Disk (DK01)| Jul. 2005 – Dec. 2010|
|              |                    |                    |                   | Half Disk (DK02)| Jan. 2011 – Jul. 2015|
| KMA [22]     | COMS-1             | MI                 | 128.2° E          | Extended Northern Hemisphere (ENH) | Apr. 2011 – Dec. 2017|

(ENH: Extended Northern Hemisphere (COMS-1), DK01: Full Disk (MTSAT-1R/2), DK02: Northern Half Disk (MTSAT-1R/2)). Accordingly, image reprojection was required to align common central longitude and match the shared available observation area. Actual image reprojection was made from geostationary projection, then to Miller projection, and then transformed to flatten spherical aspects in a flat-wide view. In Miller reprojection, an aspect of the earth might be distorted; however, cropping imagery among the exact latitude/longitude axis is a possible remedy, as well as neglecting the void space.

By reprojection, all images in a dataset are aligned with the common central longitude as 135° E. Cropped areas include 0° (Equator) ~ 70° N Latitude, 100° E ~ 180° (Antemeridian) Longitude (Total aligned area: 80° Longitude × 70° Latitude). After alignment, we resized the images into \( 128 \times 128 \) px square as a higher image resolution requires a greater number of numerical parameters to train and larger memories to process. Additionally, we organized the separated multi-channel images into NCHW (Image tensor format as Number of batches × Channel × Height × Width) format for robust image computation suggested by several performance surveys [24], [25] on convolutional operations, which are yet another general matrix multiplication problem on linear memory architecture with channel-wise convolutional filter. An overview of all aligned areas is described in Figure 3.

![Figure 3](image-url)  
**Figure 3.** An example of reprojection overlap from all kinds of observation with 135° E central longitude. Red: COMS-1 MI (ENH), Yellow: MTSAT-1R (DK02), Green: MTSAT-1R (DK01), Blue: MTSAT-2 (DK02), Violet: MTSAT-2 (DK01), White: desired crop area.

Finally, we split the dataset into three parts to build separate training, validation, and test datasets. Since each observation is highly related between a forward and backward observation, conventional k-fold preparation was not made to avoid overfitting. Instead, we segregated the validation and test datasets from the training dataset to make sure observation data that are already utilized to learn its inner features in the model should not be regarded as a validation target nor test measure target. Moreover, a validation dataset is utilized for checking whether a trained model is being overfitted or not per every one epoch. A test dataset is then utilized for literally evaluating model performance similar to the validation dataset. However, elements of the dataset are entirely far from the validation dataset to validate whether the trained model can describe better reconstruction results with generalized parameters per every epoch. As a result of explicit segregation, the training set contains data from July 2005 to December 2013 (189,490 scenes), the validation set from 2014 and 2016 (60,585 scenes), and the test set from 2015 to 2017 (57,733 scenes). Further, we compressed each dataset by the gzip algorithm with GNU Parallel [26] for input pipelines and storage optimization.

C. KICK: CASCADED TRANSPOSED CONVOLUTIONAL LAYER

Our proposed deconvolutional layer, Kick, is a unified deconvolutional layer consists of heterogeneous convolutional layers. A kick layer is comprised of a single \( 3 \times 3 \) transposed convolutional layer, three \( 1 \times 1 \) identity convolutional layers, a single \( 3 \times 3 \) convolutional layer, and an exponential linear unit (ELU) for residual activation.

\[
F_{in} = C_{in} \times H \times W
\]

\[
F_1 = F_{in} \odot \text{Conv}2D_{k3\times3,s2\times2}
\]

\[
F_{cascade} = F_1 \odot 3 \cdot \text{Conv}2D_{k1\times1,s1\times1}
\]

\[
F_{2a}, F_{2b}, F_{2c} = \text{Shift}_{-1,1,1}(F_{cascade})
\]

\[
F_3 = F_1 \odot F_{2a} \odot F_{2b} \odot F_{2c}
\]

\[
F_4 = F_3 \odot \text{Conv}2D_{k3\times3,s1\times1}
\]

\[
F_{out} = \text{ELU}(F_4), \quad C_{out} \times 2H \times 2W
\]

Figure 4 describes an overview how a single kick layer performs complex feature extraction from an input with initial number of tensor depth \((C_{in})\) (Eq. 1). (Each operation equation can be found below.) A transposed convolutional layer of

![Figure 4](image-url)  
**Figure 4.** An overall structure of convolutional operations in a Kick layer. An arrow line describes the reuse of specific tensor for further feature map manipulation.
$3 \times 3$ kernel size and $2 \times 2$ stride ($\text{Conv2D}_{3 \times 3, 2 \times 2}$) is used to generate up-sampled higher data representations (Eq. 2, $F_1$) from previous low-resolution data representations ($F_{in}$). Identity convolutional layers of $1 \times 1$ kernel size and $1 \times 1$ stride ($\text{Conv2D}_{1 \times 1, 1 \times 1}$) produce various filtered maps ($F_{\text{cascade}}$) from previously generated filter maps ($F_1$) with minimum computation. After an identity convolutional operation, these three filtered maps ($F_{2a}$, $F_{2b}$, $F_{2c}$) are shifted into three separated directions to mitigate checkerboard patterns produced by the transposed convolutional operation.

Detailed shift-and-overlap operation is described in Figure 5. As a result of candidate feature extraction (Eq. 3, $F_{\text{cascade}}$), by identity convolutional operation ($\otimes$), candidate feature maps are produced with the same height ($H$) and width ($W$) dimensions. Instead, the depth of the number of feature maps is increased by three times. Following this, candidate feature maps are sliced into three pieces to fit with the feature maps of the previous source ($F_1$). Then, each sliced feature map performs pixel shifting (Eq. 4, $F_{2a}$, $F_{2b}$, $F_{2c}$) by pixel rolling (pixel moving and bounding) in three ways: rightward, downward, and right–down diagonal direction. Since an operation of pixel rolling performs pixel shifting and moving overflowed edge pixels only at the counter side of the image, it does not replicate existing pixels in the image.

![FIGURE 5. Shifting and overlapping identity filter maps in Kick deconvolution. Newly created filter maps ($F_{2a}$, $F_{2b}$, $F_{2c}$) are separated and shifted. Finally, these filter maps are added (Eq. 5, $\oplus$) in a pixel-wise manner to fit the same number of filters with early transposed convolutional filter map ($F_1$), in order to produce a sum of pixel-wise filters. Five symbols on each feature map show that pixel rolling only performs pixel movement itself, not information replication.](image)

Then, these shifted filter maps ($F_{2a}$, $F_{2b}$, $F_{2c}$) and the original deconvolutional map ($F_1$) are added (Eq. 5, $\oplus$) in a pixelwise manner, followed by a $3 \times 3$ convolutional operation (Eq. 6, $\otimes$) which produces a complex image representation from residual operations ($F_2$). Finally, an exponential linear activation function outputs the non-linear outputs (Eq. 7, ELU) to follow deconvolutional steps with decreased depth of output tensor ($C_{out}$) as well as increased heights and widths ($2H \times 2W$) compared to the initial heights and widths ($H \times W$). A detailed schematic of the kick deconvolution layer with the operation configuration is described in Figure 6.

**D. DIFFERENCES AMONG DECONVOLUTIONAL OPERATIONS**

To describe characteristics of each deconvolutional operation, we visualized a sample deconvolutional process.

![FIGURE 6. Details of deconvolutional operations in a single kick layer.](image)

As Figure 7. A significant difference between Kick and PixelTCL deconvolution is that PixelTCL do masked tessellation from low-dimensional feature maps to up-scaled feature maps to solve checkerboard artifacts. Instead, Kick deconvolution performs subsidiary $1 \times 1$ convolution and matching pixel-shifted feature maps to damping checkerboard artifact on entire feature maps, which is far from partial masked tessellation on feature maps of PixelTCL deconvolution.

Below is a list of detailed deconvolutional operations:

(a) Plain (1a): typical deconvolution
- 1a: 2D Transposed Convolution (2X up-scaling) / kernel size: 3 / stride: 2 / padding: SAME

(b) NNr (2b) [19]: deconvolution using nearest-neighbor up-scaling
- 2a: Nearest-neighbor resize (2X up-scaling)
- 2b: 2D Convolution / kernel size: $3 \times 3$ / stride: 1 / padding: SAME

(c) PixelTCL (3e) [20]: deconvolution from feature maps dilation and masked pixel tessellation
Accordingly, Kick proposed kick deconvolutional layer (PixelTCL), and our pixel transposed convolutional layer (Plain), nearest-neighbor resizing (Plain), convolutional autoencoder [27] (ConvAE) for extra input regression and an adversarial convolutional autoencoder [5] (AdvAE) for basic feature extraction and input regression and an adversarial convolutional autoencoder [27] (AdvAE) for extra input regression through generative adversarial learning. Based on the architectures of the autoencoders, we implemented four discrete decoders on each autoencoder, such as the typical convolutional decoder in an autoencoder architecture, of an autoencoder to verify the effectiveness of the decoder model parameters, we followed the typical model structure of a convolutional autoencoder [2] for reproducible results. For more detailed convolutional layers configurations, please refer Appendix. Convolutional Layer Configurations of Autoencoder Networks.

E. EXPERIMENTAL SETUP
As the Kick deconvolutional layer can be used in any type of convolutional decoder in an autoencoder architecture, we chose two main architectures of autoencoders: a typical convolutional autoencoder [5] (ConvAE) for basic feature extraction and input regression and an adversarial convolutional autoencoder [27] (AdvAE) for extra input regression through generative adversarial learning. Based on the architectures of the autoencoders, we implemented four discrete decoders on each autoencoder, such as the the typical convolutional decoder (Plain), nearest-neighbor resizing (NNr), pixel transposed convolutional layer (PixelTCL), and our proposed kick deconvolutional layer (Kick). Accordingly, eight combinations of autoencoder model variations (2 architectures × 4 decoder types) were made. To set the model parameters, we followed the typical model structure of an autoencoder to verify the effectiveness of the decoder configuration. We set the number of convolutional filters as a multiplier of 16 to gradually increase the number of convolutional filters (64, 96, 128, 192, 256, 384) and deconvolutional filters (256, 192, 128, 96, 64, 4-to fit in the dimension of input data) with a stride size of 2 (SAME method) to easily predict the dimension of each convolution (decreased by two per layer) and deconvolution (increased by two per layer) step. As adversarial training requires a dense (1-d auxiliary data space) data space for model regularization, we reshaped the latent space representation (denoted as limbo) from 384 × 2px × 2px to 1 × 1536 and vice versa immediately before input reconstruction. Finally, an overview of the complete model structure is shown in Figure 8.

For fair model training and comparison, we use specific model configurations across all model presets based on our empirical pilot tests. Glorot (random uniform) [28] was used for random kernel initialization. To handle a large scale of the training dataset, a mini-batch gradient descent [29] algorithm with an adaptive learning rate optimization (Adam) [30] was used for the model optimization policy with an initial learning rate of 1e-4. Based on our initial model lookup, performing the entire dataset lookup for 40 epochs is enough for the epoch setup. Further, 32 instances are suitable for the mini-batch size, as it was the maximum number of batch size we could increase, in order to fit in our computing environment. For model computation and storage, we used a single-precision (32-bit/FP32) data type to cover a range of high-precision parameters. Moreover, we trained each model preset with random kernel initialization (cold-start training) repeatedly for 50 cycles to acquire less biased and reproducible results. For more detailed convolutional layers configurations, please refer Appendix. Convolutional Layer Configurations of Autoencoder Networks.

III. RESULTS
A. RECONSTRUCTION COMPARISON - QUANTITATIVE
We evaluated an image reconstruction performance from all model presets based on several metrics such as mean squared error (MSE), mean absolute error (MAE), peak signal-to-noise (PSNR) metrics, and structural similarity [31] (SSIM). In cases of both convolutional and adversarial autoencoder structures, both reconstruction performance of NNr and PixelTCL models were lower than that of the typical (Plain) model. Furthermore, our suggested model (Kick) shows better results than any other models, including typical (Plain)
decoder, as seen in Figure 9. Especially, an average performance of NNr model in ConvAE preset shows a noticeable performance gap between any other model presets. Likewise, a general training failure of PixelTCL model in AdvAE preset was observed since epoch 20.

Finally, all model presets were evaluated with test dataset (Y2015, Y2017) and Table 2 lists a set of average loss and quality results from an entire model preset accordingly. Our model (Kick) only shows an improvement from baseline (Plain), compared to other suggested models in both ConvAE and AdvAE preset. Interestingly, both NNr model in ConvAE preset and PixelTCL model in AdvAE preset shows the worst performance, which is a similar aspect from the training result.

B. RECONSTRUCTION COMPARISON - QUALITATIVE
To demonstrate an interpretable reconstruction performance of each model, we obtained multiple prediction results to compare a visual aspect and pixel distribution of each model from randomly chosen observation images as input data. We conducted two types of a qualitative survey on model reconstruction results: (1) Absolute Pixel Difference Comparison and (2) Pixel Distribution Comparison between model results and original input data. As a result,
Figure 10 describes an example of the absolute pixel difference between image reconstruction results from both ConvAE and AdvAE model presets and original input data. In the figure, all pixel values are described as a decimal range (0.0 ~ 1.0), and each model has its own reconstruction result followed by absolute pixel errors accordingly. Furthermore, all images are sorted as IR1, IR2, WV, and SWIR channel order.

There are several interesting points about reconstruction results from Figure 10:

- In typical autoencoding model (ConvAE), our model produces clearer and finer images than another model presets, particularly tiny point clouds and edges of coarse clouds.
- A nearest-neighbor resize (NNr) deconvolution generates almost blurred and uninterpretable images in both ConvAE and AdvAE model presets.
- In an adversarial learning model (AdvAE), a result from each model shows blurry and abstract results compared to a typical autoencoding model (ConvAE) preset.
- In AdvAE model preset, Plain deconvolution products significant checkerboard artifacts on WV channel (third-order), compared to other model presets.

- In AdvAE model preset, Kick deconvolution products slightly sharp results compared to PixelTCL model preset.

As shown in the figure, our suggested model shows more alike and definite reconstruction results than any other model presets, which is proved by both visual likeness and less absolute difference from input data. More comparison samples are available through an attached supplement material.

Even though we surveyed on absolute pixel differences among model presets, it is still hard to interpret and compare image reconstruction capabilities from the results of each model. Since each pixel value in the entire image space can be regarded as an interval value of actual pixel (re-scaled) range, we computed a pixel frequency histogram from reconstruction result and original source data of each model. From a statistical perspective, those pixel frequency histograms can be used to determine the image reconstruction capabilities of each deconvolutional model.

Below is a list of the information we can discover from pixel distribution histogram:

- Start and end of pixel occurrence in a range of entire distribution
- A cumulative frequency of each pixel value
FIGURE 11. A set of image reconstruction results from each ConvAE model and its pixel distribution histograms. (image colormap=CMRmap, histogram bins=256) All model results are originated from a same input source (200604172115).

- Overall pixel reconstruction trends compared to the source input distribution.

An example of pixel distribution statistics from multiple reconstruction results can be described as Figure 11 and Figure 12 respectively. Consequently, we used different colormap (CMRmap) for displaying a pixel values range (0.0 ~ 1.0 of 1024 pixel levels) of image reconstruction from each model preset. In addition, we computed multiple...
FIGURE 12. A set of image reconstruction results from each AdvAE model and its pixel distribution histograms. (image colormap=CMRmap, histogram bins=256) All model results are originated from a same input source (200604172115).

Histograms from each reconstruction result in two ways: (a) an entire pixel distribution from all channels, followed by (b) discrete pixel distribution from each channel.

In the figure, we found some noteworthy points from pixel distribution histograms as below:

- We can easily determine the performance of each model by comparing histograms between image reconstruction results and source input data. We marked each histogram of source input data as green color and image reconstruction results as red color.
FIGURE 13. A set of last deconvolution process from four discrete policies. An asterisk symbol (*) describes a partial process of each step, and an apostrophe symbol (’) describes a result of activation function after each step. Each sub operation in deconvolutional policy are described in Section II-D and Appendix. Input dimension (C x H x W): 64 x 64 x 64 -> Output dimension (C x H x W): 4 x 128 x 128.
In both ConvAE and AdvAE model presets, all model shows a significant gap on Ch3 (bottom left of each model reconstructions) histograms between reconstruction and source data.

In both ConvAE and AdvAE model presets, our suggested model (Kick) provides similar maximum pixel frequency on an entire pixel range (around 1000), while other models highly exceed the maximum pixel frequency.

In AdvAE model presets, reconstruction histograms of all model presets are significantly different from source data histograms.

One notable found in Figure 12 is a significant gap of adversarial learning (AdvAE) model histograms between reconstruction results and source input data. There are several possible reasons why adversarial models provide less accurate reconstruction results than typical autoencoding models, such as model optimization failure and difficulties of training observation images in the adversarial method. Nonetheless, our suggested model provides most close reconstruction results from both ConvAE and AdvAE model presets to a source input data in perspective of pixel distribution likeness compared to any other model presets. As a result of performance comparison on image reconstruction, our suggested kick layer on decoder shows better results from both quantitative and qualitative aspects.

IV. DISCUSSION

We found a structure of decoder unit does not only modifies the visual aspect of reconstruction output but also affects on the quantitative performance on both convolutional (ConvAE) and adversarial (AdvAE) autoencoders while training massive remote sensing imagery dataset. Furthermore, our suggested deconvolutional unit (Kick) shows better reconstruction results and fewer checkerboard artifacts compared to other suggested deconvolutional operations. If so, why Kick deconvolution works better on reconstruction and autoencoding earth observation datasets? Besides, what is the impact of enhancing reconstruction performance on autoencoder models with remote sensing imagery datasets? We must focus on how image reconstruction is processed, especially in the decoder parts of the autoencoder model architectures, to answer those fundamental questions.

A. WHY KICK DECONVOLUTION WORKS?

To explain an aspect of deconvolutional operations, we visualized multiple feature maps of each deconvolutional layer in an autoencoding model, particularly from each ConvAE model, as Figure 13. We described Kick deconvolution is a set operation of shifting and overlapping cascaded transposed convolutional operation. Meanwhile, PixelTCL deconvolution also performs overlapping deconvolutional layers.

If so, what is the main difference among four deconvolutional operations?

(a) Plain: generates an enlarged feature map containing checkerboard artifacts
(b) NNR: resizes feature maps by neighbor value referencing and $1 \times 1$ identity convolution
(c) PixelTCL: dilates feature maps by convolution and scale-up tessellation by slot-filling
(d) Kick: expands several diverged scaled-up feature maps and fusion of checkerboard patterns by pixel-shifted feature maps

More specifically, (a) Plain deconvolution performs a basic image reconstruction. Meanwhile, (b) NNR deconvolution tries to modify a previously resized feature map by extra convolutional operations, which leads to a blurred result as an output image. Besides, (c) PixelTCL deconvolution tries to resolve checkerboard artifacts by explicit features tessellation on output feature maps, which causes a pixel distribution imbalance on an output image.

Instead, (d) Kick deconvolution provides multiple pixel-shifted feature maps used for:

1. a dissolution of checkerboard patterns by overlapping
2. an enhanced distributional balance for the absorption of extremely singular feature maps
3. a wide range of feature expression based on subsidiary feature maps

We believe those three characteristics make Kick deconvolution to understand hidden information of input data better than other deconvolutional operations. As a result, (d) Kick deconvolution produces less visual difference compared to other models (a, b, c) as Figure 13.

B. WHAT DOES KICK DECONVOLUTION MEAN TO US?

Our goal to improve a reconstruction performance is simple - a better understanding of input data and its hidden information with low pixel difference between a source input data and a prediction result. Every pixel difference in image reconstruction data means more than a numerical prediction error - an optical existence of a cloud phenomenon. Therefore, our suggested deconvolutional model provides a better understanding and replay of pixel distribution from input data during the autoencoding process. In other words, any autoencoder model using our deconvolutional policy may extract and learn more usable information by itself, only by autoencoding input data itself.

V. CONCLUSION

Auto-encoding models with a set of convolutional encoding deconvolutional decoding methods are suggested to understanding complex and weather phenomena from a massive earth observation dataset with full of uncertainties. However, deconvolutional decoding in autoencoding models contains an inevitable checkerboard artifact issue, which leads to blurred and plaid pixel condensation on the final image reconstruction result.

To solve these checkerboard artifact issue, we suggested a series of cascaded convolutional operations with pixel
shifting method to physically smooth the plaid pixel condensation as 'Kick' deconvolution. (Figure 4, 5, 6) To demonstrate an effectiveness of the suggested deconvolutional operation, we evaluated several deconvolutional results in perspective of both quantitative and qualitative approaches. In quantitative analysis (Table 2), our model exceeds any deconvolutional policies in four evaluation metrics (MSE, MAE, PSNR, SSIM) at both ConvAE and AdvAE architectures. Besides, absolute pixel differences and discrete pixel distributions of our model between reconstruction image and source input were also less than any other deconvolutional operations in qualitative analysis. (Figure 10, 11, 12)

In summary, our suggested deconvolutional layer (Kick) operates better autoencoding and image reconstruction on both convolutional (ConvAE) and adversarial (AdvAE) models compared to previously suggested deconvolutional layers. Because the structure of our suggested model is easy to implement, any autoencoder models can utilize our model as a part of a versatile decoder layer. We expect our model may help researchers who are concerning about understanding large-scale and widespread weather phenomena and extracting fine-tuned latent information and convolutional model parameters from remote sensing imagery. Therefore, further targeted surveys such as weather events detection or

| TABLE 3. Convolutional Layer Configuration (Encoder). |
| Layer ID (Conv) | Layer | Type | Parameter of Layers | Trainable? |
| L1 | Conv2D | Conv. Layer | 4 × 128 × 128, 64 × 64 × 64, 96 × 32 × 32, 128 × 16 × 16, 192 × 8 × 8, 256 × 4 × 4 | Yes |
| L2 | Conv2D | Conv. Layer | 64 × 64 × 64, 96 × 32 × 32, 128 × 16 × 16, 192 × 8 × 8, 256 × 4 × 4 | |
| L3 | Conv2D | Conv. Layer | 128 × 64 × 64, 192 × 32 × 32, 256 × 16 × 16, 384 × 8 × 8 | |
| L4 | Conv2D | Conv. Layer | 256 × 16 × 16, 384 × 8 × 8 | |
| L5 | Conv2D | Conv. Layer | 512 × 8 × 8 | |
| L6 | Conv2D | Conv. Layer | 1024 × 4 × 4 | |

| TABLE 4. Deconvolutional Layer Configuration (Decoder - Plain Deconvolution). |
| Layer ID (Deconv) | Layer | Type | Parameter of Layers | Trainable? |
| L6 | Conv2D_T | Deconv. Layer (Transposed Conv.) | 384 × 2 × 2, 256 × 4 × 4, 192 × 8 × 8, 128 × 16 × 16, 96 × 32 × 32, 64 × 64 × 64 | Yes |
| L5 | Conv2D | Conv. Layer | 256 × 2 × 2, 192 × 4 × 4, 128 × 8 × 8, 96 × 16 × 16, 64 × 32 × 32, 4 × 64 × 64 | |
| L4 | Conv2D | Conv. Layer | 128 × 16 × 16, 64 × 32 × 32, 4 × 64 × 64 | |
| L3 | Conv2D | Conv. Layer | 64 × 64 × 64, 32 × 128 × 128, 16 × 256 × 256, 8 × 512 × 512, 4 × 1024 × 1024 | |
| L2 | Conv2D | Conv. Layer | 32 × 32 × 32, 16 × 64 × 64, 8 × 128 × 128, 4 × 256 × 256 | |
| L1 | Conv2D | Conv. Layer | 16 × 16 × 16, 8 × 32 × 32, 4 × 64 × 64 | |

| TABLE 5. Deconvolutional Layer Configuration (Decoder - Nearest Neighbor resize/NNr). |
| Layer ID (Deconv) | Layer | Type | Parameter of Layers | Trainable? |
| L6_nunp | image_resize | Upsampling | 384 × 2 × 2, 256 × 4 × 4, 192 × 8 × 8, 128 × 16 × 16, 96 × 32 × 32, 64 × 64 × 64 | |
| L6_conv | image_resize | Upsampling | 2 × up-scaling resize | |
| L5_nunp | Conv2D | Conv. Layer | 384 × 2 × 2, 256 × 4 × 4, 192 × 8 × 8, 128 × 16 × 16, 96 × 32 × 32, 64 × 64 × 64 | |
| L5_conv | Conv2D | Conv. Layer | 2 × up-scaling resize | |
| L4_nunp | Conv2D | Conv. Layer | 384 × 2 × 2, 256 × 4 × 4, 192 × 8 × 8, 128 × 16 × 16, 96 × 32 × 32, 64 × 64 × 64 | |
| L4_conv | Conv2D | Conv. Layer | 2 × up-scaling resize | |
| L3_nunp | Conv2D | Conv. Layer | 384 × 2 × 2, 256 × 4 × 4, 192 × 8 × 8, 128 × 16 × 16, 96 × 32 × 32, 64 × 64 × 64 | |
| L3_conv | Conv2D | Conv. Layer | 2 × up-scaling resize | |
| L2_nunp | Conv2D | Conv. Layer | 384 × 2 × 2, 256 × 4 × 4, 192 × 8 × 8, 128 × 16 × 16, 96 × 32 × 32, 64 × 64 × 64 | |
| L2_conv | Conv2D | Conv. Layer | 2 × up-scaling resize | |
| L1_nunp | Conv2D | Conv. Layer | 384 × 2 × 2, 256 × 4 × 4, 192 × 8 × 8, 128 × 16 × 16, 96 × 32 × 32, 64 × 64 × 64 | |
| L1_conv | Conv2D | Conv. Layer | 2 × up-scaling resize | |
classification using the previously trained model must be followed to evaluate a better performance of a well-trained autoencoder model.

APPENDIX

CONVOLUTIONAL LAYER CONFIGURATIONS OF AUTOENCODER NETWORKS

A. ENCODER
See Table 3.

B. DECODER - PLAIN DECONVOLUTION
See Table 4.

C. DECODER - NEAREST-NEIGHBOR RESIZE (NNr) DECONVOLUTION
See Table 5.

D. DECODER - PIXEL TRANSPOSED CONVOLUTIONAL (PixelTCL) DECONVOLUTION
See Table 6.

E. DECODER - CASCADED PIXEL SHIFT AND OVERLAP (Kick) DECONVOLUTION
See Table 7.
TABLE 7. Deconvolutional Layer Configuration (Decoder - Cascaded Pixel Shift and Overlap Deconvolution/Kick).

| Layer ID (Deconv) | Layer | Type | Parameter of Layers | Trainable? |
|-------------------|-------|------|---------------------|------------|
| L6_Plain          | Conv2D | Deconv Layer (Transposed Conv.) | 384 x 2 x 2, 3 x 3, 256, SAME, 2 x 2, - | 256 x 4 x 4 | Yes |
| L6_Spinup         | Conv2D | Conv Layer | L6_Plain, 1 x 1, 3 x 256, VALID, 1 x 1, - | (3 x 256) x 4 x 4 | Yes |
| L6_SpinAll        | Pixel Shift | L6_Spinup | Pixel Shifting (-, | No |
| L6_Prejoin        | Checkered Pixel Integration | L6_Plain/SpinAll | Pixel-wise Addition | ELU | 256 x 4 x 4 | No |
| L6_Postconv       | Conv2D | Conv Layer | L6_Prejoin, 3 x 3, 256, SAME, 1 x 1, ELU | 256 x 4 x 4 | Yes |
| L5_Plain          | Conv2D | Deconv Layer (Transposed Conv.) | L6_postconv, 3 x 3, 192, SAME, 2 x 2, - | 192 x 8 x 8 | Yes |
| L5_Spinup         | Conv2D | Conv Layer | L5_Plain, 1 x 1, 3 x 192, VALID, 1 x 1, - | (3 x 192) x 8 x 8 | Yes |
| L5_SpinAll        | Pixel Shift | L5_Spinup | Pixel Shifting (-, | No |
| L5_Prejoin        | Checkered Pixel Integration | L5_Plain/SpinAll | Pixel-wise Addition | ELU | 192 x 8 x 8 | No |
| L5_Postconv       | Conv2D | Conv Layer | L5_Prejoin, 3 x 3, 192, SAME, 1 x 1, ELU | 192 x 8 x 8 | Yes |
| L4_Plain          | Conv2D | Deconv Layer (Transposed Conv.) | L5_postconv, 3 x 3, 128, SAME, 2 x 2, - | 128 x 16 x 16 | Yes |
| L4_Spinup         | Conv2D | Conv Layer | L4_Plain, 1 x 1, 3 x 128, VALID, 1 x 1, - | (3 x 128) x 16 x 16 | Yes |
| L4_SpinAll        | Pixel Shift | L4_Spinup | Pixel Shifting (-, | No |
| L4_Prejoin        | Checkered Pixel Integration | L4_Plain/SpinAll | Pixel-wise Addition | ELU | 128 x 16 x 16 | No |
| L4_Postconv       | Conv2D | Conv Layer | L4_Prejoin, 3 x 3, 128, SAME, 1 x 1, ELU | 128 x 16 x 16 | Yes |
| L3_Plain          | Conv2D | Deconv Layer (Transposed Conv.) | L4_postconv, 3 x 3, 96, SAME, 2 x 2, - | 96 x 32 x 32 | Yes |
| L3_Spinup         | Conv2D | Conv Layer | L3_Plain, 1 x 1, 3 x 96, VALID, 1 x 1, - | (3 x 96) x 32 x 32 | Yes |
| L3_SpinAll        | Pixel Shift | L3_Spinup | Pixel Shifting (-, | No |
| L3_Prejoin        | Checkered Pixel Integration | L3_Plain/SpinAll | Pixel-wise Addition | ELU | 96 x 32 x 32 | No |
| L3_Postconv       | Conv2D | Conv Layer | L3_Prejoin, 3 x 3, 96, SAME, 1 x 1, ELU | 96 x 32 x 32 | Yes |
| L2_Plain          | Conv2D | Deconv Layer (Transposed Conv.) | L3_postconv, 3 x 3, 64, SAME, 2 x 2, - | 64 x 64 x 64 | Yes |
| L2_Spinup         | Conv2D | Conv Layer | L2_Plain, 1 x 1, 3 x 64, VALID, 1 x 1, - | (3 x 64) x 64 x 64 | Yes |
| L2_SpinAll        | Pixel Shift | L2_Spinup | Pixel Shifting (-, | No |
| L2_Prejoin        | Checkered Pixel Integration | L2_Plain/SpinAll | Pixel-wise Addition | ELU | 64 x 64 x 64 | No |
| L2_Postconv       | Conv2D | Conv Layer | L2_Prejoin, 3 x 3, 64, SAME, 1 x 1, ELU | 64 x 64 x 64 | Yes |
| L1_Plain          | Conv2D | Deconv Layer (Transposed Conv.) | L2_postconv, 3 x 3, 4, SAME, 2 x 2, - | 4 x 128 x 128 | Yes |
| L1_Spinup         | Conv2D | Conv Layer | L1_Plain, 1 x 1, 3 x 4, VALID, 1 x 1, - | (3 x 4) x 128 x 128 | Yes |
| L1_SpinAll        | Pixel Shift | L1_Spinup | Pixel Shifting (-, | No |
| L1_Prejoin        | Checkered Pixel Integration | L1_Plain/SpinAll | Pixel-wise Addition | ELU | 4 x 128 x 128 | No |
| L1_Postconv       | Conv2D | Conv Layer | L1_Prejoin, 3 x 3, 4, SAME, 1 x 1, ELU | 4 x 128 x 128 | Yes |

ACKNOWLEDGMENT

MTSAT-1R/2 observation data [32] were provided by the Center for Environmental Remote Sensing (CEReS), Chiba University, Japan. COMS-1 observation data [33] were provided by the National Meteorological Satellite Center (NMSC) of Korea Meteorological Agency (KMA). Computing resources supporting this work were mainly provided by the High-Performance Computing (HPC) Support Program for 2019 (xo) and 2020 (xo4wj) through the National IT Industry Promotion Agency (NIPA), South Korea. The authors would also like to acknowledge the support of NVIDIA Corporation with the donation of multiple GPUs used for the initial survey of this research, and the Korea National Supercomputing Center for technical support with supercomputing resources.

DATA AVAILABILITY STATEMENT

For enquiries about the satellite observation data, please refer CEReS [32] website (http://www.cr.chiba-u.jp/) for MTSAT-1R/2 observation and NMSC [33] website (https://nmsc.kma.go.kr) for COMS-1 observation.
