Clustering Cloud Types from the Geostationary Operational Environmental Satellite (GOES-16)

Nazia Farhat, Katherine Haynes, Jason Stock, Joe Strout

1. Introduction

The Geostationary Operational Environmental Satellite[1] (GOES-16) contributes to an over 40-year history of continuous satellite imagery and data on atmospheric conditions, with continuous high-resolution spatial coverage over American continents. GOES data is currently being used in a variety of weather and aviation forecasting applications; however, due to computational constraints, applying machine learning techniques to geostationary satellite imagery has primarily focused on either varying cloud cover over specific regions (e.g. Andersen & Cermak, 2018) or cloud cover at any vertical level (e.g. Qin *et al.*, 2019; Shang *et al.*, 2018; Wind *et al.*, 2010). Accurately identifying and predicting cloud types over large-scale regions, such as North and South America, is an important yet difficult problem that is only now able to be investigated using big data techniques. In this study, we investigate different cloud types seen by GOES using deep learning.

Identifying and predicting cloud types using GOES data has a wide range of uses for evaluation of climate change and weather process understanding. For example, knowledge of cloud types can help identify patterns of rainfall and their sensitivity to climate change. Understanding how cloud structures vary regionally would further our knowledge on the climate system and help predict localized responses to change. Determining cloud types directly from radiances could be used to replace or supplement the processing of radiances into cloud properties. Finally, accurate cloud identification would provide more accurate sky conditions to improve weather forecasts.

2. Methodology

We perform three unsupervised clustering techniques on a sample GOES-16 dataset, with the goal of comparing the predicted clusters to physically-based cloud types in order to determine how well these cloud types natively emerge from minimally-processed GOES-16 channel data.

2.1. Dataset Overview

For this study, we used satellite data from GOES-16 which covers North and South America along with the surrounding oceans. The Advanced Baseline Imager on GOES-16 takes measurements of the region covered at a time interval of 10 to 15 minutes. Data is obtained in 16 different channels (7 visible and 9 infrared), which have spatial resolutions varying from 0.5 to 2 km. We averaged all of the channels to the same 2 km by 2 km grid.

To avoid seasonal impacts, we analyzed data from October 27-November 30, 2017 (6019 files) for a total data size of 2.1 TB. From this dataset, we used a smaller oceanic subset in the Eastern Pacific Ocean just off the southwest coast of Costa Rica, from 3° - 9° N and 85° - 91° W. This subset includes daytime retrievals only (9 AM to 3 PM LST) to avoid solar zenith angle impacts, and has a size of 14 GB. The dataset is divided into 20x20 pixel images that will be used for the clustering, and the truth for each of

[1] https://www.goes-r.gov/mission/history.html

these images is the most occuring cloud type. To further aid processing and investigate feature importance, rather than using all 16 channels we conduct seven experiments using different channel combinations: EX0) 1; EX1) 3, 13; EX2) 1, 2, 3; EX3) 1, 5, 8, 11; EX4) 8, 9, 10; EX5) 1, 2, 3, 8, 9, 10; and EX6) 2, 8, 13.

We supplemented the 16 radiance channels with information about cloud types derived from the Clouds from AVHRR Extended (CLAVR-x) algorithm (Heidinger et al., 2019). These cloud types are used as the truth in our cluster evaluation, where each 20x20 scene is labeled as the most occurring cloud type from the individual pixels. The distribution of cloud types over our subset are shown in Figure 1. In the Eastern Pacific Ocean, six different types occur during our study period. Nearly half of all scenes contain water clouds, followed by clear (23%) and fog (13%). Supercooled water, cirrus, and overlapping clouds each occur in less than 7% of the scenes.

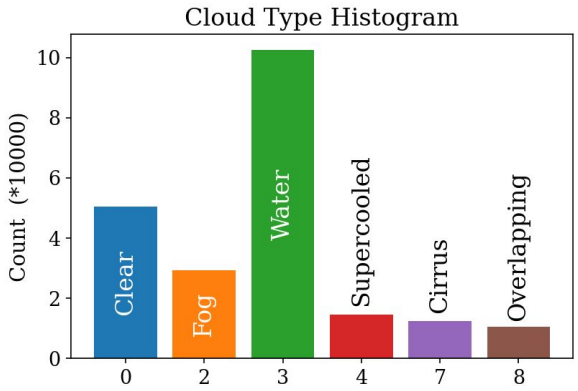


Figure 1: Cloud type histogram for the Eastern Pacific Ocean subset from October 27-November 30, 2017.

2.2. Approach

***k*-means Complete** A simple *k*-means clustering was performed on the top-left pixel of each image in the EX6 data set, to total 219,904 data points. Analysis was done in Scala using Apache Spark and Spark's MLlib library[2], with data stored in the Hadoop Distributed File System. 13 compute nodes were used with 4 workers each, for a total of 52 worker threads. The number of *k*-means clusters was varied from 2 to 10, and the KMeans algorithm was allowed to settle for 20 iterations (with follow-up experiments used to verify that the solution did not change significantly with up to 2000 iterations). Results were evaluated using the *within-set sum of squared errors* (WSSSE) measure.

Figure 2 shows how the WSSSE varied with the number of clusters. The "elbow" in the resulting curve is estimated to be at k=4, at which point the WSSSE was 972.7 (reducing slightly to 971.9 when the algorithm was run for 2000 iterations).

[2] https://spark.apache.org/mllib/

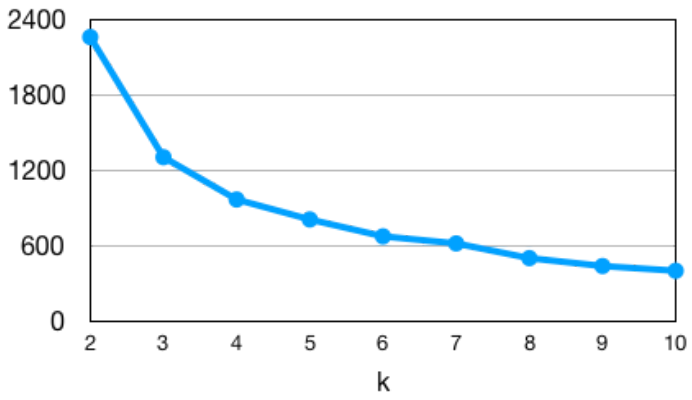


Figure 2: Within-set sum of squared errors (WSSSE) as a function of number of clusters (*k*) in the baseline *k*-means clustering of data from channels 2, 8, and 13. Vertical axis: WSSSE.

***k*-means + Autoencoder** A Simple Autoencoder (SAE), first introduced by Rumelhart *et al.* (1986), is a fully connected neural network with the ability to learn a compressed representation of data. Dimensionality reduction is performed by learning a latent vector with fewer units than the input and output by training the model to predict its own input via backpropagation. Improvements to the SAE architecture were later introduced by Guo, X *et. al* (2017) with a Convolutional Autoencoder (CAE). The structure uses convolutional layers for downsampling, a fully connected autoencoder to generate the latent vector, and convolutional transpose layers to upsample. This provides more effective results when dealing with images.

An end-to-end autoencoder is trained using the raw GOES channels as the input and compared to itself as the target. To cope with the large quantity of data samples we utilize the distributed PyTorch library for training. Not only does this enable seamless comparisons of various architectures with support of automatic differentiation, but also vastly improves training time by distributing computations among a cluster of nodes. Given a trained model, we use the weights of the latent vector parameterized by the encoded input samples to generate new data points represented by the salient features in the data. Thereafter, we perform *k*-means clustering over these new points to identify the underlying characteristics. Packages from Scikit-learn simplify this task and enable us to leverage a variety of different evaluative metrics.

Together, there are three primary steps involved in this approach, including: (1) developing a neural network that most accurately represents the data, (2) clustering the latent vector for all samples using *k*-means, and (3) evaluating cluster performance with changes in input channel combinations.

Deep Embedded Clustering Algorithm Deep Embedded Clustering is an unsupervised deep learning clustering algorithm, first introduced by Xie *et al.* (2016). It maps a high dimensional data space to a lower dimensional feature space and learns cluster assignment by optimizing the clustering objective comparing with an auxiliary distribution of the data points derived from their soft cluster assignment at a periodic interval.

This model network consists of an autoencoder and a custom soft layer and trained in two phases. In the first phase the autoencoder is pre-trained to minimize the Mean Squared Error between the output of the decoder and the input images themselves. In the second phase a separate branch is created and trained with the encoder portion of the pretrained autoencoder and a custom layer, in which the number

of hidden neurons is strictly kept to the number of clusters explored, in order to do a soft cluster assignment of the data points. The weights of the encoder are initialized from the pretrained weights and custom layer's weights are initialized from the cluster centroids derived from the *k*-means clustering of the compressed latent features of the pretrained autoencoder. Care is taken to keep the number of compressed latent features equal to the clusters-number in order to extract the distinctive features for cluster assignment of the data points. An auxiliary target distribution is derived from the soft cluster assignment and this periodically updated distribution is utilized to optimize KL Divergence loss of the cluster assignment.

Our project incorporates the improved DEC model proposed by Guo *el al.* (2017) to do cloud cluster analysis, where a CAE is used to maintain the spatial distribution of the data points. Our model was run on distributed PyTorch to accelerate the training and the parameter tuning. The input images have been normalized over all the data points in the preprocessing stage before being fed into the DEC model.

3. Experimental Evaluation

To evaluate our experiments, we will look at the autoencoder network architecture and computational performance, as well as the performance of *k*-means clustering on the resulting latent vectors. This will be followed by a discussion of the DEC results.

3.1. Network Architecture

An initial step towards clustering results from the autoencoder is to create a model that most accurately represents the data. We assess the accuracy of a model by measuring the root-mean-squared error (RMSE) between the input samples and output of the network, as defined by:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}$$

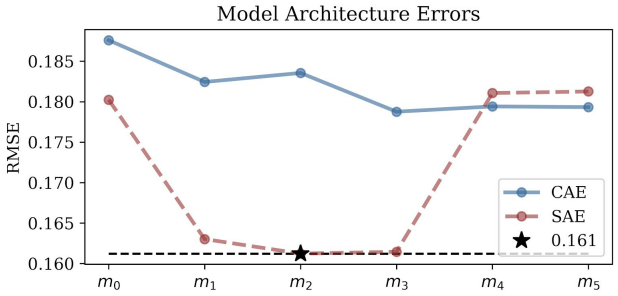
where a perfect model has a RMSE = 0. It is important that the input and output of the network are in the same units to correctly measure the error. Therefore, we scale the input for each channel to be between [0,1] by dividing by the maximum value for each channel. Then we normalize the data with a mean of zero and standard deviation of one. The final layer of the network uses a hyperbolic tangent (tanh) to map the output between [-1,1], thus, yielding the same range as the input.

For each training step, the RMSE is minimized using Adaptive Moment Estimation (Adam) to improve convergence and performance of training. We initialize Adam with the default beta parameters from Kingma, D.P. (2014), and tune the learning rate to a value of 0.001. Each epoch iterates over the total 219,904 samples with a batch size of 512. Therefore, a given node in a cluster of six will pass over a subset of data in 72 batches in a single epoch for a total of 10 epochs. A thorough analysis of how the batch size and number of nodes impact training times are found in section 3.2.

Modifications to the SAE are relatively simple with changes to the structure of the hidden layers. Initial experiments found that utilizing the tanh activation function after each layer produced the most stable results. Therefore, we focus on experimenting with the number of layers and respective number of units. The CAE also requires tuning for the dimensions of the hidden layers with addition to the size and strides for the convolutional filters. This model is constructed with more complex layers, including: batch

normalization after each convolution to improve gradient flow and stability during training, as well as using leaky rectified linear units for activations and fully connected layers to generate the latent vector.

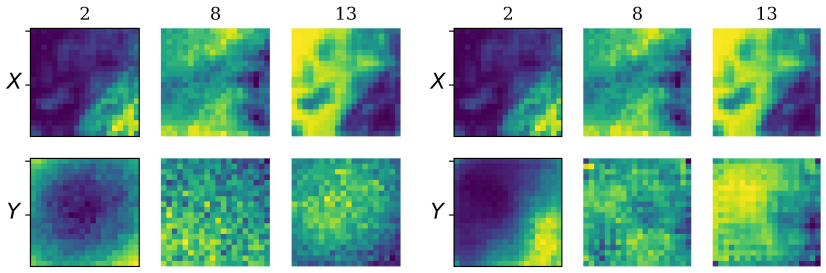
Figure 3 illustrates the change in RMSE for different structures when trained on the visible channels. The models with the smallest error are the best candidates to use for clustering. A clear trend for the CAE exists whereby having a deeper network structure that compresses each sample more towards the latent vector performs better. However, the errors observed for the SAE show no evidence for how changes in the hidden layers affect performance.



	SAE	CAE	
	Hidden Layers	Hidden Layers	Kernels and Strides
m_0	[512, 128, 64, 32, 16, **3**]	[4, 8, 10, **3**]	[(3, 1), (4, 2), (4, 2), ()]
m_1	[512, 128, 32, 8, **3**]	[4, 8, 10, **3**]	[(4, 1), (5, 2), (3, 2), ()]
m_2	[256, 128, 64, 16, **3**]	[4, 8, 10, **3**]	[(5, 1), (4, 2), (3, 1), ()]
m_3	[256, 64, 16, **3**]	[10, 10, 10, **3**]]	[(3, 1), (4, 2), (4, 2), ()]
m_4	[128, 32, 16, 8, **3**]	[10, 10, 10, **3**]	[(4, 1), (5, 2), (3, 2), ()]
m_5	[64, 16, 4, **3**]	[10, 10, 10, **3**]	[(5, 1), (4, 2), (3, 1), ()]

Figure 3: Minimum achieved RMSE with different model architectures. All layers are reflected equally around the latent vector dimension.

A challenging aspect of training the SAE model is stabilizing the large number of weights and biases composing the network. For example, the SAE m_2 model, as seen in figure 3, consists of 10 layers with a total of 700,595 trainable parameters. Whereas the best performing CAE model m_3 has nine layers and only 8,662 trainable parameters. This suggests that while the CAE has a more complex structure it may also have better stability and learn the structure of the data more accurately. As a result we observe that the RMSE for the SAE models fluctuate with slight structural changes. To further evaluate this observation we visualize the output of both networks to find that the SAE generates images with more noise, and the CAE resembles the ground truth more accurately (Figure 4).



(a) Simple Auto Encoder (b) Convolutional Auto Encoder

Figure 4: Model output Y for a given sample X across channels 2, 8 and 13.

3.2. Computational Performance

A significant amount of time spent in evaluating our approach was allotted to the overhead of training our networks. To better understand the scalability of distributed PyTorch we train six identical CAE

across varying numbers of nodes and capture the total training and data loading durations. The cluster is composed of multiple HP-Z420-XeonE5-2650v2 workstations, each with a 8x2.5GHz processor with 32 GB of memory. All models are run on the CPU using the PyTorch Gloo backend architecture. Each node is responsible for a subset of the data as well as averaging its weights by broadcasting to other nodes in every iteration. Therefore, the greater number of nodes used, the smaller the subset size at the expense of greater network communication. By scaling to six nodes we obtain a 2x improvement with 30 minute duration (Figure 5).

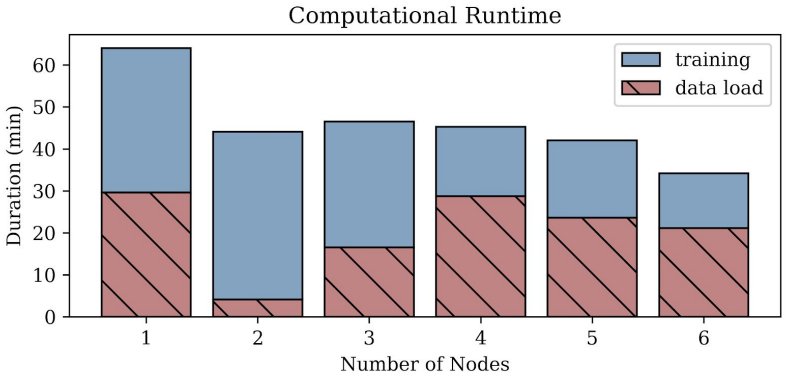


Figure 5: Data loading and training duration among all steps using distributed PyTorch across a cluster of varying number of nodes.

3.3. Clustering Results

We performed *k*-means clustering on the latent vectors produced by the autoencoder and evaluated the results by analyzing the optimal number of clusters, the performance compared to the true cloud type, and the distribution of the types in each cluster.

Optimal Number of Clusters As done for *k*-Means Complete, we evaluated the latent vectors produced by the autoencoders using the WSSSE measure (Figure 6 A), finding similar results. For all of the experiments, the sharpest "elbow" appeared at either 3 or 4 clusters. In addition to the WSSSE measure, we evaluated the average Silhouette Coefficient. For this metric, a score of 0 indicates overlapping clusters, while a score of 1 indicates highly dense clustering. All of the experiments have the highest value for either 2 or 3 clusters and decrease with increasing clusters. The score ranges from 0.3 for EX5 to 0.8 for EX4, and the scores tend to be higher using the SAE architecture, with the exception of EX6.

To explore the domain space of the latent vectors and further investigate the patterns seen in the average Silhouette Coefficient, we created silhouette plots for SAE EX4 (highest average Silhouette Coefficient), CAE EX6 (average), and CAE EX5 (lowest) (Figure 7 first column). Starting with SAE EX4 (Figure 7 top row), the two clusters are extremely imbalanced, with the majority of scenes belonging to a single cluster. Visualizing this in 3-D (Figure 7 middle column), the latent vector space is predominantly linear. Separating the clusters by opposite ends of the domain provides separability; however, noticing that the maximum range for the three components of the latent vectors is 0.002, this likely indicates the autoencoder was unable to differentiate the scenes in this experiment, likely indicating poor performance compared against cloud type and a lack of information in the selected channels.

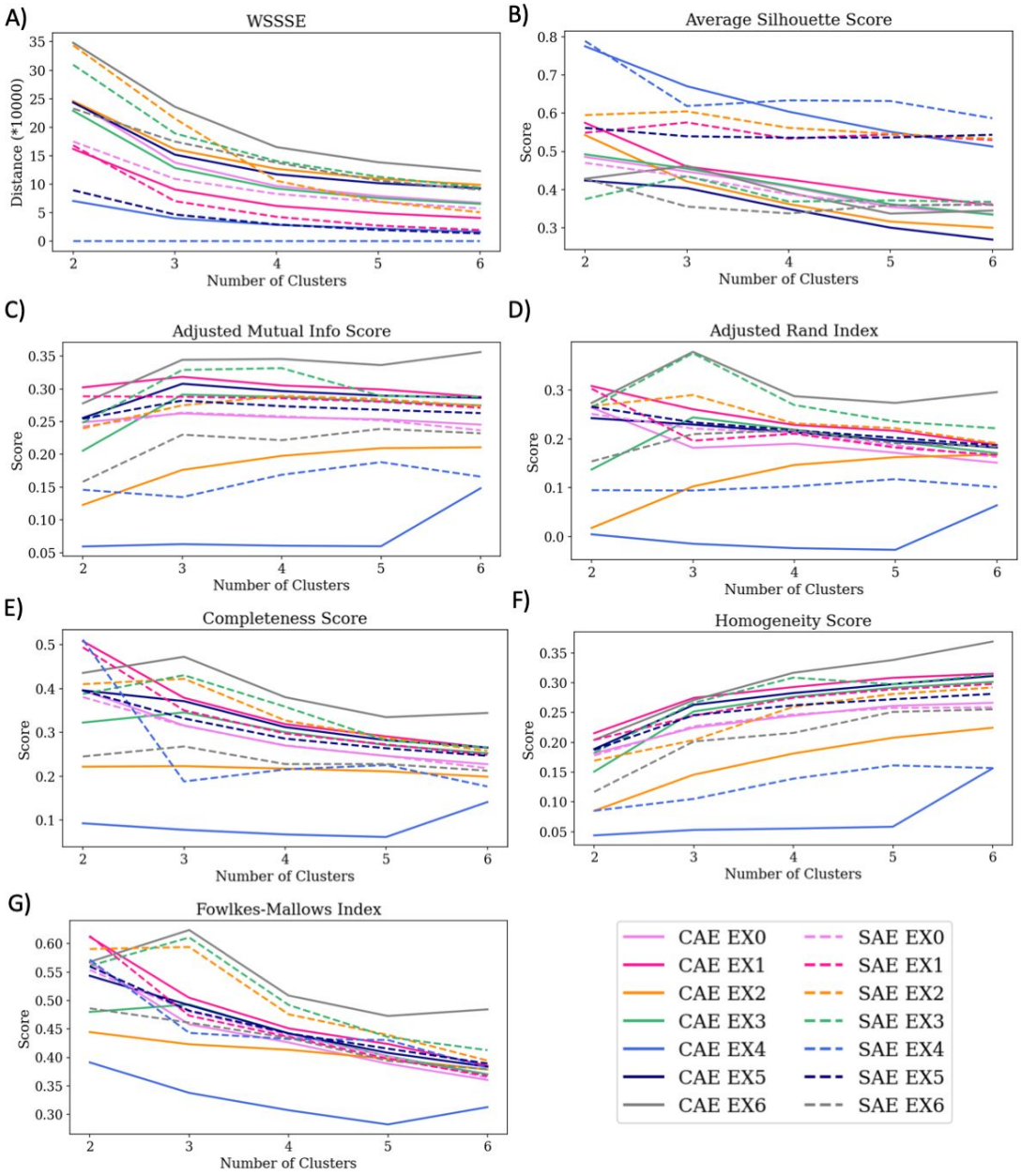


Figure 6: Cluster performance for each of the experiments using seven different scoring algorithms.

Moving to CAE EX6 (Figure 7 middle column), the clusters are still imbalanced; however, the 3-D visualization shows that the ranges of the latent vectors is much larger (~4), perhaps indicating a more meaningful split between the clusters. Illustrating this in a different way, we further reduced the latent vector space to 2-D using principal component analysis (PCA) and overlay the individual samples (black dots) and cluster centroid (white X). This helps to show this domain is no longer linear, but instead is more expansive to separate features that may be helpful in categorizing the cloud types.

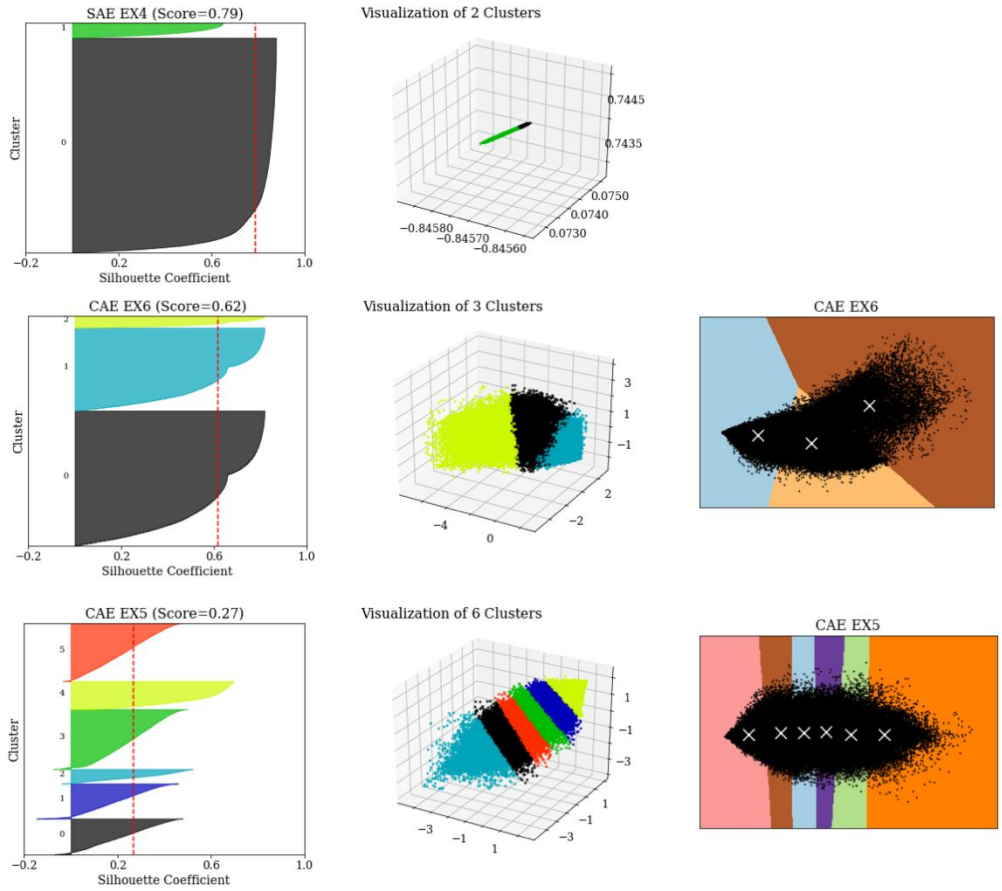


Figure 7: Silhouette plots (left column), 2-D visualizations (middle column), and PCA-reduced visualizations (right column) for three different experiments.

Finally, CAE EX5 is shown in the bottom row of Figure 7, again showing unbalanced clusters. Since there are 5 clusters, this imbalance may be acceptable due to the imbalance in the cloud types; however, the 2-D visualization again indicates a linear vector space. While the range is much greater than for SAE EX4, the linearity still likely indicates that this selection of channels is not optimal in fully capturing the range of scenes since we know that the cloud types are not linear. This is further illustrated in the PCA-reduced panel (Figure 7 bottom right), where the clusters are vertically aligned. This analysis indicates that 3 is likely the optimal number of clusters, while showing that the latent vector space from the autoencoder varies between experiments.

Performance To evaluate the clustering of the different experiments from the autoencoder-produced latent vectors, we analyzed five different metrics that utilize ground truth assignments, all of which have optimal values of 1. First is the Adjusted Mutual Information Score (Figure 6 C), which measures the agreement of the two assignments. None of the experiments have a high score that would be indicative of significant agreement, and CAE EX6 performs the best with a score of 0.35 for 3-6 clusters. As expected from the WSSSE and Silhouette Coefficient analysis, CAE EX4 performs poorly. All of the experiments have the lowest score for 2 clusters, which could be due to either poor performance or the mismatch between comparing two clusters against six different cloud types or intermixing of the cloud types between the 2 predicted clusters. Although the cause of the low score is somewhat difficult to discern, the relative performance between experiments is still viable in indicating that CAE EX6 produces

the clusters most closely matching cloud type, and we will investigate the mismatches for CAE EX6 in the next section to help elucidate the cause of the low score. The fact that the score does not increase with increasing clusters as the number of clusters nears the number of cloud types indicates minimal ability of the channel combinations in the experiments to directly match the six cloud types.

The second evaluation metric we analyzed is the Adjusted Rand Index (Figure 6 D), which measures the similarity between the true and predicted clusters. Results of this metric are similar to the previous metric, with values ranging from 0 for CAE EX4 to near 0.4 for CAE EX6 with 3 clusters and minimal change in score for different numbers of clusters.

The next two metrics we evaluated are the Completeness (Figure 6 E) and Homogeneity (Figure 6 F). Completeness measures if all members of a given class are assigned to the same cluster, and for all experiments this score decreases with increasing number of clusters. This shows that not all of a single cloud type is included in a single predicted cluster, and it is not surprising that it decreases with increasing clusters as the number of options for classification increases, lowering the probability of correct assignment. Homogeneity measures if each cluster contains only members of a single class, and for all experiments this score increases with increasing number of clusters. This is not surprising, since with two clusters some of the cloud types must be grouped together rather than directly matching; however, again relative comparisons are still valid to see which experiment best groups cloud types. The order of the experiments is the same for both metrics, with CAE EX 1 performing the best using 2 clusters closely followed by CAE EX6 using 3 clusters, and CAE EX4 the worst.

Finally, we analyzed the Fowlkes-Mallows Score, which gives an indication of the pairwise precision and recall. This shows the same pattern seen in the Completeness and Homogeneity scores: CAE EX1 with 2 clusters and CAE EX6 with 3 clusters perform the best and CAE EX4 performs the worst. For all but CAE EX6, the score decreases with increasing clusters, showing that the performance gets worse as the number of clusters increases. Since this metric is based on precision and recall, this shows that using more clusters will lead to more mismatches.

Across all scores, this analysis shows that the CAE EX 6 with 3 clusters is the best, most consistent performer, closely followed by CAE EX1 with 2 clusters. Looking at the performances between CAE and SAE, their relative performance varies with experiment. For EX0 and EX5, CAE and SAE performed similarly; for EX1 and EX6, CAE performs better; and for EX2 and EX4 SAE performs better. We hypothesize that this is due to the interaction of the channels in the experiments with the architectural strengths, and for future work it would be interesting to investigate this further.

Cloud Type Distribution Using three of the top performing experiments, we analyze how the different cloud types are distributed amongst clusters using histograms overlaid by cloud-type counts per cluster (Figure 8). Starting with CAE EX1 with 2 clusters, Figure 8 A shows that nearly all the clear (0), water (3), supercooled (4), and overlapping (8) scenes are assigned to single clusters, illustrating why this experiment has a reasonably high completeness score; however, both fog (2) and cirrus (7) are split between groups. Looking at what clouds are grouped together, the clear scenes are in a distinct group with roughly one-third of the fog and cirrus types, which makes sense meteorologically since fog is low and difficult to detect while cirrus can be quite thin resembling clear conditions. Nearly all of the types containing typically thicker clouds are grouped together in the opposing cluster. It is encouraging that this combination of channels shows some efficacy at distinguishing clear from cloudy conditions.

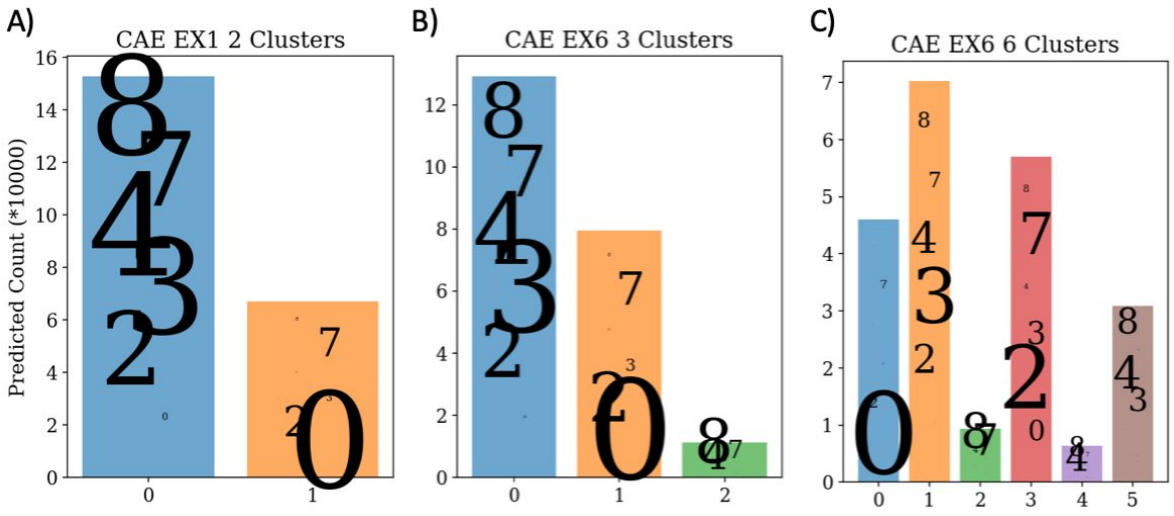


Figure 8: Cluster histograms overlaid by the number of cloud types per cluster as indicated by the size of the cloud type number.

CAE EX6 shows similar results (Figure 8 B) isolating clear conditions; however, it adds a third group that contains fractions of supercooled (4), cirrus (7), and overlapping (8). These cloud types are associated with ice, which is exciting because it indicates that these channels may have the potential to detect icing conditions. Although further analysis investigating the scenes is necessary to confirm, we hypothesize that clustering these channels may be attempting to separate clear, cloudy, and icing conditions.

Finally, we evaluated CAE EX6 with 6 clusters to investigate a direct comparison between clusters and cloud type. Clear conditions are separated out; however, the rest of the clusters contain mixes of cloud types. While the cloud types are mixed, the groupings do make some meteorological sense. Cloud types 2, 3, and 4 are all low-level cloud types that are frequently seen together, and 7 and 8 often occur in the same regions because of the way they are defined in the cloud type algorithm. Looking at the images associated with each clustering may provide further understanding to the cloud types being grouped together.

3.4. Deep Embedded Clustering Results

DEC analysis was attempted for this project to increase the purity of the cluster assignments, but due to time and resource limitations, it was not possible to optimize the model enough to reach our end goal. We tried basic DEC analysis on our data, but the results were not very encouraging. Improved DEC showed some promising results. However, it was still very difficult to run this model with all the data points as we were constantly running into out-of-memory issues. For this reason we chose only EX6 (channels 2, 8, 13) to train the network for 3 to 5 clusters and evaluated their Silhouette score. The best result was obtained for 3 clusters with a Silhouette score of 0.61.

4. Conclusions

We were able to successfully implement all three clustering approaches. Both *k*-means Complete clustering and *k*-means with autoencoder latent vectors indicated that three clusters are best to match cloud types, which was supported by an evaluation of various scoring metrics combined with an analysis

on the distribution of the cloud types per cluster. Analysis of the *k*-means clustering on the autoencoder-produced latent vectors shows promising results. In general, the CAE yielded higher results than the SAE architecture, having a larger latent vector space that led to better clustering. EX6 and EX1 were the top performing experiments, indicating that channel 13 (infrared) combined with a visible channel (2 or 3) may be skillful at separating clear from cloudy conditions on the native GOES-16 data, and using the autoencoder on the combination of channels 2, 8, and 13 may have some potential to further separate icing conditions. Finally, the deep embedded cluster algorithm was able to detect that clustering was sensitive to channel 13, which was the channel in common to the two highest performing experiments with the autoencoder. Unfortunately, limitations in computing resources and time limited our analysis from this algorithm; however, our implementation and preliminary results show potential.

Member Contributions

Nazia Farhat	Developed Deep Embedded Clustering model on distributed PyTorch; trained and evaluated different networks; assisted in writing final report.
Katherine Haynes	Collected and pre-processed the data; performed *k*-means analysis on the autoencoder latent vectors; assisted with writing & editing of the final report.
Jason Stock	Developed the autoencoders using distributed PyTorch; contrasted different models and evaluated performance; assisted with writing & editing of final report.
Joe Strout	Performed "*k*-means complete" experiment; participated in project planning and discussions; assisted with writing & editing of final report.

Bibliography

Andersen, H. & Cermak, J. (2018). First fully diurnal fog and low cloud satellite detection reveals life cycle in the Namib. *Atmos. Meas. Tech*., **11**, 5471-5470, https://doi.org/10.5194/amt-11-5461-2018.

Guo, X.; Liu, X.; Zhu, E.; Yin, J. (2017) Deep clustering with convolutional autoencoders. In Proceedings of the 24th International Conference on Neural Information Processing, Guangzhou, China, 14–17 **373**:382.

Heidinger, A., Walther, A., Botambekov, D., Straka III, W., Wanzong, S. & Li, Y. (2019). The clouds from AVHRR Extended (CLAVR-x) user's guide. Version 6.0.4. University of Wisconsin-Madison. https://docs.google.com/document/d/1NHuKBJr23i1k0ysihoyThg32e44l4CoDe5CjWydn0JI/edit

Qin, Y., Steven, A.D.L., Schroeder, T., McVicar, T.R., Huang, J., Cope, M. & Zhou, S. (2019). Cloud cover in the Australian region: development and validation of a cloud masking, classification and optical depth retrieval algorithm for the Advanced Himawari Imager. *Front. Environ. Sci.*, **7**:20, https://doi.org/10.3389/fenvs.2019.00020.

Rumelhart, D. E.; Hinton, G. E. & Williams, R. J. (1986), Learning Internal Representations by Error Propagation, *in David E. Rumelhart & James L. Mcclelland*, ed., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundation;* MIT Press, Cambridge, MA, **318**:362.

Kingma, D.P., Ba, J (2014): Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations.* arXiv:1412.6980

Shang, H., Letu, H., Nakajima, T.Y., Wang, Z., Ma, R., Wang, T., … & Shi, J. (2017). Diurnal cycle and seasonal variation of cloud cover over the Tibetan Plateau as determined from Himawari-8 new-generation geostationary satellite data. *Scientific Reports*, **8** (1105), https://doi.org/10.1038/s41598-018-19431-w.

Wind, G., Platnick, S., King, M.D., Hubanks, P.A., Pavolonis, M.J., Heidinger, A.K., Yang, P., & Baum, B.A. (2010). Multilayer cloud detection with the MODIS near-infrared water vapor absorption band. *J. Appl. Meteor. Climatol.*, **49**, 2315-2333. https://doi.org/10.1175/2010JAMC2364.1.

Xie, J., Girshick, R. & Farhadi, I. (2016). Unsupervised deep embedding for clustering analysis. *Proceedings of Machine Learning Research*, **48**, 478–487, New York, New York, USA, 20–22.