Modernized Deep Embedded Clustering (MDEC): From Machine Vision to Astrophysics

Jack Lawrence

Overview

The Modernized Deep Embedded Clustering (MDEC) is a framework and tool that I developed at UnitX as a Vision Technician in order to improve our capabilities to detect defects in our customers' products. The algorithm is comprised of two deep learning (DL) models, a convolutional autoencoder that maps images into an N-dimensional feature space and a clustering layer that learns to categorize this feature layer by associating the learned nodes with the input images [3]. Note that the size of the feature layer is equivalent to the number of clusters (aka categories or labels) that describe each image.

For example, this document describes the algorithm applied on the MNIST digit dataset, a collection of handwritten digits from 0-9. The algorithm splits the dataset into training and validation sets, studies the training images, and then groups the images into a category (0-9) based on their lower-dimensional representation [4].

However in it's practical application, this algorithm is used to cluster masked defects in larger images of manufactured parts. A more in-depth explanation of the motivation and application of this algorithm is provided below.

Required Dependencies

- `python 3.11.8`
- `tensorflow 2.19.0`
- `numpy 1.26.4`
- `pandas 2.2.2`
- `scipy 1.13.0`
- `matplotlib 3.8.3`
- `scikit-learn 1.4.1.post1`

I recommend working in a virtual environment with libraries installed using '`pip`'. The '`conda`' and '`conda-forge`' package managers tend to have outdated packages that don't always work well with tensorflow.

Why did I develop this?

Throughout my time at UnitX, I have been tasked with developing machine vision solutions prospective customers (usually manufacturers). This process begins with a Proof of

Concept (POC) where I create a small demonstrative inspection station, involving lights, camera, robotic arms, and/or anything else we have available to create the best custom, automated solution.

The next step, and the reason most customers come to UnitX specifically, involves training a neural network to detect defects in the images we capture of their defective parts. Though this process is very repeatable across customers, parts, and applications, it relies predominantly on two main factors:

- The quality of the training data (ie the images)
- And the consistency of the labeled defects

Because we measure success not by the number of units sold, but the number of units successfully deployed, we rely on the percentage of FAs (False Acceptances) and FRs (False Rejections), to validate the success of our integration. This means, that a working, deployed, system must falsely accept and falsely reject no more than approx 1% of the parts inspected during a deployment (typically 300-500 total parts).

So what does that have to do with clustering? Well, recall the two factors I mentioned. The first is dependent on the quality of the solution I develop (ie: how well I design a camera and light system, which is something I've gotten good at). The second factor, the consistency of labeling, is trickier to perfect, mostly due to the fact that defects of the same type can vary widely across applications, or even the same part. A scratch on the top of a rotor looks much different from one on the side teeth. Discoloration on a plastic part varies in color and texture depending on the cause of the defect. And more often than not, most customers don't even know what exact defects they're looking for or how to diagnose their defects; instead they care more about the size, location, and frequency of the defect.

This is why I decided to develop the MDEC algorithm. I allows me to label defects of various geometries, textures, and colors that I know are of concern to the customer, and then let the algorithm decide how to best categorize them. This way, when I train the separate detection model, I provide it the best chance at finding defects in new samples, because the defect categories it looks for have been expertly outlined by the MDEC DL algorithm. It is also important to note that the detection algorithm searches for defects in a feature space, much like the MDEC algorithm, so their workflows connect seamlessly.

In fewer words, the MDEC algorithm is an unsupervised DL tool that allows me to cluster or categorize defects in images, which can then be used to train a supervised detection model in order to maximize our chance at catching defects.

How is this relevant to astrophysics research (in general)?

Simply put, it's an unsupervised data classification algorithm that is extremely generalizable. The key here is the word 'unsupervised', meaning it does not rely on a specific, or even sizable, training set in order to work successfully, nor does it require labeled data. In it's most minimal form, the MDEC algorithm studies single or multi-dimensional tensors

for trends that are not easily interpretable by humans, which it uses to draw conclusions about the input data. I'll provide an astrophysical hypothetical.

Imagine you have an apparently uniform image of a cloud of gas. The image could also be a dataset that describes local densities, temperatures, or even scattering coefficients for a particular wavelength. It may look like a bunch of numbers, or it may even be interpretable to some degree. But it might be difficult to understand, or you may want a second opinion, a more detailed analysis, or a tool to verify your conclusions. So you segment the data tensor into smaller components, keeping track of where you draw your masks, and feed it to the MDEC algorithm.

The algorithm groups these segments, perhaps into 12 different groups. Upon further inspection of these grouped areas, you uncover a larger, less obvious structure to the cloud of gas. You may find that the gas is actually a collection of smaller clouds, or that the gas is moving in a particular direction, or that the gas is utterly and entirely uninteresting!

The goal of this algorithm is not to replace human interpretation, to write a paper for you, or to even be useful in all scenarios. But it is an example of artificial intelligence as a useful scientific tool that can lead us in new, sometimes surprising directions.

The state of the algorithm

The idea of Deep Embedded Clustering (DEC) is not a new idea, and it remains an open research field. For me and my work, it is an ongoing project that I develop daily (though unfortunately I am unable to share all aspects of my progress due to an NDA). Regardless, the algorithm I have designed is a unique interpretation of a project started in 2015 by Dr.Junyuan Xie et al. that has since been improved (IDEC), varied (VaDE), and now modernized (MDEC).

The qualifier "Modernized" was chosen first and foremost to emphasize the flexibility of my algorithm compared to its predecessors, but also because it uses the most up-to-date packages available for tensorflow. The algorithm works with the current line of GPUs (5000 series), though personally I run it with a 4000 series. Notable improvements in my algorithm include:

- a convolutional autoencoder that preserves 2D data structure, with max pooling, as it translates a image to the feature space [3]
- more metrics for both the autoencoder and cluster training [2][1]
- increased visibility into the clustering progress through epoch graphics and gifs [4]
- a learning rate scheduler that decreases the likelihood of cluster convergence on local minima
- the added option of augmenting input data (random rotations and contrast adjustments) for increased training variety
- expanded warnings and notifications for all processes (helpful for debugging)

- updated and reorganized logging with pandas
- and a 'test_station' script that allows the user to visualize live (or historical) metrics, as well try out and adjust the autoencoder parameters (start here, it's fun)

Most importantly, I have decoupled the autoencoder and the clustering algorithm, so that technically any autoencoder can be used with the clustering algorithm, as long as it was trained on data of the same size as is to be clustered (though this is intuitive).

Figures

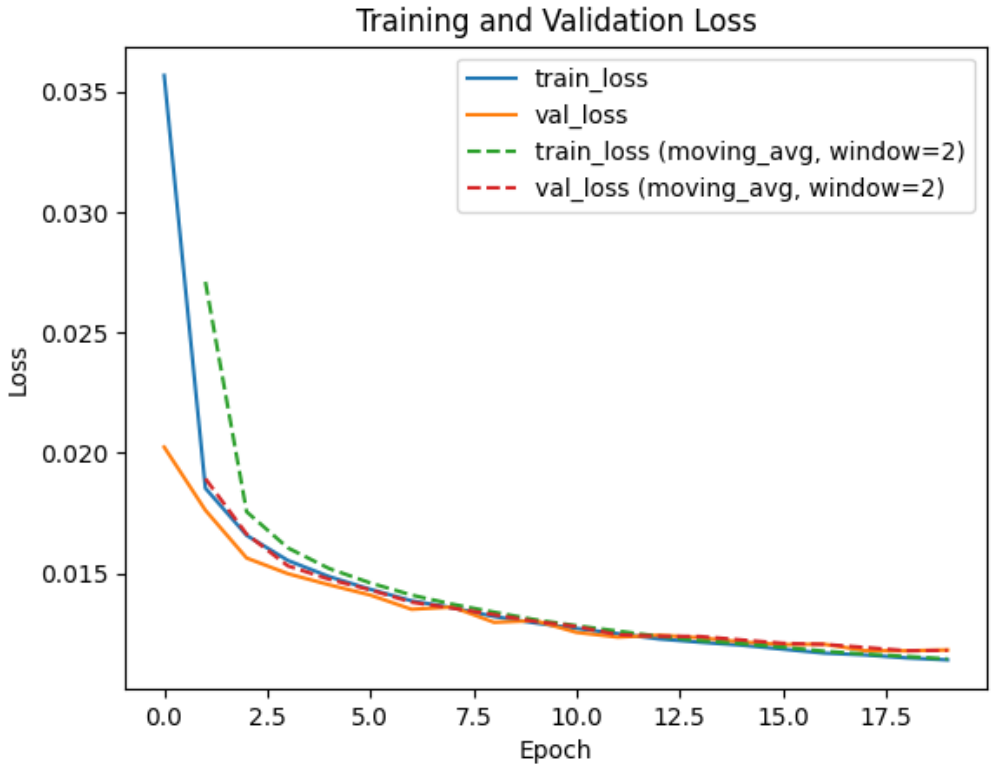


Figure 1: Here is an example of the metrics tracked and saved during autoencoder pretraining. The plot includes training loss, validation loss, as well as moving averages for both metrics.

Figure 2: This plot tracks the improvement of the clustering algorithm with three characteristic accuracy measurements: the Normalized Mutual Info (NMI) score, the Hungarian Assignment Algorithm (ACC), and the Adjusted Rand Index (ARI) score. The metrics also displays the clustering loss, reconstruction loss, and total loss as the model trains. Note: this particular graph describes a well-converging model with a max accuracy of approx. 79%.

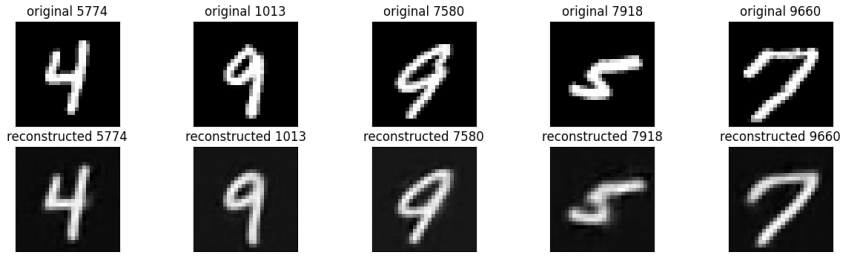
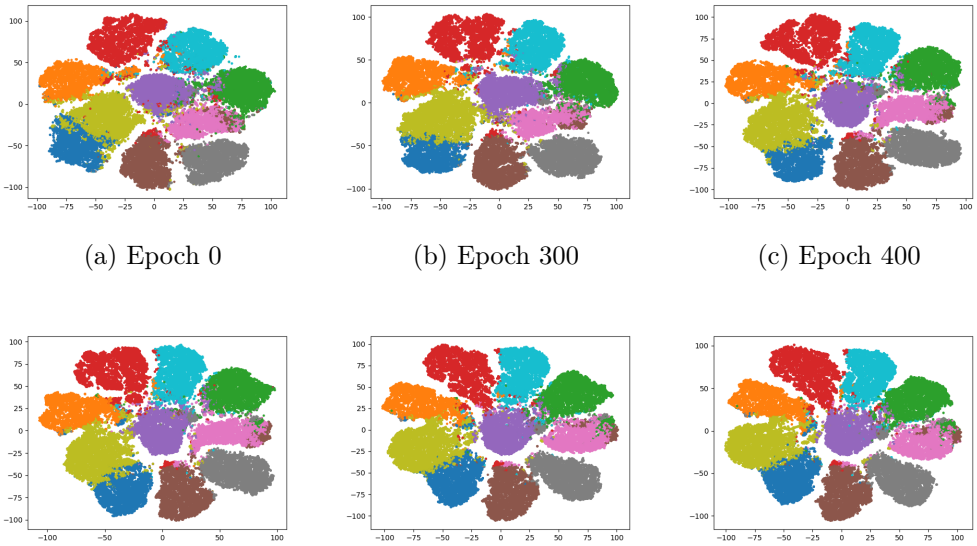


Figure 3: Example of autoencoder taking an input (top row), mapping it into a lower-dimensional space, and than reconstructing it (bottom row). Notice how the reconstructed image closely resembles the input image, though due to the random and imperfect nature of deep learning, slight variations indicate information loss during compression.



(a) Epoch 0 (b) Epoch 300 (c) Epoch 400

(d) Epoch 600 (e) Epoch 1000 (f) Epoch 1600

Figure 4: Example of clustering algorithm over time. The separation of clustering indicates learning and distinction. The algorithm takes between 10-15 minutes to reach this point; this describes approx 86% clustering accuracy!