Heuslerene Autoencoder

Justin B. Hart*[1,2]

[1]Department of Physics, Lehigh University, Bethlehem, PA 18015, USA
[2]Department of Physics, University of Florida, Gainesville, FL 32611, USA

Abstract

We present an implementation of the Elf Autoencoder framework to generate and evaluate residual convolutional autoencoders for classifying band structures in 2D Heusler compounds, using a ResNet-based autoencoder to extract latent features. We evaluate these models to identify those that yield the most meaningful latent representations for clustering. In addition, we develop a web application that allows rapid exploration of these band structure classifications.

1 Introduction

In this article, we explain our implementation of the Elf Autoencoder described by Pentz et al.,[1] which we used to cluster 546 Heuslerene band structures. The Elf Autoencoder is a framework for training and analyzing residual convolutional neural network (CNN)-based autoencoders on band structures. In this work, we use the framework to generate and evaluate a set of 216 autoencoders trained on 2D Heuslerene materials.

For this project, we first generated 546 band structure plots in a form readable to the CNN, using data from VASP EIGENVAL files. We then trained these residual CNN autoencoders over a parameter space of size 216. Subsequently, we used the generated autoencoders to extract latent space representations ("fingerprints", i.e., low-dimensional encodings of band structure feature) representing the structure of the band images found by the network. Finally, we applied clustering algorithms and projections to evaluate the data, using multiple validation measures to define the best autoencoder models and clustering algorithms.

With this data we were able to develop a web application, presented in Section 3, that allows users to visualize clustering groupings derived from all 216 trained autoencoders. This application allows users to quickly find groupings of materials and better explore the large feature space.

2 Methods

2.1 Band Structure Preprocessing

Our band structures are generated from two files: the VASP EIGENVAL output and a file containing the materials Fermi energy. The path traverses three high-symmetry points, as seen in Figure 1. For the autoencoder, only the graph itself is important, so the axis and other features are removed, as also seen in the figure. The image size is 224 by 224 pixels, leading to a total feature size of 50,176. For our band structures, we created three sets of images with windows of ± 1, 2, and 3

*Email: Justin.Hart@ufl.edu

eV from the Fermi energy. Our line width was selected to be 1 point as defined by the Matplotlib library. [2] All code is available in the `Preprocess/` folder of the GitHub repository.

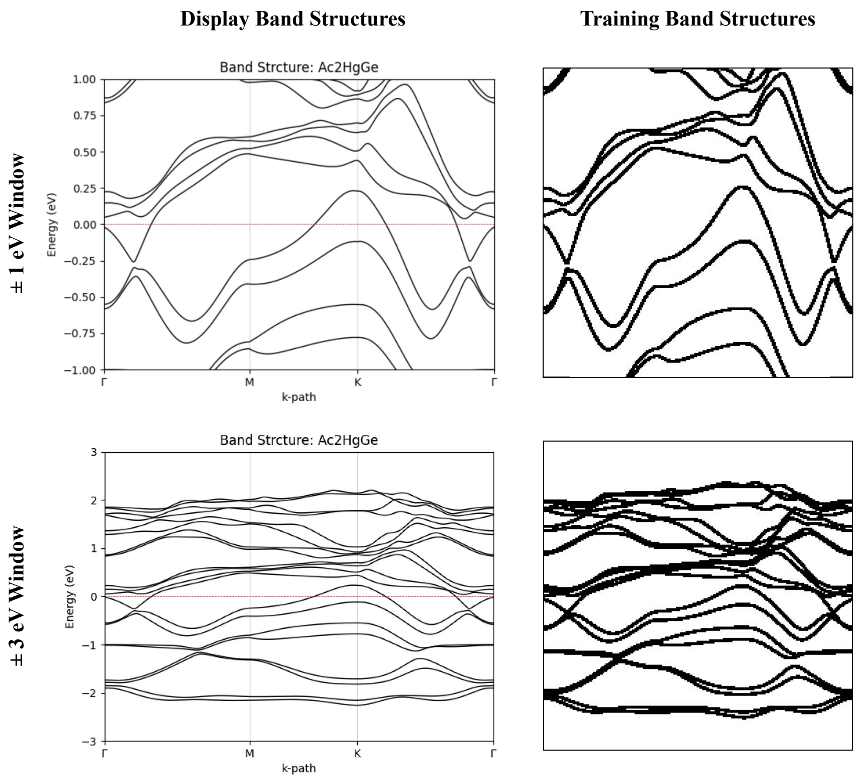

Figure 1. Image of generated band structures (left) and the training band structures (right). The Display Band Structures show an accurately labeled plot, while the Training Band Structures contain only information that is used for training the Autoencoder. This figure shows two band structures for the same material, with the top having a window of ±1 eV and the bottom having a window of ±3 eV from the Fermi energy.

2.2 Autoencoder

Using the Elf Autoencoder framework, we train 216 residual convolutional autoencoders. They are of the ResNet-18 and ResNet-50 architectures, with a latent space size of either 49, 98, or 147 features (an $x = 1, 2,$ or 3 dimensional latent space of size $7 \times 7 \times x$). For the training, we also explore a train-test split of size 0 (no validation set), 0.1, and 0.5. The window above and below the Fermi energy is varied between ±1 and ±3 eV, and we train the model through 30, 60, 90, or 120 epochs to have a range of under and overfitting. These will be referred to as the autoencoder hyperparameters throughout the paper. During training, we apply localized noise to all training images as in Pentz et al. in order to combat overfitting. An example of noise applied can be seen in Figure 2.

The ResNet Architecture is a type of residual CNN that has found widespread use in image recognition. Its design consists of connected convolutional layers, which decrease the image size and extract information about larger structures, sigmoid neurons, which allow for training, and residual connections, which help prevent vanishing gradients. In our case, ResNet-50 has 50 layers, making it a much deeper model than ResNet-18 with its 18 layers. When applying the autoencoder, the model takes in a 224×224 image, applies 18 or 50 layers of convolutions and sigmoid neurons, and reduces the size until there are $z \times 49$ neurons left, where z is the latent dimension. Then, it applies the same number of deconvolutions and neurons until the model returns to a 224×224 image. The binary cross-entropy loss is then calculated between the resulting image and the original image, and the goal of further training is to minimize this value. After training is complete, we can extract a fingerprint of the image by applying the first half of the network and noting the values output. We can visualize the effectiveness of the encoding by applying the second half of the network. The belief is that the neural network will encode important structures in the band which would not be seen in regular clustering. [3]

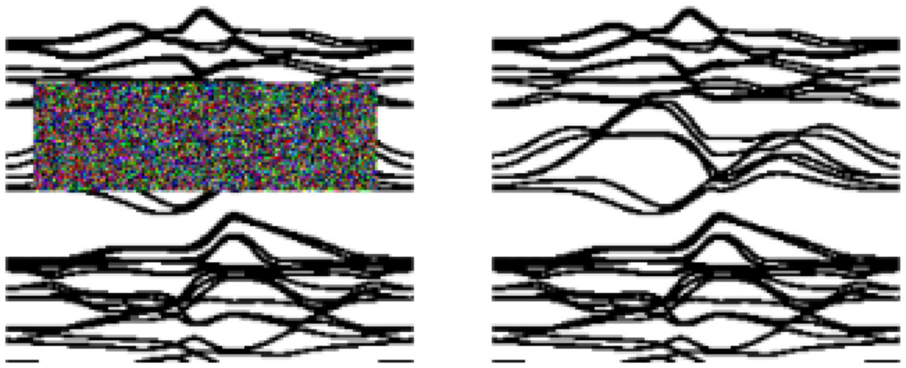

Figure 2. Image of noise applied to training data in order to prevent overfitting. All band structures in training have this localized noise applied, with the hope that it will prevent overfitting of the model. Overfitting can still occur in more deeply trained models, however, which can cause the fingerprint to contain less structure data in favor of finely tuned points for each band structure.

The training and test loss for each training can be found in the supplementary materials. This shows how well different models are able to reconstruct the original data, as well as how overfit they are towards their training data. If the training and validation loss are similar values, it implies the model is well fit. Our implementation is based on the Elf Autoencoder framework introduced by Pentz et al., with code adapted from their publicly available repository. Their code, further, is adapted from a GitHub repository by Horizon2333. [1, 4]

2.3 Clustering

To evaluate the auto encoding and to extract clusters, we examined 3 clustering algorithms: Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN), K-Means, and Gaussian Mixture Model (GMM). [5–7]

HDBSCAN is a density-based clustering algorithm, meaning it searches for clusters with high density, without assuming cluster shape. Notably, unlike the other clustering algorithms, HDBSCAN will select some points as 'noise' and will not put them into a cluster. This generally prevents the clustering of materials which are not together, something that can happen in K-Means and GMM. There are three main parameters that affect the clustering: the minimum cluster size, the minimum sample size, and the cluster selection method. The minimum cluster size is the smallest grouping which can be considered a cluster. The minimum sample size is a parameter affecting how conservative the clustering is, with higher values leading to more conservative clustering. The cluster selection method can either be 'excess of mass' or 'leaf'. Excess of mass generally results in one or two large clusters and a few smaller clusters, while leaf produces many small homogeneous clusters. When selecting HDBSCAN parameters, we searched over the minimum cluster size and minimum sample size to find the best model and implemented 'leaf' for the cluster selection method as its smaller, more homogeneous clusters gave us a better understanding of the band groups. As with Pentz et al., we used the Minkowski metric with $p = 0.2$. To implement, we used the Python HDBSCAN library. [8]

K-Means and GMM are the two additional clustering methods employed. Unlike HDBSCAN, these algorithms require the user to specify the number of clusters in advance, which can lead to underfitting or overfitting depending on the choice. K-Means partitions data by minimizing within-cluster variance, typically using Euclidean distance. GMM, by contrast, models each cluster as a Gaussian distribution, incorporating covariance information and allowing for more flexible cluster shapes. Both methods were implemented using the Python scikit-learn library [9].

All clustering is done on the autoencoded fingerprints (of size 49, 98, or 147). Then, for visualization, the fingerprints are projected on a Uniform Manifold Approximation and Projection (UMAP) embedding, explained in Section 2.4.1, colored with the labels found by clustering. In the end, we chose to focus on HDBSCAN as we found that it produced the most realistic clustering. As can be seen in Figure. 3, K-Means is required to place every point in a cluster, leading to large clusters that do not generally group similar band structures together. GMM produces a figure which is almost identical to K-Means. HDBSCAN, however, labels points as noise allowing for more meaningful clusters of similar band structures.

2.4 Clustering Validation

We used two main methods to validate the accuracy of our clustering: closeness in a UMAP projection, and validation scoring. Together, these allow us to understand how well fit the clusters were and find numeric values that allowed us to select a 'best' model.

2.4.1 UMAP Projection

To better visualize the data and its clustering, a UMAP projection was used. [10] These projections have found widespread use for visualizing data, due to their ability to preserve both local and global distances when they project data from a higher dimension to 2D. Hence, they provide an excellent source for spot-checking clustering validity. In our case, the clustering algorithms mentioned above are applied to the high dimensional fingerprint. These fingerprints are then projected from 49-, 98-, or 147-dimensional latent spaces down to 2D using UMAP, where we then color the points based on the higher-dimensional clustering. As can be seen in Figure 3, the clusters are generally near each other in the UMAP projection for both models, meaning the two methods of clustering are in agreement.

2.4.2 Scoring

There are numerous ways to score clustering, ranging from silhouette score and Scatter-Density between-within index (S-Dbw) for general clustering to more specialized scores like the Density-Based Clustering Validation (DBCV) score designed for scoring algorithms which make less assumptions about the shapes of a cluster. In this analysis, we focus on the S-Dbw index and DBCV. Additionally we calculated the mean Adjusted Mutual Information score (AMI) between models to find the model which has the most representative clusters in the ensemble.

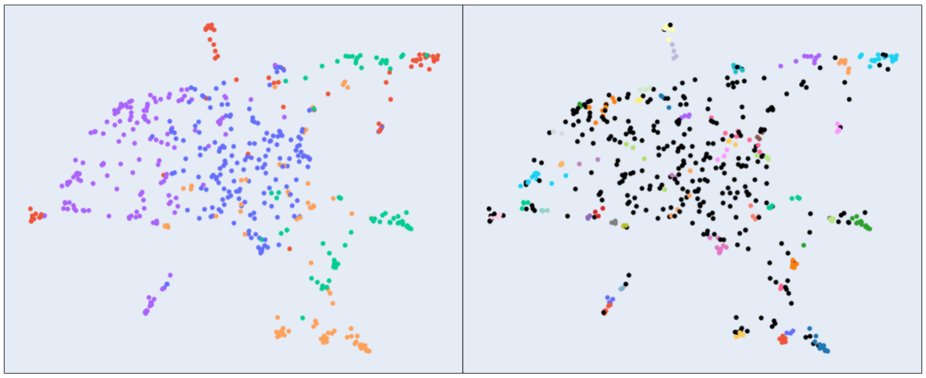

Figure 3. On the left is the K-Means clustering with 5 clusters, while on the right is the HDBSCAN clustering, with the minimum sample equal to 2 and minimum cluster size equal to 3. The black dots in the HDBSCAN clustering represent noise, that is points not belonging to any cluster. While a large number of points are classified as noise, losing information about some of these materials, the benefit is that the remaining clusters are highly informative with similar materials.

S-Dbw calculates both the scattering of the data and its inter-cluster density. Essentially, it is a score of both how compact each cluster is and how well separated the clusters are. Lower values are better, with a minimum of 0, and it has been shown by Liu et al. to be one of the more stable metrics. [11] We use the S-Dbw Python library for the calculations, and during them consider noise to be a single cluster. [12]

DBCV is a clustering metric designed specifically for density-based clustering algorithms, like HDBSCAN. It uses relative density connections between objects to calculate the score, instead of global metrics like S-Dbw. The score has a maximum value of 1, with higher score being better. [13] To implement, we used the Fast Density-Based Clustering Validation (DBCV) GitHub repository. [14]

AMI allows you to input two models' labeling, outputting a score between 0 and 1 on how close these two label sets are, with 0 meaning they are unrelated and 1 meaning they are identical. We begin with the assumption that all models contain useful clusters, an assumption that seems valid from our exploration of different clustering. We then calculate the pairwise AMI between every autoencoder model for a set of HDBSCAN parameters. From these pairwise calculations, we identify models with the highest mean AMI scores (those that most consistently agree with others). When calculating pairwise AMI, we consider both treating noise as a separate cluster and removing it entirely. Removing noise may lead to very few labels left, however, so must be interpreted with care. The method was implemented using the Python scikit-learn library [9].—

2.5 Clustering Search

To select the model further explored in the paper, we performed a nested grid search over all 1 eV models. For each HDBSCAN parameter in the range of minimum samples between 2 and 10, and minimum cluster size between 2 and 10 we calculated both the DBCV value and the mean AMI for the fingerprint of each autoencoder model clustered using the aforementioned HDBSCAN parameters. Then, we found the maximum of both scores for each parameter, and their associated model. The heat map of this search can be seen in Figure 4.

From this search, we found that the autoencoder with hyperparameters ResNet-50, Latent Space 3,

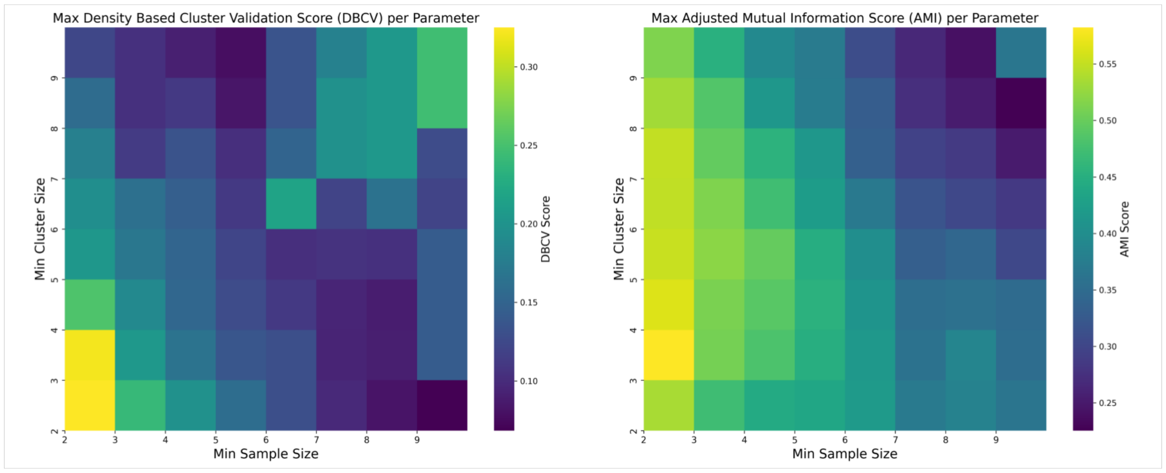

Figure 4. The maximum DBCV and AMI scores for each model at different Minimum Cluster Size and Minimum Sample Size values.

Training Steps 90, and Train-Test-Split 0.1 was consistently getting the highest DBCV scores for each selection in the grid space, as well as the highest DBCV score out of all models. Additionally, for each of the HDBSCAN parameters in the grid space, it was always in the top 10 for AMI scores, with the final parameters of Minimum Sample Size 2 and Minimum Cluster Size 3 having an AMI score of 0.4431 compared to the highest at 0.4567. This score indicates relatively good agreement with the other models, especially as noise was considered to be its own separate cluster. When noise was removed for the DBCV score, this AMI rose to 0.6593, with the highest at 0.6690, indicating very good agreement on which clusters were notable. Hence, this model both produced the most valid clusters and was a good representative of the clustering between models.

We further wished to self-consistently check that Minimum Sample Size of 2 and Minimum Cluster Size of 3 was the best option for clustering, so we ran another grid search on this best model to find where it had the highest DBCV and lowest S-Dbw scores. This grid search can be seen in Figure 5 and demonstrates these parameters as a good choice. Although a minimum cluster size of 2 would result in a slightly lower S-Dbw, there are slightly more well defined clusters with size 3, and a higher maximum AMI score, leading to our choice.

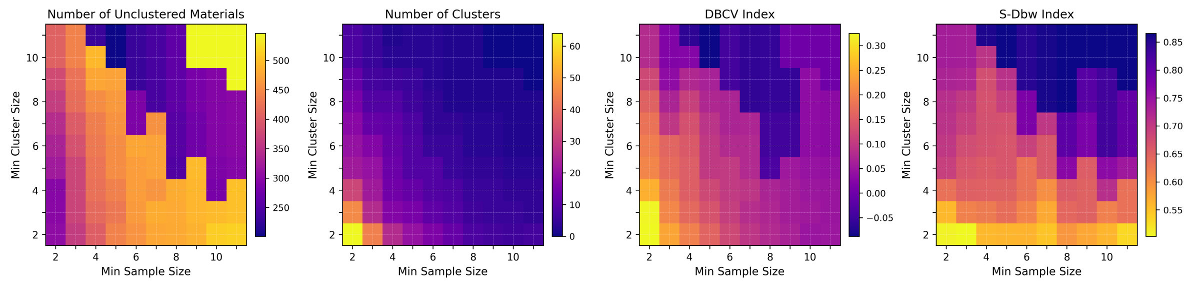

Figure 5. The number of unclustered materials, number of clusters, DBCV Index, and S-Dbw index for our best model over the different HDBSCAN parameters.

2.6 Secondary Clustering

After clustering based on HDBSCAN, there are noise points left over. To find larger clusters, we remove the noise and apply a second round of clustering to the data, using DBSCAN on the UMAP projected points. This allowed us to find larger structures in the data, as band structures which are relatively similar, but not similar enough for HDBSCAN, are clustered.

DBSCAN is a density-based clustering model, like HDBSCAN, where regions of similar density are clustered together. It has two parameters: epsilon and minimum samples. These regulate the cutoff value to create regions, and the number of points required to be a region. They were set to 0.8 and 5, respectively, based on the same clustering in Pentz et al. [1] The model we used for DBSCAN is from scikit-learn. [9] After clustering, we use the information to create circular, colored silhouette around each point in the new cluster. An example of the resulting secondary clustering is shown in Figure 6.

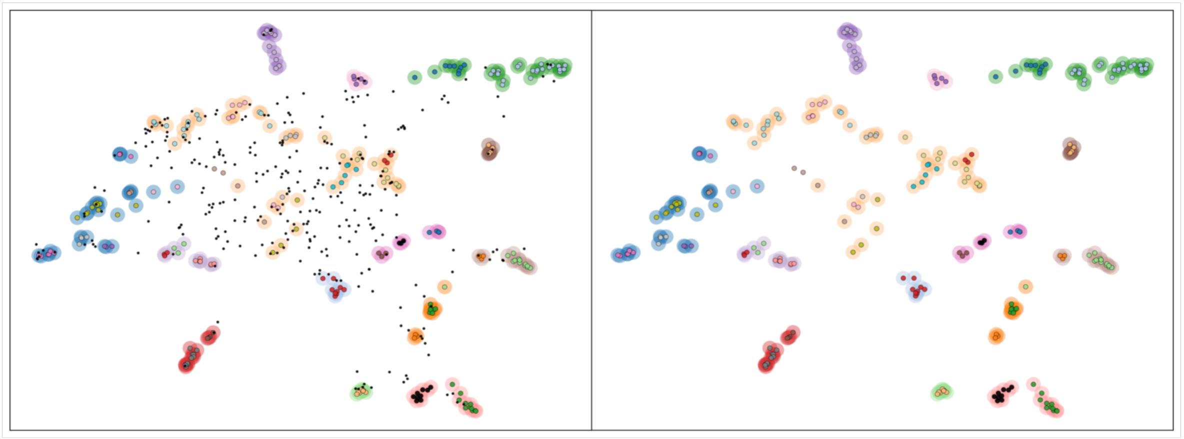

Figure 6. An image of the secondary clustering applied to the best model. On the left, you can see this model with noise, while on the right the noise points have been removed. The outer hue is the secondary DBSCAN clustering, while the dots maintain the color of their HDBSCAN original clustering.

2.7 Reclustering Noise

It was hypothesized that the noise indicated by HDBSCAN might still contain useful information about the band structures. To test this, we implemented a way to recluster noise on the web app, explained in Section 3. Through this reclustering, however, we found that the original pass of HDBSCAN did well indicating noise, as there was little to no useful clustering found in any attempts to recluster noise.

3 Web App

You can access the files to run the web app at its current GitHub repository. [15] To use, pull the git repository and follow the instructions in Section 3.1.

3.1 Running the Application

To run this code, the user must first be in the `heuslerene-explorer` folder. Once inside, they can install the required Python libraries by running `pip install -r requirement.txt`. Then, the user can run `python explorer.py` to start the app.

Once running, the user should navigate to `http://127.0.0.1:8050/` in their web browser. Here, you can select the autoencoder model and clustering algorithm you would like to see applied to the data, and then hover over the points to see the corresponding molecule and band structure. A welcome page explains the different settings the user may apply.

3.2 Implementation

For visualizing band structures, we have provided a web app `explorer.py` generated using the dash framework of Plotly. [16] This allows us to run a local webserver, running Python and HTML, to visualize the data. All information is loaded into .csv files, with each autoencoder model having two files: `{model_name}.csv` and `{model_name}_embedding.csv`. The first file contains the molecule names and their 49-, 98-, or 147-length autoencoded fingerprints. The second file contains the UMAP projection of the fingerprints, allowing for quick 2D plotting. In the web app, the user can select the model from dropdown menus, with each selection opening up a different .csv file to load the encoding.

Once the images are projected, the user then can select a clustering algorithm (HDBSCAN, K-Means, or GMM) and their parameters (minimum samples, minimum cluster size, and cluster selection method for HDBSCAN and number of clusters for K-Means and GMM). Finally, when the user mouses over specific points in the plot, the corresponding band structure will also be projected on the page, allowing users to see which categories are clustered together. These band structures are from the corresponding `image/` files, and have the same window size as the one selected by the user for the model.

Additionally, the user can search for different elements in the space using the 'Select Elements' dropdown. If the filtering is set to 'Inclusive,' all compounds containing any of the selected elements will appear. If the filtering is set to 'Exclusive,' only compounds containing all of the selected elements will appear.

There are also advanced options the user can use. The "Double Cluster" option allows the user to apply the process described in Section 2.6, applying a DBSCAN clustering to the non-noise points chosen by HDBSCAN. Additionally, they can apply the process described in Section 2.7 to the noise, being able to choose the parameters of the HDBSCAN model trained only on the noise values. Finally, they can hide both noise and non-noise points to get a better understanding of the clusters.

Note, there is also the ability to visualize the embedding of the unencoded images. This allows the user to see the difference between using and not using the autoencoder. However, for clustering specifically, the algorithms operate on the UMAP projection, not on the original data. This is a necessity due to the extremely high dimensionality of the image space (50,176 features), which would otherwise make clustering computationally infeasible.

4 Conclusion

In this article we have presented the implementation of the Elf Autoencoder for our Heuslerene band structures. We have also shown how our best model was chosen and validated. Finally, we have developed a web app implementation that allows for easy searching through our materials. All of this has allowed us to cluster the band structures, and better search our large dataset.

In the future, we aim to develop this framework into a streamlined toolkit that automates the entire pipeline. Based on user-specified parameters, the toolkit would generate and evaluate a suite of autoencoder models, identify those yielding the most meaningful clustering, produce relevant visualizations, and enable seamless integration into the web application.

Supplementary Information

Supplementary material, including autoencoder training losses, AMI scores, and code availability is provided in supplement.tex.

References

[1] Henry Kelbrick Pentz et al. "Elf autoencoder for unsupervised exploration of flat-band materials using electronic band structure fingerprints". In: *Communications Physics* 8.1 (Jan. 2025), p. 25. ISSN: 2399-3650. DOI: 10.1038/s42005-025-01936-2. URL: https://doi.org/10.1038/s42005-025-01936-2.

[2] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[4] Horizon2333. *imagenet-autoencoder: Auto-Encoder trained on ImageNet.* Accessed: 2025-07-25. n.d. URL: https://github.com/Horizon2333/imagenet-autoencoder.

[5] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. "Density-Based Clustering Based on Hierarchical Density Estimates". In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Jian Pei et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172. ISBN: 978-3-642-37456-2.

[6] Gareth James et al. "Unsupervised Learning". In: *An Introduction to Statistical Learning: with Applications in Python*. Cham: Springer International Publishing, 2023, pp. 503–556. ISBN: 978-3-031-38747-0. DOI: 10.1007/978-3-031-38747-0_12. URL: https://doi.org/10.1007/978-3-031-38747-0_12.

[7] Cathy Maugis, Gilles Celeux, and Marie-Laure Martin-Magniette. "Variable selection for clustering with Gaussian mixture models". In: *Biometrics* 65.3 (2009), pp. 701–709.

[8] Leland McInnes, John Healy, and Steve Astels. "hdbscan: Hierarchical density based clustering". In: *The Journal of Open Source Software* 2.11 (2017), p. 205.

[9] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[10] Leland McInnes et al. "UMAP: Uniform Manifold Approximation and Projection". In: *Journal of Open Source Software* 3.29 (2018), p. 861. DOI: 10.21105/joss.00861. URL: https://doi.org/10.21105/joss.00861.

[11] Yanchi Liu et al. "Understanding of Internal Clustering Validation Measures". In: Dec. 2010, pp. 911–916. DOI: 10.1109/ICDM.2010.35.

[12] Alexander Lashkov, Sergey Rubinsky, and Polina Eistrikh-Heller. *s-dbw: Python implementation of the S_Dbw cluster validity index.* https://pypi.org/project/s-dbw/. Version 0.4.0. Accessed July 24, 2025. 2019.

[13] Davoud Moulavi et al. “Density-Based Clustering Validation”. In: *Proceedings of the 2014 SIAM International Conference on Data Mining (SDM)*, pp. 839–847. DOI: 10.1137/1.9781611973440.96. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.96. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611973440.96.

[14] Felix Siebert. *Fast Density-Based Clustering Validation (DBCV)*. https://github.com/FelSiq/DBCV. GitHub repository, accessed July 24, 2025. 2020.

[15] Justin Hart. *heuslerene-explorer*. https://github.com/HexagonalExpanse/heuslerene-explorer. GitHub repository, accessed July 24, 2025. 2025.

[16] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly.